НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

_комп'ютерних наук_____

(назва кафедри)

_____ _Голуб Б.Л.,доц.,к.т.н._

(підпис) (ПІБ)

"__04__" червня 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

«Програмне забезпечення автоматизованої системи по контролю успішності учнів школи»

Спеціальність 121 – «Інженерія програмного забезпечення»

ОПП - «Інженерія програмного забезпечення»

Гарант освітньої програми

_____к.т.н., доцент_____ _____ _____Вайганг Г.О.

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

_____ _____ _____ _____Панкратьєв В.О

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконала _____ _ _____ Мироненко Д.С._____

(підпис) (ПІБ)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук

_____/ Голуб Б.Л., доцент, к.т.н /
підпис

" 16 " грудня 2024 р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту
Мироненко Дар'ї Сергіївні

Спеціальність 121 – «Інженерія програмного забезпечення»

1. Тема бакалаврської кваліфікаційної роботи «Програмне забезпечення автоматизованої системи по контролю успішності учнів школи»

Затверджена наказом ректора НУБіП України №2248"С" від "16" грудня 2024 р.

2. Термін подання завершеної роботи на кафедру 2025 . 05 . 25
рік, місяць, число

3. Вихідні дані до бакалаврської кваліфікаційної роботи
Опис програмного забезпечення,

4. Перелік питань що розглядається:
   1. Системний аналіз предметної області інформаційної системи
   2. Проєктування інформаційного та програмного забезпечення
   3. Розробка інформаційного та програмного забезпечення
   4. Рекомендації щодо впровадження та експлуатації системи
   5. Висновки

Керівник бакалаврської кваліфікаційної роботи_____/ В.О. Панкратьєв /
підпис ініціали та прізвище

Завдання прийняв до виконання_____/ Д.С. Мироненко/
підпис ініціали та прізвище

Дата отримання завдання 2024 . 12 . 16
Рік місяць, число

ЗМІСТ

ЗМІСТ 3

ВСТУП 5

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 7

1.1 7 Постановка задачі

1.2 9 Моделювання предметної області

1.3 12 Діаграма прецедентів

1.4 17 Діаграма активності

1.5 Діаграма послідовності 19

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 24

2.1 Логічна модель даних у вигляді ER-діаграми 24

2.2 30 Діаграма класів та кооперацій

2.3 36 Діаграма пакетів

2.4 Діаграма компонентів 38

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 44

3.1 Система управління базою даних 44

3.2 Створення бази даних 45

3.3 Вибір інструментарію для створення прикладного програмного забезпечення 49

3.4 Принцип роботи у середовищі візуальної розробки програм 50

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ 54

4.1 Тестування системи 54

4.2 Вимоги до апаратного та програмного забезпечення 55

4.3 Опис роботи програми 58

ВИСНОВКИ 68

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 69

ДОДАТОК А 70

ДОДАТОК Б 75

ДОДАТОК В 85

ВСТУП

Актуальність. Освітній процес у загальноосвітніх закладах є фундаментом для формування особистості та майбутнього розвитку суспільства. В умовах стрімкої цифровізації та глобалізації, школа стоїть перед викликом оптимізації внутрішніх процесів, зокрема тих, що стосуються контролю успішності учнів. Традиційні методи обліку та моніторингу навчальних досягнень часто є трудомісткими, схильними до помилок та вимагають значних часових витрат з боку педагогічного складу та адміністрації. Збільшення обсягів інформації про успішність учнів, включаючи оцінки з різних предметів, відвідуваність уроків та результати контрольних робіт, актуалізує потребу у впровадженні сучасних інформаційних технологій.

Значну частину робочого часу вчителів займає рутинна робота з оформлення журналів, виставлення оцінок, підготовки звітів та формування аналітичних даних щодо успішності. Цей процес може бути спрощений та підвищений у точності завдяки автоматизації. Впровадження автоматизованої системи контролю успішності учнів дозволяє зменшити часові витрати на обробку даних, мінімізувати ймовірність помилок при формуванні звітності, а також покращити оперативний контроль та доступ до актуальної інформації про навчальні досягнення кожного школяра. Ключову роль у цьому відіграє централізована база даних, яка забезпечує надійне зберігання та швидкий доступ до всієї необхідної інформації.

Сучасна школа потребує простого, надійного та зручного у використанні програмного забезпечення для ефективного обліку успішності та оцінки навчальних досягнень учнів. Така система має вирішувати комплекс завдань: спростити роботу з реєстрами учнів, навчальних предметів, уроків та оцінок; забезпечити формування різноманітної звітної документації для вчителів та адміністрації, батьків; а також надати інструменти для аналізу динаміки успішності та виявлення проблемних зон. Саме тому виникла необхідність у

розробці системи, яка дозволить максимально спростити та автоматизувати процес контролю успішності, звільнивши час педагогів для безпосередньої освітньої діяльності.

Основною метою даної бакалаврської кваліфікаційної роботи є розробка програмного забезпечення автоматизованої системи для контролю успішності учнів школи. Система повинна забезпечити зручне ведення реєстру учнів, предметів, уроків та оцінок, а також надати інструменти для ефективного формування звітності та аналізу навчальних досягнень.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Дослідити предметну область контролю успішності учнів у загальноосвітніх закладах та визначити ключові процеси й інформаційні потоки.
2. Проаналізувати наявні програмні рішення для автоматизації обліку успішності у шкільному середовищі.
3. Сформулювати функціональні та нефункціональні вимоги до проектованої системи.
4. Побудувати UML-діаграми (прецедентів, послідовності, активності, класів) для візуалізації архітектури та логіки роботи системи.
5. Обрати оптимальні інструменти та технології розробки.
6. Розробити інтуїтивно зрозумілий інтерфейс користувача, що забезпечує зручність та доступність для всіх категорій користувачів.
7. Спроектувати та реалізувати структуру реляційної бази даних на основі Microsoft SQL.
8. Розробити програмний продукт, що відповідає всім функціональним вимогам.
9. Провести комплексне тестування розробленої системи.

Кінцевий програмний продукт дозволить ефективно зберігати, обробляти, редагувати, переглядати та швидко знаходити потрібну інформацію про учнів, їхні навчальні досягнення з різних предметів, а також формувати необхідні

звіти. Це дозволить максимально спростити роботу вчителів та адміністрації, надаючи їм потрібну інформацію в зручному форматі в будь-який момент.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка задачі

Розробка автоматизованої системи контролю успішності учнів школи спрямована на оптимізацію та цифровізацію процесів обліку, моніторингу та аналізу навчальних досягнень школярів. Система покликана забезпечити ефективну взаємодію між вчителями, адміністрацією школи та батьками, сприяючи прозорості освітнього процесу, спрощенню ведення шкільної документації та підвищенню оперативності доступу до актуальної інформації про успішність кожного учня.

Основні цілі створення системи:

1. Оптимізація роботи педагогічного та адміністративного персоналу шляхом автоматизації рутинних операцій з обліку оцінок та відвідуваності.
2. Формування єдиної централізованої бази даних, що містить інформацію про учнів, навчальні предмети, розклад уроків та їхні оцінки.
3. Спрощення процесів додавання, редагування та перегляду даних про навчальні досягнення учнів.
4. Надання інструментів для генерації різноманітної звітної документації щодо успішності учнів для внутрішнього використання та для батьків.
5. Забезпечення достовірності та актуальності інформації про навчальні результати.

Наразі в освітніх закладах часто спостерігаються труднощі, зумовлені відсутністю уніфікованої автоматизованої системи обліку успішності. Це призводить до необхідності ручного заповнення журналів, численних копій документів, а також значних часових витрат на складання звітів та проведення аналізу навчальних досягнень. Велика кількість однотипних операцій, таких як

виставлення та коригування оцінок створює додаткове навантаження на вчителів.

Представлена автоматизована система не ставить за мету повну заміну всіх адміністративних процесів, проте її впровадження суттєво покращить ведення обліку успішності та управління навчальним процесом. Вона надасть користувачам зручний інтерфейс та комплекс інструментів для щоденної роботи, зменшуючи ймовірність людських помилок та підвищуючи ефективність освітнього менеджменту.

Функціональні вимоги до системи:

1. Облік учнів, навчальних предметів, уроків та оцінок з можливістю додавання, редагування та видалення даних.
2. Формування та експорт звітної документації щодо успішності учнів у поширених форматах, таких як .docx або .pdf.
3. Розмежування прав доступу до функціоналу системи відповідно до ролей користувачів (вчитель, завуч, директор).
4. Пошук і перегляд даних і інформації про учнів, клас, предмет, уроки, оцінки).

Нефункціональні вимоги до системи:

1. Безпека: забезпечення конфіденційності та цілісності даних шляхом реалізації механізмів авторизації, автентифікації користувачів та розмежування прав доступу.
2. Надійність: гарантування стабільної та безперебійної роботи системи, включаючи можливість резервного копіювання та відновлення даних у разі потреби.
3. Масштабованість: здатність системи до подальшого розширення функціоналу та збільшення обсягу оброблюваних даних без суттєвої втрати продуктивності.
4. Зручність використання: наявність інтуїтивно зрозумілого інтерфейсу користувача та легкої навігації, що мінімізує час на освоєння системи.

5. Легкість обслуговування: прозора архітектура програмного забезпечення, що забезпечує легкість внесення змін, оновлень та розширення функціоналу.

Вся інформація в системі зберігається у централізованій базі даних, що дозволяє отримувати актуальні дані у реальному часі. Реалізація багаторівневого захисту за допомогою авторизації та розмежування прав доступу гарантує безпеку конфіденційних даних. Програмне забезпечення розроблено з урахуванням максимальної зручності використання, простоти навчання для кінцевих користувачів та максимальної адаптації до їхніх потреб.

1.2 Моделювання предметної області

Моделювання предметної області є інтегральною складовою процесу системного аналізу та проєктування інформаційних систем. На цьому етапі відбувається детальне вивчення та формалізація реальних процесів та сутностей, які будуть автоматизовані розроблюваним програмним забезпеченням. Ефективне моделювання вимагає глибокого розуміння як технічних аспектів майбутньої системи, так і специфіки функціонування освітнього середовища, зокрема, процесів, пов'язаних з обліком успішності учнів.

Предметна область, що розглядається в рамках даної дипломної роботи, охоплює всі елементи, взаємозв'язки та операції, що стосуються контролю та моніторингу навчальних досягнень школярів. Точне та адекватне моделювання цієї області є критично важливим для успішної розробки програмного забезпечення, оскільки будь-які неточності або пропущені вимоги на початкових етапах можуть призвести до значних витрат на їх виправлення на пізніших стадіях життєвого циклу проєкту.

Модель предметної області в контексті інформаційних систем являє собою абстраговане представлення реального світу, що відображає ключові концепції,

їхні атрибути та взаємозв'язки. Вона слугує своєрідним "словником" для розробників, допомагаючи їм краще зрозуміти вимоги до бази даних, логіки функціонування системи та інтерфейсу користувача.

Для побудови якісних та зрозумілих моделей предметної області широко використовується Уніфікована мова моделювання (UML – Unified Modeling Language). UML є стандартизованою графічною нотацією, що дозволяє візуалізувати, проєктувати та документувати різні аспекти програмних систем. Застосування UML сприяє покращенню комунікації між усіма зацікавленими сторонами проєкту та забезпечує єдине розуміння архітектури та функціоналу майбутньої системи. UML-діаграми дозволяють описувати як статичну структуру системи (за допомогою діаграм класів), так і її динамічну поведінку (використовуючи діаграми прецедентів, послідовності, активності).

У процесі створення автоматизованої системи по контролю успішності учнів школи, модель предметної області набуває центрального значення для ідентифікації фундаментальних сутностей та визначення зв'язків між ними. Первісний аналіз предметної області, проведений згідно з поставленими завданнями бакалаврської кваліфікаційної роботи, дав змогу виокремити такі ключові сутності, що формують ядро функціонування системи:

1. **Учень:** Основний об'єкт обліку, що характеризується персональними даними (ПІБ, дата народження, клас) та пов'язаний зі своїми оцінками та відвідуваністю.
2. **Предмет:** Навчальна дисципліна, що має назву та пов'язана з конкретними вчителями, уроками та оцінками учнів.
3. **Урок:** Представляє собою конкретне заняття з певного предмета у певному класі, що має дату, тип заняття та може містити інформацію про оцінки отримані на цьому уроці.

4. **Оцінка:** Числове значення, що відображає рівень навчальних досягнень учня з певного предмета за конкретний урок або період. Містить інформацію про дату, предмет, урок.
5. **Вчитель:** Особа, відповідальна за викладання певних предметів та виставлення оцінок. Має ідентифікаційні дані та пов'язаний з предметами та класами.
6. **Клас:** Група учнів, об'єднаних за віковими та навчальними критеріями, що має назву та пов'язаний зі списком учнів та предметами.

Для візуалізації статичної структури предметної області програмного забезпечення для автоматизованої системи по контролю успішності учнів школи була використана діаграма класів UML. Ця діаграма наочно відображає перелічені вище сутності, їхні атрибути та взаємозв'язки. Зокрема, вона демонструє зв'язок типу "один-до-багатьох" між сутностями "Клас" та "Учень", вказуючи на те, що один клас може містити множину учнів. Аналогічно, реалізовано зв'язок "багато-до-багатьох" між сутностями "Вчитель" та "Предмет", що відображає можливість викладання декількох предметів одним вчителем та викладання одного предмета декількома вчителями.

Діаграми активності UML були застосовані для моделювання ключових бізнес-процесів системи. Прикладами таких процесів є: створення уроку і додавання нової оцінки. Діаграми послідовності використовувалися для деталізації взаємодії між користувачами та системою при виконанні конкретних функціональних вимог.

Використання UML стало ключовим для візуалізації архітектури системи на початкових етапах розробки, а також для формування спільного розуміння предметної області між розробником та потенційними користувачами. Ця єдність у баченні є фундаментальною для успішного проєктування та впровадження автоматизованої системи по контролю успішності учнів.

При створенні моделі предметної області критично важливим є дотримання кількох ключових методологічних підходів. По-перше, принцип абстракції вимагає зосередження виключно на суттєвих характеристиках предметної області, що безпосередньо впливають на функціонування системи обліку успішності, мінімізуючи надмірну деталізацію. Це сприяє підвищенню чіткості та лаконічності моделі. По-друге, принцип декомпозиції передбачає поділ складної системи на менші, більш керовані та зрозумілі компоненти. Наприклад, функціональність, пов'язана з управлінням даними учнів, може бути логічно відокремлена від функціональності управління оцінками. Такий підхід значно спрощує процес розробки та подальшого супроводу системи. По-третє, принцип інкапсуляції полягає у приховуванні внутрішньої логіки об'єктів та наданні зовнішнім користувачам лише необхідного інтерфейсу для взаємодії. Це підвищує модульність системи та її гнучкість до модифікацій.

Використання формальних методів моделювання предметної області, зокрема UML, забезпечує чітку та структуровану документацію системи. UML-діаграми не лише візуалізують поточний стан проєкту, але й слугують надійною основою для подальших етапів життєвого циклу розробки програмного забезпечення, включаючи проєктування бази даних, кодування, тестування та підтримку. Детальна модель полегшує розуміння архітектури системи, значно спрощує внесення змін та розширення функціоналу в майбутньому, тим самим мінімізуючи потенційні ризики та оптимізуючи загальний процес розробки.

Підсумовуючи, доцільно стверджувати, що моделювання предметної області має фундаментальне значення для успішного створення програмного забезпечення автоматизованої системи контролю успішності учнів школи. Цей процес не обмежується лише технічною реалізацією, а є комплексним інтелектуальним завданням, що передбачає глибоке осмислення специфіки освітнього процесу та обліку успішності, а також налагоджену комунікацію між усіма зацікавленими сторонами, такими як розробники, вчителі та шкільна

адміністрація. Вкладення ресурсів у ретельне моделювання суттєво зменшує ризики проєкту, підвищує його прозорість та забезпечує розробку програмного продукту, що адекватно відповідає функціональним потребам користувачів і ефективно автоматизує контроль навчальних досягнень школярів.

1.3 Діаграма прецедентів

Діаграма прецедентів (або діаграма варіантів використання, англ. *Use Case Diagram*) є одним з фундаментальних інструментів Уніфікованої мови моделювання (UML). Її призначення полягає в описі функціональних можливостей програмної системи з позиції її кінцевих користувачів. Цей вид діаграм активно застосовується на початкових стадіях проєктування для виявлення, структурування та візуалізації вимог до системи, сприяючи досягненню спільного розуміння її потенціалу серед розробників, замовників та інших зацікавлених сторін.

Ключові елементи діаграми прецедентів:

1. **Межа системи (System Boundary):** Відображається у вигляді прямокутника, що охоплює всі варіанти використання. Елементи, розташовані за межами цієї границі, не є частиною функціоналу, що реалізується програмним забезпеченням.
2. **Актор (Actor):** Являє собою будь-яку зовнішню сутність, що взаємодіє із системою. Це може бути користувач, інша інформаційна система або технічний пристрій. Актори графічно позначаються у вигляді стилізованих фігурок людини. Один актор може бути залучений до декількох варіантів використання, і навпаки — один варіант використання може бути доступний для взаємодії з кількома акторами.
3. **Варіант використання (Use Case):** Це функціональна одиниця, що описує конкретну взаємодію актора із системою, спрямовану на досягнення певної цілі. Кожен варіант використання позначається еліпсом і має лаконічну назву, наприклад, "Перегляд звітної інформації", "Додавання оцінок учнів".

4. **Сценарії (Scenarios):** Кожен варіант використання може бути деталізований за допомогою сценаріїв, що представляють собою конкретну послідовність дій під час взаємодії актора із системою. Сценарії поділяються на основні (що описують успішне виконання задачі) та альтернативні (що враховують можливі помилки або виняткові ситуації).
5. **Зв'язки (Relationships):**
    - **Асоціація (Association):** Відображає взаємодію між актором та варіантом використання.
    - **Include (Включення):** Вказує на те, що один варіант використання обов'язково включає поведінку іншого. Це означає, що функціональність, описана в одному прецеденті, є невід'ємною частиною іншого. Наприклад, "Управління інформацією про класи" може обов'язково включати "Додавання інформації про учнів.
    - **Extend (Розширення):** Використовується для опису опціональних, розширених або умовних сценаріїв. Такий зв'язок показує, що один варіант використання може бути роширений іншим, якщо виконуються певні умови.

Призначення та переваги діаграм прецедентів:

Основна мета діаграми варіантів використання полягає у формуванні єдиного функціонального бачення системи. Вона є ефективним інструментом комунікації між усіма учасниками розробки: розробниками, бізнес-аналітиками, дизайнерами, замовниками та кінцевими користувачами. Цей вид діаграм дозволяє:

- чітко визначити що саме система повинна робити (на противагу тому, як саме це буде реалізовано);
- акцентувати увагу на поведінці системи з позиції користувача;

- Уникнути передчасного заглиблення у технічні деталі на початкових стадіях проєктування;
- сформувати міцну основу для подальшого деталізованого проєктування, включаючи розробку діаграм класів, активності або послідовностей.

Діаграми прецедентів є потужним методом для структурованого опису функціональних потреб програмного забезпечення. Їхня візуальна ясність і легкість розуміння роблять їх ідеальними для початкового вивчення вимог, демонстрації загальної архітектури системи та продуктивного обговорення її функціональності в команді розробників. Використовуючи такі компоненти, як дійові особи (актори), сценарії використання та взаємозв'язки, можна чітко та однозначно представити можливості та обмеження системи.

На основі розробленої діаграми прецедентів, що представлена на рис. 1.1 "Діаграма прецедентів ", були ідентифіковані наступні ключові актори, кожен з яких взаємодіє із системою для виконання визначеного набору функцій:

- **Директор**: Цей актор, як керівна особа навчального закладу, має можливість перегляду звітної інформації. Це дозволяє йому отримувати узагальнені аналітичні дані про навчальні досягнення учнів, загальний стан успішності у школі, що є важливим для прийняття стратегічних рішень та оцінки ефективності освітнього процесу.
- **Завуч**: Відповідаючи за організацію навчально-виховного процесу, Завуч володіє широким спектром адміністративних функцій. Його взаємодія із системою охоплює:
    - **Управління інформацією про предмети**: дозволяє Завучу додавати, редагувати дані про навчальні дисципліни, що викладаються у школі.
    - **Управління інформацією про вчителів**: включає функції обліку, додавання та редагування персональних даних викладацького складу.

- **Управління інформацією про класи**: надає можливість створювати і керувати даними про навчальні класи.
- **Додавання та редагування інформації про учнів**: надає можливість додавати нових учнів в систему та для подальшого коригування існуючих даних про них.
- **Перегляд звітної інформації**: дозволяє Завучу отримувати доступ до аналогічних звітів, що доступні Директору.

- **Вчитель**: Ключовий актор, який безпосередньо взаємодіє з системою для фіксації та моніторингу навчального процесу. Функціональність, доступна Вчителю, включає:
  - **Створення уроків**: Можливість фіксації проведених занять, вказавши дату, тип заняття, предмет.
  - **Додавання оцінок учнів за уроки**: Основна функція, що дозволяє Вчителю вносити бали учнів за конкретні уроки.

- **Перегляд звітної інформації**: Дозволяє Вчителю отримувати доступ до аналогічних звітів, що доступні Директору і Завучу.

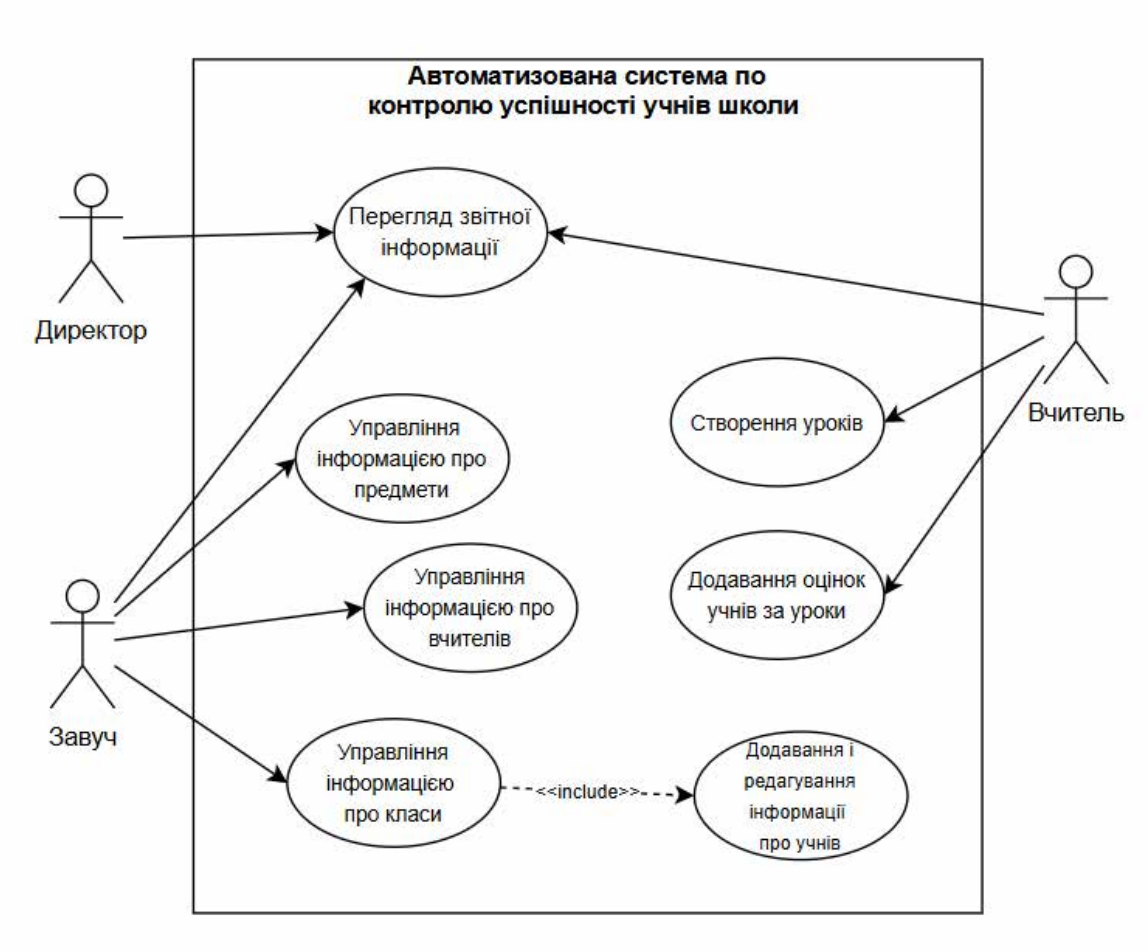


Рис. 1.1 Діаграма прецедентів

Визначені актори для даної предметної області системи представлені в таблиці 1.1.

Таблиця 1.1

Актор	**Короткий опис**
Директор	Особа, яка здійснює загальне управління системою, має повноваження переглядати звітну інформацію та контролювати діяльність вчителів та завучів.

Закінчення таблиці 1.1

Завуч	Особа, що організовує та координує навчальний процес, взаємодіє з вчителями та керує інформацією про предмети, вчителів та класи, учнів.
Вчитель	Особа, що безпосередньо взаємодіє з учнями та системою для створення уроків, додавання оцінок за уроки.

1.4 Діаграма активності

Діаграма активності є одним з найбільш ефективних та наочних засобів для моделювання динамічної поведінки системи. Вона дозволяє візуалізувати послідовність операцій, що відбуваються в рамках певного процесу, та виявити потенційні недоліки ще до етапу впровадження програмного рішення. Будучи частиною уніфікованої мови моделювання UML, ці діаграми забезпечують єдине та однозначне розуміння між різними учасниками розробки: аналітиками, програмістами та замовниками.

Хоча зовнішньо діаграма активності може нагадувати традиційну блок-схему, її ключова перевага полягає у здатності відображати не лише лінійну послідовність дій, але й паралельне виконання процесів, що є критично важливим для архітектури сучасних складних інформаційних систем. Такі діаграми особливо зручні для ілюстрації комплексної логіки, що включає паралелізм або множинні альтернативні шляхи виконання.

Ключовим елементом діаграми активності є власне "активність" – стан системи, під час якого виконується конкретна дія. Після завершення цієї дії система переходить до наступної активності. Переходи можуть бути зумовлені певними умовами, зовнішніми подіями або значеннями параметрів. На діаграмі також використовуються спеціальні позначення для початку та завершення процесу, точки розгалуження та об'єднання потоків, що дозволяє повноцінно охопити різноманітні бізнес-сценарії.

Ці функціональні можливості роблять діаграми активності незамінним інструментом при плануванні автоматизації бізнес-процесів. Вони дозволяють

зручно представити процеси у вигляді алгоритмів, за якими функціонуватиме програмне забезпечення, зокрема, для автоматизованої системи контролю успішності учнів.

На етапі аналізу критично важливо точно визначити кожну дію в рамках процесу, відповідальних осіб та необхідні ресурси. Розробник повинен мати глибоке розуміння операційної логіки. Недостатнє осягнення процесів може призвести до створення неефективного або неповного програмного продукту, що спричинить додаткові витрати або навіть провал проєкту.

Таким чином, діаграма активності – це значно більше, ніж просто візуалізація. Вона є фундаментальним інструментом для системного аналізу, глибокого розуміння та проєктування поведінки програмних рішень. Її використання сприяє глибшому проникненню в бізнес-логіку, дозволяє заздалегідь виявити потенційні проблеми, оптимізувати робочі процеси та, як результат, створити більш продуктивну систему.

На діаграмі активності, яка зображена на рис. 1.2, зображено послідовність дій для актора "Вчитель" у системі контролю успішності учнів. Процес починається з входу вчителя до системи через інтерфейс користувача. Після успішної авторизації вчитель може паралельно ініціювати дві незалежні дії: "Створення уроку" або "Додавання оцінки". Після завершення однієї з цих дій або обох, система перевіряє коректність виконаної операції. У випадку виникнення помилки, система генерує "Повідомлення про помилку" і повертає вчителя до початку процесу. Якщо ж дія успішна, відбувається "Збереження даних". Далі вчитель може "Запитати звітну інформацію". Аналогічно, якщо при цьому запиті виникає помилка, виводиться "Повідомлення про помилку", і система повертає вчителя до етапу повторного запиту звіту. У разі успішного запиту, Вчитель переглядає звіт, і даний процес завершується.

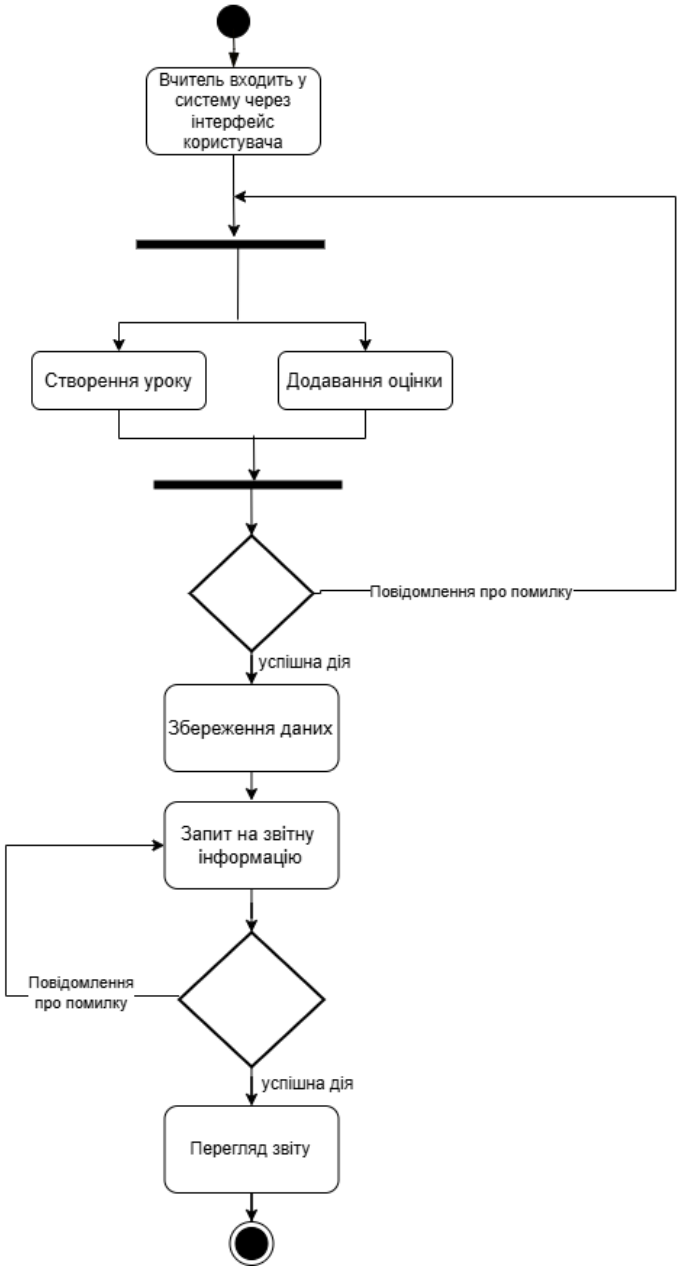


Рис.1.2 Процес додавання оцінок учнів (діаграма активності).

1.5 Діаграма послідовності

Діаграма послідовності (англ. *Sequence Diagram*) є одним із видів діаграм UML, призначеним для ілюстрації хронологічного порядку взаємодії між компонентами програмного комплексу під час виконання певного функціонального сценарію. Цей інструмент дає змогу наочно представити динаміку системи, показуючи, як її об'єкти обмінюються повідомленнями у часовій перспективі.

На таких діаграмах візуалізуються як зовнішні користувачі (актори), так і внутрішні модулі системи, між якими відбувається обмін інформацією для досягнення поставленої мети. Це сприяє чіткому визначенню задіяних об'єктів у реалізації сценарію, послідовності їхніх дій та характеру передаваної інформації.

Ключовою перевагою діаграм послідовності є можливість виявити логіку взаємодії елементів на етапі проєктування, до початку безпосередньої розробки. Це слугує надійною основою для архітектури майбутніх програмних класів, методів та обробників подій.

Центральним елементом діаграми є вертикальна часова вісь, що дозволяє визначити порядок викликів: повідомлення, розташоване вище, було передано раніше. Кожен учасник взаємодії має "лінію життя" — вертикальну пунктирну лінію, що позначає період активності об'єкта протягом сценарію. Обмін інформацією між об'єктами відображається горизонтальними стрілками, що символізують виклики методів або передачу даних. Моменти активності об'єкта підкреслюються "фокусом управління" (activation bar) — видовженим прямокутником, який вказує на період виконання об'єктом певної дії.

Таким чином, діаграма послідовності є потужним інструментом для моделювання, що дає змогу візуалізувати поведінку системи, виявити потенційні невідповідності у логіці взаємодій та краще підготуватися до реалізації програмного забезпечення.

В Додатку А (сторінка 1) представлена діаграма послідовності, розроблена для візуалізації взаємодії об'єктів у контексті функціонування автоматизованої системи по контролю успішності учнів школи.

Актори діаграми:

1. **Директор** — керівник навчального закладу, який ініціює певні процеси та отримує звіти про успішність та діяльність системи.
2. **Завуч** — адміністративний співробітник, який взаємодіє з системою для управління навчальним процесом, включаючи дані про вчителів, класи та предмети, учнів.

3. **Вчитель** — педагогічний працівник, який взаємодіє з системою для додавання уроків, оцінок та перегляду звітності.

Системні компоненти:

1. **Система** — програмна платформа, що координує взаємодію між акторами та сховищем даних.
2. **БД (База даних)** — централізоване сховище інформації, з яким відбувається обмін даними для збереження та отримання.

Основні процеси взаємодії:

1. **Управління інформацією (додавання/редагування)**:
    - Процес розпочинається з Завуча, який ініціює операцію додавання або редагування даних про вчителів, предмети, класи або учнів.
    - Система обробляє запит, виконує перевірку коректності введених даних.
    - Після перевірки, дані передаються до Бази Даних для збереження.
    - База Даних виконує операції збереження змін.
    - Система отримує підтвердження від БД і повертає Завучу повідомлення про успішне збереження інформації.
2. **Створення уроків та додавання оцінок:**
    - Вчитель ініціює створення нового уроку або додавання оцінок учням за урок.
    - Система перевіряє коректність наданих даних.
    - При успішній перевірці, дані зберігаються в Базі Даних.
    - Система підтверджує Вчителю успішне додавання інформації.
3. **Перегляд звітної інформації:**
    - Актори, такі як Завуч, Директор або Вчитель, можуть запросити звітну інформацію.
    - Система формує запит до Бази Даних.
    - База Даних виконує запит і повертає необхідні дані.
    - Система обробляє та відображає звіт для відповідного актора.

Особливості взаємодії:

1. **Послідовність взаємодії:** Діаграма наочно демонструє часову послідовність обміну повідомленнями між користувачами, системою та базою даних.
2. **Валідація даних:** Вбудовані процеси перевірки коректності введених даних гарантують цілісність інформації перед її збереженням.
3. **Зворотний зв'язок:** Система надає користувачам підтвердження про успішне виконання операцій, підвищуючи довіру до системи.
4. **Розмежування доступу:** Різні актори мають специфічні права та можливості взаємодії з функціоналом системи та даними, відповідно до їхньої ролі.

Ця діаграма послідовності деталізує кроки взаємодії між користувачами, програмним комплексом та базою даних, ілюструючи порядок дій та обмін інформацією, що є необхідним для ефективної розробки програмних компонентів.

Висновок до розділу 1

У результаті проведеного системного аналізу предметної області було детально вивчено специфіку функціонування загальноосвітнього навчального закладу з погляду обліку та моніторингу академічної успішності учнів.

Визначено основні проблеми, з якими стикаються учасники освітнього процесу, зокрема: фрагментованість даних, відсутність єдиного цифрового середовища для обліку результатів, складність формування звітності та обмеженість у доступі до актуальної інформації для різних ролей користувачів.

У процесі аналізу були окреслені ключові ролі користувачів майбутньої системи — директор, заступник директора (завуч), вчитель та учень. Для кожної ролі визначено функціональні обов’язки, очікувані дії в межах системи та специфічні потреби в автоматизації. Це дозволило сформувати початковий перелік функціональних вимог до системи. Поряд із цим сформульовано нефункціональні вимоги, серед яких: безпечність доступу, інтуїтивність, тощо.

Крім того, побудовані діаграми UML — зокрема, діаграма прецедентів, діаграма активності та діаграма послідовності — дозволили візуально описати базові сценарії взаємодії користувачів із системою та уточнити логіку реалізації ключових процесів. Це забезпечило глибше розуміння функціональних потоків, допомогло уникнути суперечностей на наступних етапах проєктування та створило основу для побудови цілісної архітектури програмного забезпечення.

Таким чином, проведений аналіз предметної області та підготовка концептуальних моделей створили методично обґрунтоване підґрунтя для подальшої розробки автоматизованої системи, що буде ефективно підтримувати навчальний процес, підвищувати прозорість оцінювання та сприяти цифровій трансформації освітнього середовища.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Концептуальне проєктування архітектури даних є фундаментальним етапом у розробці будь-якої інформаційної системи, зокрема й для автоматизованої системи контролю успішності учнів. У цьому контексті ключову роль відіграє ER-модель (Entity-Relationship Model), або модель сутність-зв'язок. Вона є потужним семантичним інструментом, що дозволяє візуалізувати та структурувати предметну область у вигляді набору взаємопов'язаних сутностей, їхніх характеристик (атрибутів) та встановлених між ними зв'язків. Ця універсальна методологія успішно застосовується для дизайну різноманітних типів баз даних, включаючи реляційні, ієрархічні, мережеві та об'єктно-орієнтовані сховища даних.

Застосування ER-моделювання на початкових стадіях проєктування, особливо для об'ємних баз даних, істотно мінімізує ризики виникнення архітектурних помилок. Такі дефекти, якщо їх не виявити на ранніх етапах, можуть призвести до значних витрат часу, фінансових ресурсів та зусиль на їхнє усунення під час тестування або, що гірше, після запуску системи в експлуатацію. Неадекватно спроєктована структура бази даних часто вимагає суттєвої переробки залежного програмного коду, що негативно позначається на загальній ефективності проєкту.

ER-модель представляє логічну структуру бази даних у графічному форматі, відображаючи ключові сутності предметної області, їхні атрибути та взаємозв'язки. Важливо підкреслити, що це концептуальний рівень моделювання, який абстрагується від специфічних деталей реалізації. Одна й та сама логічна модель може бути втілена у різних фізичних архітектурах бази даних, залежно від обраної СУБД, технічних вимог та операційного середовища.

Програмне забезпечення для проєктування ER-діаграм

Для створення ER-діаграм у професійному середовищі широко використовуються CASE-засоби (Computer-Aided Software Engineering). Одним з провідних рішень є **CA ERwin Data Modeler** (раніше відомий як ERwin). Цей інструмент надає розширені можливості для проєктування, документування та підтримки баз даних, а також сховищ і вітрин даних. Моделі, створені за допомогою ERwin, забезпечують візуальне представлення складної структури даних, сприяючи ефективному управлінню, організації та адмініструванню різних аспектів корпоративної інформаційної інфраструктури, зокрема з урахуванням об'єму даних, технологій баз даних та середовищ розгортання.

ERwin значно спрощує процес проєктування баз даних, дозволяючи розробникам створювати інтуїтивно зрозумілі графічні моделі "сутність-зв'язок" (ER-діаграми). Ці діаграми детально відображають усі елементи даних та логіку їхніх взаємозв'язків. Для побудови повноцінної логічної моделі достатньо ввести бізнес-правила, що відображають специфіку конкретної предметної області, після чого система автоматично генерує відповідну структуру, включаючи атрибути, зв'язки та їхні угруповання.

Окрім стандартного функціоналу, ERwin пропонує можливість додавання користувацьких властивостей, що забезпечує високий ступінь адаптації моделі до унікальних потреб певної галузі або проєкту. Ця гнучкість робить інструмент надзвичайно зручним для ретельного документування складних структур даних, підвищуючи продуктивність командної роботи та подальшу ефективність реалізації бази даних.

Створення ER-діаграми (сутнісно-зв'язкової моделі) є критично важливим етапом у проєктуванні бази даних для автоматизованої системи контролю успішності учнів школи з кількох причин:

1. **Формалізація структури даних:** ER-діаграма дозволяє чітко визначити основні інформаційні сутності системи (наприклад, "Учні", "Вчителі",

"Предмети", "Оцінки", "Уроки", "Класи" тощо) та встановити логічні зв'язки між ними. Це забезпечує правильне та послідовне структурування майбутньої бази даних.

2. **Мінімізація помилок на етапі впровадження:** Детальне та однозначне уявлення про зв'язки між сутностями допомагає уникнути надлишкового дублювання інформації або порушення цілісності даних, що може призвести до системних збоїв або некоректної роботи програмного забезпечення.
3. **Забезпечення масштабованості системи:** Завдяки ER-діаграмі значно легше ідентифікувати частини системи, які можуть бути розширені або модифіковані в майбутньому (наприклад, додавання функціоналу для електронних журналів, аналітики успішності, комунікації з батьками), без порушення базової логіки та цілісності архітектури.

Таким чином, застосування ER-моделювання на етапі проєктування гарантує побудову міцної, логічно обґрунтованої та масштабованої структури даних, що є запорукою успішної розробки та подальшого функціонування автоматизованої системи.

Для забезпечення ефективного функціонування автоматизованої системи контролю успішності учнів школи була розроблена логічна модель даних, що включає декілька основних сутностей, кожна з яких відображає важливі аспекти предметної області. Графічне представлення цієї структури відображено на рисунку 2.1, де зображена ER-діаграма системи.

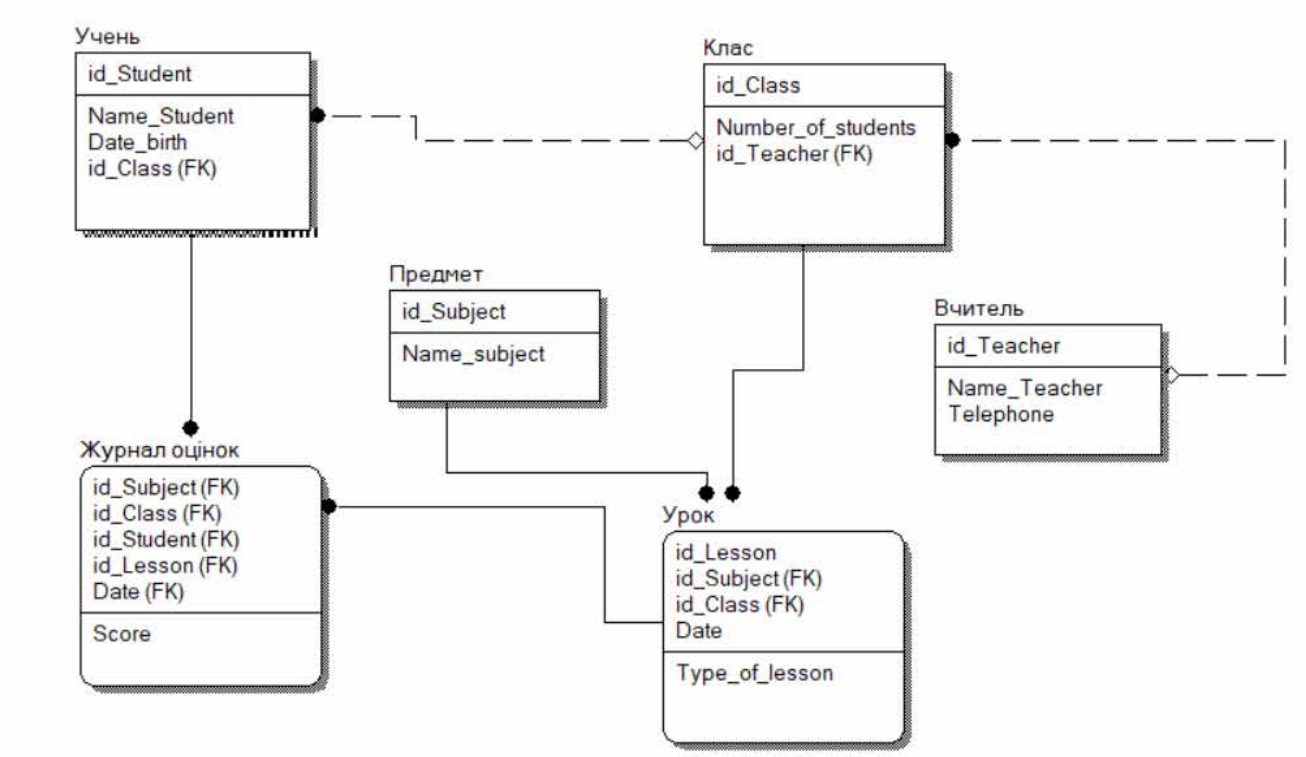


Рис. 2.1 ER-діаграма

Опис ключових сутностей системи

1. **Учен:** ця сутність призначена для зберігання персоналізованої інформації про школярів. Вона містить унікальний ідентифікатор учня (id_Student), його повне ім'я (Name_Student), дату народження (Date_birth) та посилання на клас, до якого належить учень (id_Class як зовнішній ключ). Взаємозв'язок між учнем і класом є "багато-до-одного", тобто багато учнів можуть належати до одного класу.
2. **Клас**: дана сутність є організаційною одиницею в системі, яка об'єднує учнів. Вона включає унікальний ідентифікатор класу (id_Class), кількість учнів у класі (Number_of_students) та зовнішній ключ до вчителя, який є класним керівником (id_Teacher). Зв'язок між класом і вчителем є "багато-до-одного", що дозволяє одному вчителю керувати декількома класами, хоча в контексті класу це вказує на конкретного класного керівника.
3. **Вчитель**: ця сутність зберігає дані про педагогічний склад школи. Вона містить унікальний ідентифікатор вчителя (id_Teacher), його ім'я

(Name_Teacher) та контактний телефон (Telephone). Вчитель може бути асоційований з одним або декількома класами як класний керівник, а також вести різні предмети.

4. **Предмет**: сутність "Предмет" описує навчальні дисципліни, які викладаються у школі. Вона включає унікальний ідентифікатор предмета (id_Subject) та його назву (Name_subject). Предмети пов'язані з уроками та оцінками, відображаючи навчальний процес.
5. **Урок**: ця сутність моделює проведення конкретних навчальних занять. Вона містить унікальний ідентифікатор уроку (id_Lesson), зовнішні ключі до предмета (id_Subject), класу (id_Class) та вчителя, що проводить урок (через опосередкований зв'язок з Класом), дату проведення уроку (Date) та його тип (Type_of_lesson). Кожен урок належить до певного класу, предмета та проводиться у визначений день.
6. **Журнал оцінок**: сутність "Журнал оцінок" є центральним елементом для обліку успішності учнів. Вона містить зовнішні ключі до предмета (id_Subject), класу (id_Class), учня (id_Student) та уроку (id_Lesson), а також дату отримання оцінки (Date) та саму оцінку (Score). Ця сутність фіксує зв'язок "багато-до-багатьох" між учнями, предметами, класами та уроками, дозволяючи реєструвати оцінки за конкретні уроки для конкретних учнів у певних класах.

Аналіз моделі на відповідність вимогам нормалізації баз даних (3НФ)

Третя нормальна форма (3НФ) є важливою для забезпечення мінімізації дублювання даних та покращення структури бази даних. Для того, щоб модель була в 3НФ, вона повинна задовольняти такі умови:

1. Вона повинна бути в 2НФ.
2. Не повинно бути транзитивних залежностей. Тобто, не повинно бути таких випадків, коли одна таблиця залежить від іншої через третю таблицю.

Перша нормальна форма (1НФ): Усі таблиці в моделі містять атомарні значення. Кожне поле зберігає лише одне значення, без списків чи вкладених

структур. Наприклад, у сутності "Учень" кожний атрибут (наприклад, Name_Student, Date_birth) є неподільним. Повторювані групи відсутні у всіх сутностях.

Друга нормальна форма (2НФ): Модель не містить часткових функціональних залежностей неключових атрибутів від частини складеного ключа. У сутності "Журнал оцінок", де складений ключ може формуватися з комбінації зовнішніх ключів (наприклад, id_Subject, id_Class, id_Student, id_Lesson), всі неключові атрибути (наприклад, Score) повністю залежать від усього складеного ключа. Інші таблиці мають прості первинні ключі, і всі їхні атрибути безпосередньо залежать від цих ключів.

Третя нормальна форма (3НФ): У системі не виявлено транзитивних залежностей, що означає, що всі неключові атрибути залежать безпосередньо від первинного ключа своєї таблиці, а не від інших неключових атрибутів:

- Наприклад, у сутності "Учень" атрибути Name_Student та Date_birth безпосередньо залежать від id_Student, а не від id_Class.
- У сутності "Клас" атрибут Number_of_students безпосередньо залежить від id_Class, а не від id_Teacher.
- У сутності "Вчитель" атрибут Telephone безпосередньо залежить від id_Teacher, а не від Name_Teacher.
- У сутності "Журнал оцінок" атрибут Score залежить безпосередньо від комбінованого ключа, а не від інших полів у цій же таблиці, таких як Date.

Загалом, структура бази даних відповідає принципам **третьої нормальної форми (3НФ)**, оскільки:

- Кожен атрибут належить одній логічній сутності.
- Відсутні зайві залежності між неключовими атрибутами.
- Забезпечена логічна організація та цілісність даних.

Отже, побудована логічна модель бази даних є чітко структурованою, узгодженою з вимогами предметної області автоматизованої системи контролю успішності учнів школи, і повністю відповідає третій нормальній формі. Це гарантує ефективну роботу системи, спрощує її підтримку та забезпечує можливість легкої адаптації до майбутніх функціональних змін.

2.2 Діаграма класів та кооперацій

Діаграма класів є фундаментальним компонентом у мові уніфікованого моделювання (UML), що забезпечує статичне, структурне представлення архітектури програмної системи. Вона слугує для візуалізації основних будівельних блоків системи — класів, а також їхніх атрибутів (властивостей) та операцій (методів). Завдяки цій діаграмі можна чітко розмежувати, які дані інкапсулює кожен клас і які функціональні можливості він надає.

Окрім визначення внутрішньої структури класів, діаграма класів ефективно ілюструє різноманітні типи взаємозв'язків між ними. До цих зв'язків належать асоціації (загальний зв'язок), агрегації (зв'язок "частина-ціле" з незалежним життєвим циклом компонентів), композиції (зв'язок "частина-ціле" із залежним життєвим циклом компонентів) та успадкування (відношення "є різновидом"). Таке графічне відображення сприяє глибокому розумінню архітектури системи як окремими розробниками, так і всією командою проєкту.

Діаграма класів також виконує роль потужного інструменту на етапі безпосередньої розробки, слугуючи свого роду кресленням або шаблоном для написання програмного коду. Вона структурує процес створення класів та реалізації їхньої взаємодії, роблячи його більш послідовним та передбачуваним.

Цей тип діаграм відіграє ключову роль у систематизації логіки взаємодії об'єктів, що є незамінним для створення повної системної документації та ефективного планування тестування, забезпечуючи високу якість кінцевого програмного продукту.

Створена діаграма класів, розміщена в Додатку А (сторінка 2), детально ілюструє архітектуру інформаційної системи для управління навчальним процесом у школі. Вона охоплює ключові ролі учасників, їхні функціональні обов'язки та логічні взаємозв'язки між об'єктами системи. До основних класів моделі належать: "Учень", "Клас", "Вчитель", "Директор", "Завуч", "Предмет", "Урок", "Журнал оцінок", "Звіт". Ці класи спільно забезпечують функціональність обліку учнів, фіксації успішності та управління освітніми ресурсами.

Таблиця основних класів та їх функціональності наведена нижче в табл. 2.2.

Таблиця 2.2

Таблиця основних класів та їх функціональності

№	Клас	Основні властивості	Ключові методи
1	Директор	ID	Авторизація, перегляд звітів
2	Завуч	ID	Авторизація, перегляд звітів, додавання учнів, створення класів, додавання предметів, додавання вчителів
3	Вчитель	ID, ПІБ, телефон	Авторизація, перегляд звітів, створення уроків, додавання оцінок
4	Предмет	ID, назва	- (сутність, що описує предмет)
5	Урок	ID, дата і час, клас, предмет, тип заняття	- (сутність, що описує урок)
6	Звіт	Назва, дата, тип, текст звіту	- (сутність, що описує звіт)
7	Клас	ID, назва, список учнів, список уроків	Перегляд списку учнів і уроків для цього класу
8	Журнал оцінок	ID, дата, урок, учень, предмет, оцінка	- (сутність, що описує оцінку)
9	Учень	ID, ПІБ, клас, оцінки	Отримання оцінок

Першим ключовим класом виступає **«Директор»**, який виконує функцію вищого адміністратора системи. Його обов’язки зводяться до проходження авторизації та ознайомлення зі звітною інформацією, сформованою

педагогічним складом. Наявність лише одного атрибута — ID — підкреслює, що директор виконує радше контролюючу, ніж операційну функцію.

Наступним за значущістю є клас **«Заступник директора» (Завуч)**, що відповідає за організаційно-навчальну діяльність у системі. До функціоналу цього класу належать операції з управління списком предметів і викладачів, створення нових класів, додавання учнів, а також проходження авторизації. Завуч тісно взаємодіє з класами «Клас», «Вчитель», «Предмет» та «Учень», виступаючи координатором навчального процесу.

Клас «Вчитель» є центральною фігурою освітнього середовища. Його атрибути охоплюють унікальний ідентифікатор, ПІБ та номер телефону. Вчитель має широкий спектр можливостей: створення та редагування уроків, виставлення оцінок, перегляд звітів. Саме через цей клас реалізується більшість освітніх дій у системі.

Окрему увагу привертає клас **«Урок»**, який відображає окрему навчальну одиницю. Він містить інформацію про дату, час проведення, клас, предмет і тип заняття. Кожен урок створюється вчителем і пов’язаний із певною навчальною дисципліною, що забезпечує структурованість розкладу.

Далі розглядається клас **«Журнал оцінок»**, що відіграє роль проміжного ланцюга між уроками та учнями. Його структура включає дату, урок, ПІБ учня та виставлену оцінку. Саме завдяки цьому класу реалізується механізм фіксації академічної успішності в системі.

Клас «Учень» представляє основного користувача освітнього процесу. До його атрибутів належать ID, ПІБ, клас та список отриманих оцінок. Метод отримання оцінок забезпечує перегляд індивідуальних результатів, а наявність зв’язку з класом і журналом дозволяє формувати повну картину навчального прогресу.

Важливою складовою є і **клас «Клас»**, що об’єднує учнів у групи за рівнем навчання. Серед його характеристик — ID, назва класу, списки учнів і уроків. Метод перегляду списку учнів забезпечує доступ до поточної структури класного колективу. Формування класів здійснюється заступником директора.

Клас **«Предмет»** зберігає базову інформацію про навчальні дисципліни: унікальний ідентифікатор та назву предмета. Цей клас пов’язаний із уроками, а його зміст формується та оновлюється завучем.

Завершує структуру **клас «Звіт»**, який відповідає за узагальнення аналітичної інформації. До його полів належать назва звіту, дата формування, тип і текстовий зміст. Звіти створюються вчителями, а доступ до них мають керівники, що дозволяє адміністрації оперативно оцінювати загальну успішність у школі.

Діаграми кооперацій, зображені на рисунках 2.3-2.5, є одним із ключових засобів візуалізації динамічної поведінки об’єктно-орієнтованої системи. У контексті розробки програмного забезпечення для автоматизованої системи контролю успішності учнів вона слугує ілюстрацією того, як об’єкти системи (тобто конкретні екземпляри класів) взаємодіють у межах певного сценарію — наприклад, виставлення оцінки, створення звіту чи реєстрація нового класу.

Основна мета діаграми полягає у демонстрації послідовності повідомлень, що передаються між об’єктами, для досягнення конкретної функціональної мети. Така діаграма дозволяє аналітикам, архітекторам і розробникам отримати цілісне уявлення про динаміку внутрішньої логіки системи: хто з ким взаємодіє, у якій послідовності відбувається обмін інформацією, та які саме методи чи дії викликаються під час процесу.

Кожен елемент діаграми кооперації представляє окремий об’єкт, що є реалізацією відповідного класу, визначеного на діаграмі класів. Зв’язки між об’єктами зображаються лініями, які вказують на можливість взаємодії та іноді містять рольову характеристику — пояснення функції об’єкта у взаємодії. Обмін повідомленнями між об’єктами позначається стрілками, які не лише вказують напрямок передачі, а й супроводжуються номерами виклику, що дає змогу точно відстежити послідовність виконання операцій.

На відміну від діаграми класів, яка є статичною моделлю структури системи й охоплює всі класи, їхні атрибути та методи, діаграма кооперації фокусується виключно на одному конкретному випадку використання. Вона не

демонструє повної архітектури, а лише ту частину, яка є релевантною до вибраного сценарію. Важливо зазначити, що до діаграми включаються лише ті об’єкти, які безпосередньо задіяні в реалізації відповідного функціоналу — таким чином забезпечується чіткість і цільова спрямованість моделі.

Ще однією важливою особливістю є контекстозалежність — об’єкти можуть виконувати різні ролі в різних кооперaціях. Один і той самий об’єкт, залежно від сценарію, може ініціювати дію, бути її отримувачем або брати участь у логічному ланцюгу передачі інформації, виконуючи проміжну функцію.

У рамках системи контролю успішності учнів така діаграма є надзвичайно корисною для моделювання ключових освітніх процесів — наприклад, реєстрація нових учнів у класах, внесення оцінок у журнал та подальше формування аналітичних звітів. Завдяки своїй динамічній природі діаграма кооперації дозволяє глибше зануритися в логіку бізнес-процесів, забезпечуючи як високий рівень деталізації, так і зручність для перевірки правильності сценаріїв взаємодії.

Загалом, діаграма кооперації виступає невід’ємною частиною системного аналізу та проектування, дозволяючи перевірити узгодженість між об’єктами, підтвердити відповідність сценаріїв очікуваній поведінці системи та забезпечити ефективне документування складних взаємозв’язків у навчальному середовищі.

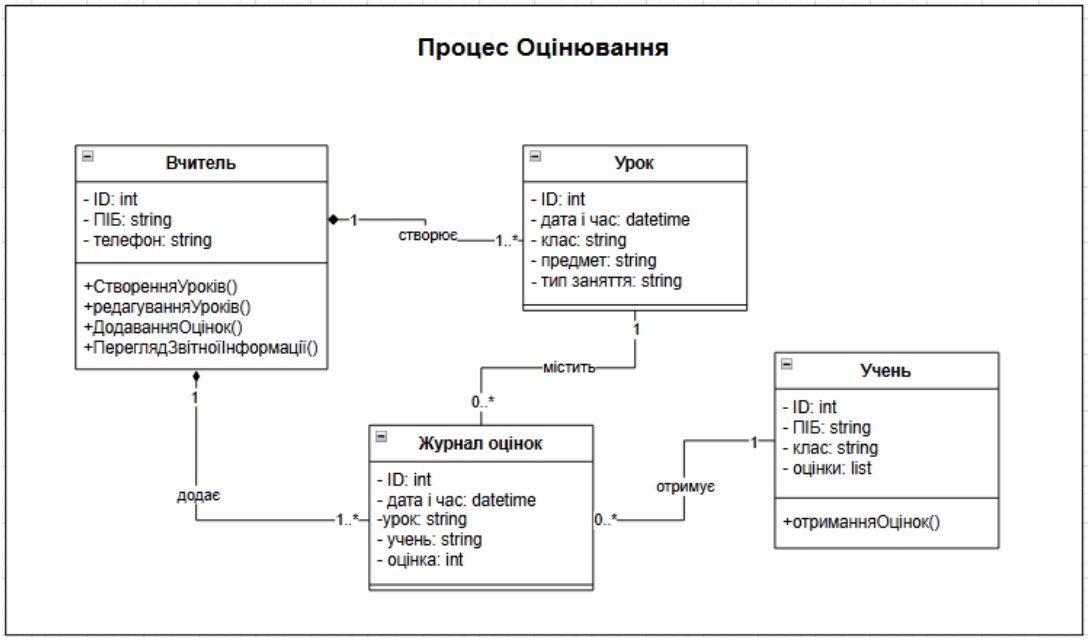


Рис. 2.3 Сценарій оцінювання учня

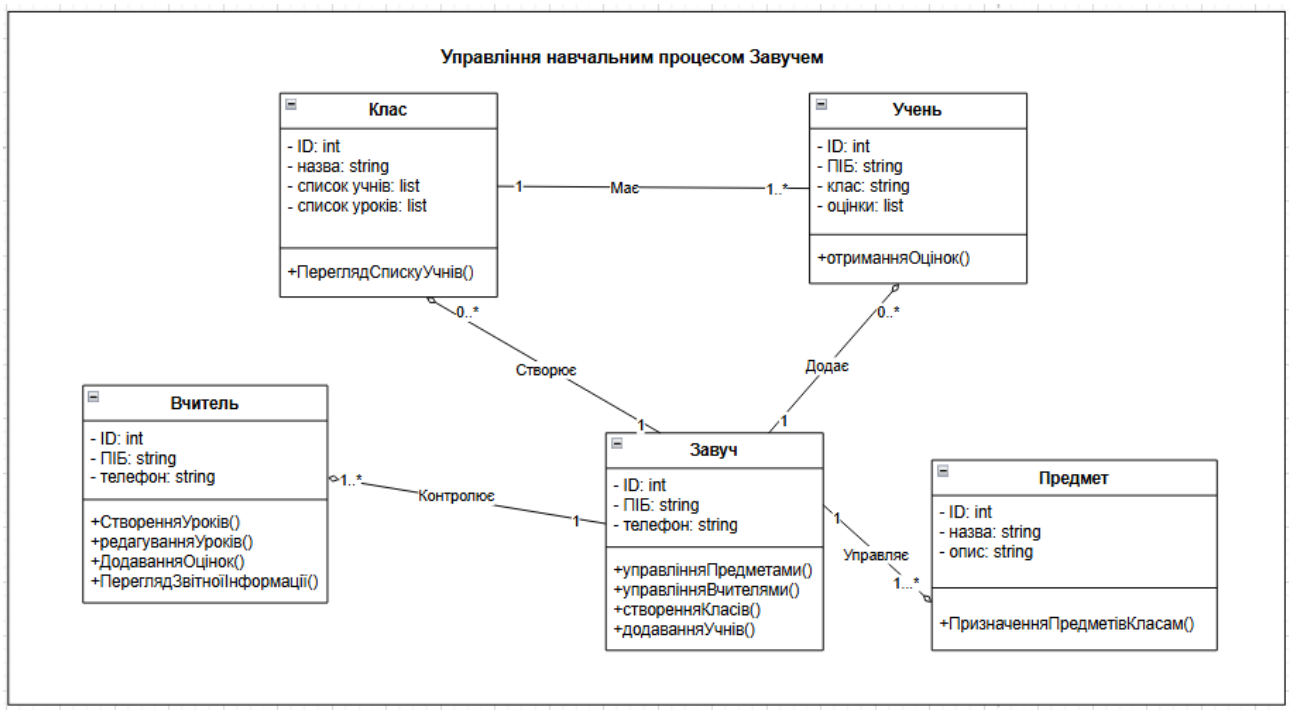


Рис. 2.4 Сценарій управління навчальним процесом Завучем

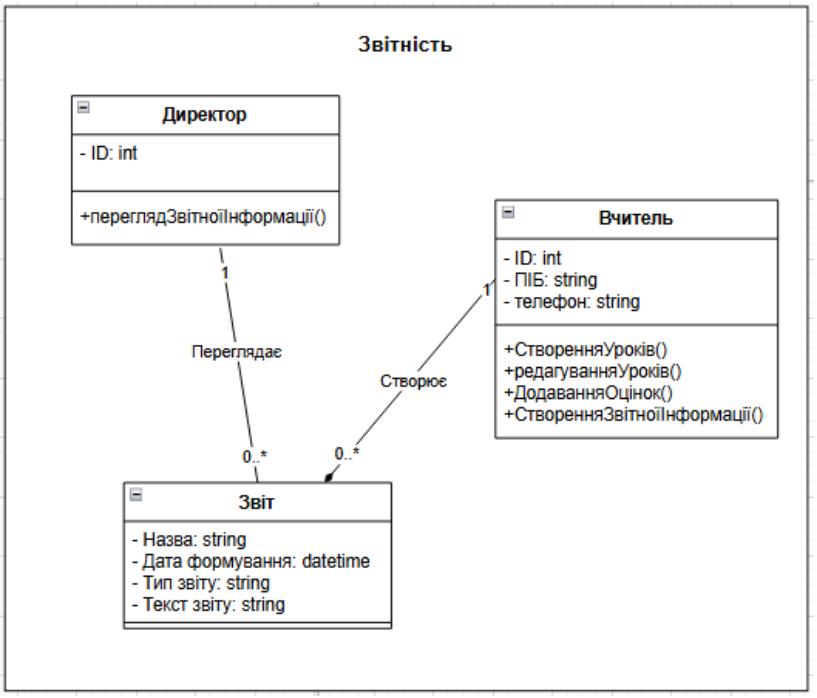


Рис. 2.5 Сценарій перегляду звітності

2.3 Діаграма пакетів

Для ефективної побудови архітектури програмного забезпечення було використано діаграму пакетів (package diagram), що є складовою уніфікованої мови моделювання UML. Такий тип діаграм дозволяє представити логічну структуру системи через групування пов’язаних елементів (класів, інтерфейсів, модулів) у функціональні блоки — пакети. Це дає змогу не лише спростити візуальне сприйняття системи, але й забезпечити модульність та структурну чіткість на етапах розробки, тестування і подальшої підтримки. На представленій діаграмі пакетів, яка розміщена у Додатку А (сторінка 3), зображено архітектурну організацію програмного забезпечення, розробленого для контролю успішності учнів у загальноосвітньому закладі. Вся система поділена на три рівні, кожен з яких виконує окрему роль у забезпеченні роботи інформаційної системи: презентаційний рівень, рівень логіки системи та рівень доступу до даних.

1. Презентаційний рівень

Цей рівень відповідає за взаємодію користувача із системою. Він містить модуль інтерфейсу користувача, через який директор, завуч, вчитель можуть

ініціювати певні дії, переглядати навчальну інформацію, вводити або редагувати дані. Компонент інтерфейсу передає запити до внутрішніх модулів логіки системи, що дозволяє розділити відповідальність між візуалізацією та бізнес-логікою. Наявність спрямованих зв’язків типу *use* до всіх внутрішніх підсистем підкреслює, що інтерфейс має доступ до всієї функціональності системи.

2. Рівень логіки системи

Цей шар є центральним ядром розробленого ПЗ і реалізує бізнес-правила та функціональність, пов’язану з навчальним процесом. Він охоплює низку модулів, кожен з яких відповідає за окремий напрям:

- **Управління учнями** — включає механізми додавання, редагування, перегляду списку учнів, їх оцінок та зв’язку з класами.
- **Управління вчителями** — дозволяє створювати й редагувати інформацію про викладачів, визначати їхні предмети, які вони викладають.
- **Управління уроками** — відповідає за формування і створення уроків, їх типу й змісту.
- **Управління оцінками** — забезпечує облік результатів навчання, дозволяє виставляти й переглядати оцінки.
- **Управління класами** — модуль організації учнів у навчальні групи, призначення класного керівника.
- **Управління предметами** — включає функції додавання нових дисциплін та редагування існуючих.
- **Звітність** — формує статистичні, індивідуальні й зведені звіти щодо успішності на основі даних, що надходять із інших модулів.

Взаємозв’язки між модулями реалізовані через відносини використання (*use*), що вказує на взаємозалежність функціональних блоків для досягнення цілісної роботи системи.

3. Рівень доступу до даних

Цей рівень забезпечує технічну реалізацію збереження, захисту та обробки даних. Його складовими є:

- **База даних** — централізоване сховище інформації про учнів, оцінки, класи, вчителів, предмети та звіти. Саме сюди надсилаються запити на збереження або отримання інформації з модулями бізнес-логіки.
- **API** — програмний інтерфейс, який надає доступ до деяких функцій або даних у структурований спосіб, зокрема для модулів звітності.
- **Сервіси безпеки** — відповідають за автентифікацію користувачів, контроль прав доступу. Вони інтегруються з усіма модулями логіки та презентують єдиний вхідний шлюз безпечної взаємодії.

Таким чином, запропонована структура дозволяє розділити систему на логічно завершені частини, кожна з яких виконує конкретну функцію, зберігаючи при цьому єдність загальної архітектури. Завдяки такому підходу до моделювання значно спрощується супровід, тестування та масштабування програмного забезпечення. Діаграма пакетів наочно демонструє принцип інкапсуляції, залежностей та узгодженої взаємодії між компонентами, що є критично важливим для успішної реалізації системи контролю успішності учнів у шкільному середовищі.

2.4 Діаграма компонентів

Діаграма компонентів у UML моделюванні використовується для відображення того, з яких логічних частин складається програмне забезпечення та як ці частини взаємодіють між собою. Цей тип діаграми не лише показує структуру системи, а й дозволяє описати архітектуру з погляду функціонального поділу: що саме виконує кожен окремий модуль, які інтерфейси використовуються для зв’язку між ними, та які залежності існують між компонентами.

На відміну від діаграм класів, що фокусуються на внутрішній структурі об'єктів і їхніх зв'язках, діаграма компонентів демонструє вищий рівень абстракції — архітектурну побудову системи. Вона дозволяє уявити програмне забезпечення як набір незалежних частин, що працюють разом як єдине ціле.

Кожен компонент розглядається як завершена логічна одиниця з чітко визначеною відповідальністю. Це зручно при плануванні розробки, розподілі задач між командами та забезпеченні повторного використання модулів у майбутніх проєктах.

У рамках розробки програмного забезпечення для автоматизованої системи контролю успішності учнів було реалізовано архітектуру з урахуванням принципів модульності, що дозволило побудувати діаграму компонентів, розміщену в Додатку А (сторінка 4), яка охоплює ключові підсистеми: інтерфейс користувача, авторизаційний блок, модулі логіки обробки даних, функціональні підсистеми взаємодії з даними та базу даних.

1. Інтерфейс користувача

Цей компонент відповідає за взаємодію кінцевого користувача з системою. У його складі передбачено підкомпоненти:

- **Авторизація** — забезпечує вхід у систему за допомогою логіна та пароля.
- **Взаємодія з формами** — реалізує подання запитів, введення та відображення навчальної інформації.

Цей компонент має залежності з іншими модулями, оскільки саме через нього ініціюється більшість дій: додавання учнів, редагування уроків, створення звітів тощо.

2. Система авторизації

Окремий функціональний блок, який відповідає за перевірку облікових даних користувачів. Складається з двох підкомпонентів:

- **Перевірка облікових даних** — здійснює автентифікацію.

- **Передача інформації про користувача** — забезпечує ідентифікацію авторизованого користувача для інших компонентів системи.

Даний компонент функціонує автономно, не взаємодіючи безпосередньо з бізнес-логікою, і забезпечує початкову точку доступу користувачів до системи.

3. Модулі логіки обробки даних

Центральна частина програмного забезпечення, яка містить основні компоненти, що реалізують бізнес-логіку системи:

- **Управління даними про учнів / вчителів / класи / предмети / уроки / оцінки** — реалізують функції додавання, редагування, пов’язування об'єктів між собою.
- **Формування звітності** — агрегує навчальні дані для створення звітів про успішність.

Кожен із цих компонентів має тісну залежність з відповідними модулями, що відповідають за інтерфейсну частину, а також з компонентом бази даних.

4. Модулі функціональної взаємодії

На цьому рівні представлені компоненти, що відповідають за виконання окремих дій у межах конкретної підсистеми:

- **Модуль роботи з учнями** — додає або редагує дані про учнів, а також здійснює прив'язку до класів;
- **модуль роботи з вчителями** — включає створення/редагування інформації про вчителів;
- **модуль роботи з класами** — дозволяє створювати, редагувати класи, додавати учнів і призначати класного керівника;
- **модуль роботи з предметами та уроками** — реалізує створення уроків, вибір предметів;
- **модуль роботи з оцінками** — передбачає виставлення та зміну балів;
- **модуль звітності** — забезпечує отримання зведених даних.

Ці компоненти мають чітко обмежені зони відповідальності, що дозволяє легко масштабувати або модифікувати систему без ризику порушити інші частини.

5. База даних

Фізичний рівень зберігання інформації. У структурі бази передбачено таблиці: Учні, Вчителі, Класи, Предмети, Уроки, Оцінки

Кожен компонент логіки обробки даних має прямий або опосередкований зв’язок із відповідними сутностями бази даних, що дозволяє здійснювати всі CRUD-операції (create, read, update, delete).

Загальний потік взаємодії в системі:

1. Користувач ініціює запит через інтерфейс (наприклад, додати оцінку);
2. Система авторизації перевіряє права доступу;
3. Запит надходить до відповідного функціонального модуля (наприклад, робота з оцінками);
4. Модуль взаємодіє з логікою обробки даних;
5. Відповідна операція над базою даних виконується через компоненти доступу;
6. Результат повертається користувачу через інтерфейс.

Така архітектурна організація забезпечує гнучке масштабування, можливість незалежного тестування модулів, повторне використання компонентів та централізоване управління даними. Поділ системи на окремі логічні блоки сприяє підвищенню надійності, ефективності обслуговування та прозорості реалізації навчальних процесів.

Висновки до розділу 2

У другому розділі було представлено результати проєктування архітектури програмного забезпечення системи контролю успішності учнів з використанням засобів моделювання уніфікованої мови UML. Візуалізація структури та логіки взаємодії між компонентами системи на ранньому етапі

розробки є вкрай важливою для забезпечення її цілісності, надійності та подальшої масштабованості.

Зокрема, діаграма класів дозволила відобразити предметну область у вигляді сутностей, їхніх атрибутів, методів та зв’язків. Це сприяло формуванню чіткої логічної моделі, що охоплює основні об’єкти системи — учнів, вчителів, адміністрацію, навчальні одиниці та оцінювання. Така структуризація допомогла окреслити функціональні обов’язки кожної ролі у системі та взаємозалежності між ними.

Діаграма кооперації дала змогу описати конкретні сценарії взаємодії об’єктів у межах визначених функціональних процесів: виставлення оцінки, створення звіту чи додавання учня до класу. Вона ілюструє не лише перелік об’єктів, задіяних у певній операції, але й послідовність та напрям обміну повідомленнями між ними, що особливо важливо для моделювання динаміки роботи системи.

Використання діаграми пакетів забезпечило логічне групування модулів за функціональним призначенням. Розподіл системи на презентаційний рівень, рівень бізнес-логіки та рівень доступу до даних дозволив створити багаторівневу архітектуру, де кожен пакет виконує строго визначену роль. Така структурна декомпозиція сприяє зручності навігації у коді, покращує читабельність і підтримуваність програмного забезпечення.

Діаграма компонентів, у свою чергу, надала можливість представити систему як сукупність незалежних, проте взаємопов’язаних функціональних блоків. Кожен компонент системи — від інтерфейсу користувача до модуля формування звітності — має чітко окреслену зону відповідальності та взаємодіє з іншими модулями через інтерфейси. Це дозволяє зменшити зв’язність між частинами системи, підвищити її гнучкість і забезпечити модульність архітектури.

Усі розглянуті діаграми UML є взаємодоповнюючими та сприяють комплексному розумінню і проєктуванню системи. Вони дозволяють не лише побудувати архітектуру майбутнього програмного продукту, а й виявити

потенційні проблеми ще до початку реалізації коду. Застосування таких підходів у розробці ПЗ для освітньої сфери, зокрема — для системи контролю успішності учнів, забезпечує створення надійного, структурованого і масштабованого рішення, що здатне ефективно підтримувати навчальний процес і вдосконалювати його на основі аналітичних даних..

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління базою даних

Система управління базами даних (СУБД) є критично важливим компонентом будь-якого сучасного програмного забезпечення, що передбачає зберігання, обробку та захист великого обсягу інформації. У контексті розробки автоматизованої системи контролю успішності учнів шкільного закладу СУБД забезпечує централізоване збереження структурованих даних про учнів, вчителів, класи, предмети, оцінки, звіти та інші елементи навчального процесу.

Основними функціями, які покладаються на СУБД, є надійне зберігання даних, забезпечення цілісності та несуперечності інформації, підтримка доступу на основі ролей користувачів, а також забезпечення безпечної взаємодії з даними у режимі реального часу. Завдяки цьому досягається не лише висока ефективність у роботі з навчальною інформацією, але й мінімізується ризик втрати або некоректного оновлення даних.

У межах даного проєкту як основну СУБД було обрано **Microsoft SQL Server** — реляційну систему управління базами даних, яка зарекомендувала себе як стабільне, функціональне та масштабоване рішення для корпоративного і освітнього програмного забезпечення. Обґрунтування вибору базується на низці переваг:

- **повна підтримка реляційної моделі** — дозволяє ефективно структурувати інформацію у вигляді таблиць з чіткими зв’язками між ними;
- **можливість реалізації складної бізнес-логіки** — за рахунок використання збережених процедур, тригерів та перевірок обмежень;

- **інтеграція з екосистемою Microsoft** — зокрема, з середовищем Visual Studio та .NET Framework, що забезпечує зручність під час розробки та налагодження;
- **потужна система контролю доступу та безпеки** — забезпечує автентифікацію, авторизацію та шифрування даних;
- **засоби резервного копіювання та відновлення** — гарантують захищеність критично важливої інформації навіть у разі технічних збоїв.

Усе це робить Microsoft SQL Server оптимальним вибором для реалізації системи контролю успішності в сучасному навчальному закладі, забезпечуючи ефективну та безпечну роботу всієї цифрової інфраструктури.

3.2 Створення бази даних

На етапі реалізації структури бази даних для системи контролю успішності учнів першочергово було виконано створення нової бази даних під назвою (Success Control Database), яка стала основним сховищем для всієї інформації, пов’язаної з навчальним процесом, користувачами та оцінюванням (на рис. 3.1 представлено SQL-код створення бази даних).

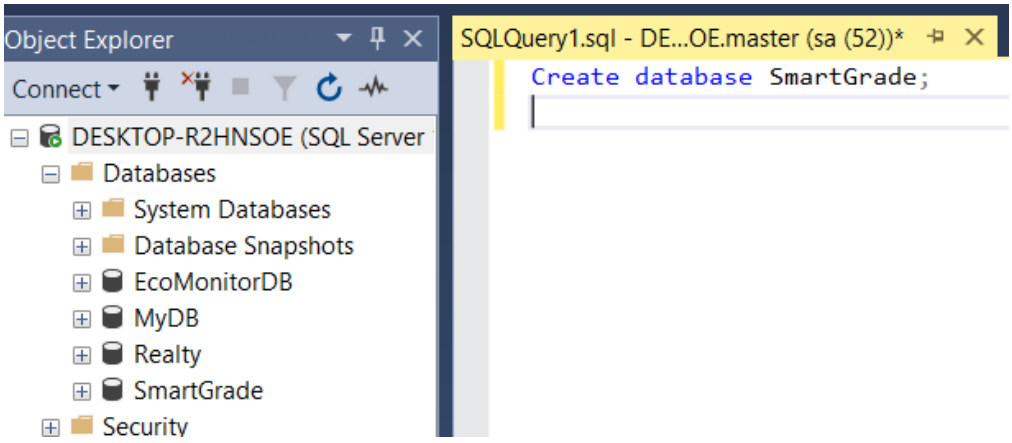
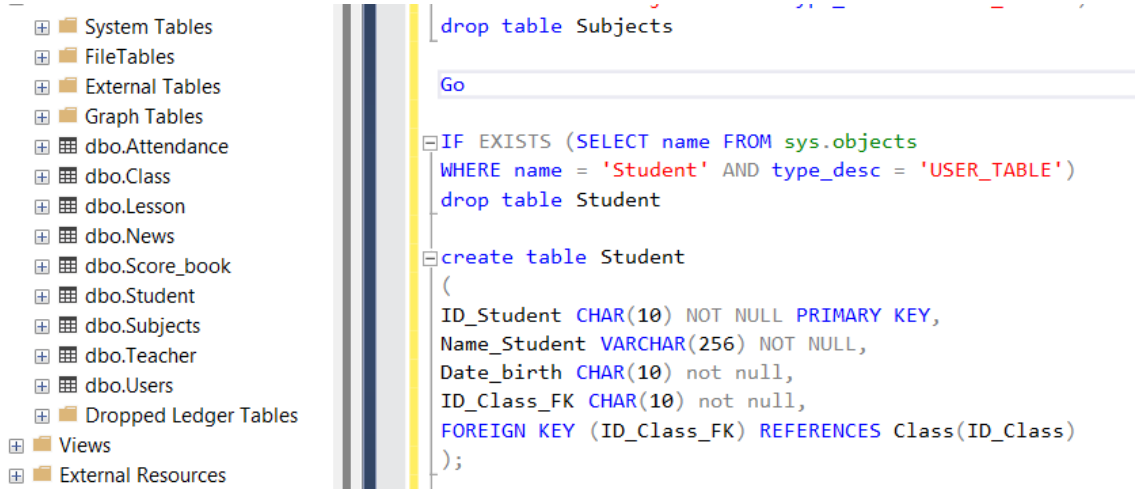


Рис. 3.1 Створення бази даних

Після ініціалізації бази даних було спроєктовано та реалізовано набір таблиць, які відповідають ключовим сутностям предметної області: **учні, вчителі, класи, предмети, уроки, оцінки** тощо. Таблиці створювалися із

застосуванням SQL-запитів, у яких було чітко вказано назви полів, їхні типи даних, обмеження цілісності (наприклад, первинні та зовнішні ключі), унікальність значень, а також каскадні обробки видалень і оновлень, де це було доцільно.



На рисунку 3.2 представлено фрагмент SQL-коду, який демонструє створення таблиці Students, призначеної для збереження інформації про учнів: ідентифікатора, ПІБ, дати народження, класу навчання. В Додатку Б (сторінки 1-2) розміщується повний лістинг коду створення таблиць бази даних.

Рис. 3.2 SQL-запит для створення таблиці Student

З метою забезпечення цілісності даних між сутностями було реалізовано логічні зв’язки через зовнішні ключі, що гарантує узгодженість інформації при операціях оновлення або видалення. Наприклад, оцінки учня не можуть існувати без запису про самого учня в таблиці Students, що регулюється зовнішнім ключем FOREIGN KEY.

Окрему увагу під час реалізації було приділено налаштуванню системи прав доступу, що є важливим аспектом безпеки в освітніх інформаційних системах. У середовищі Microsoft SQL Server було створено ролі для різних категорій користувачів: Director, Teacher, HeadTeacher. Для кожної ролі визначено чіткий набір дозволених дій (операцій SELECT, INSERT, UPDATE, DELETE) у відповідності до функціональних повноважень кожного типу користувача. Це забезпечує розмежування прав доступу, дозволяє уникати несанкціонованих змін та сприяє збереженню конфіденційності чутливої

інформації (на рисунку 3.3 представлено фрагмент SQL-коду налаштування ролей та прав доступу в базі даних).

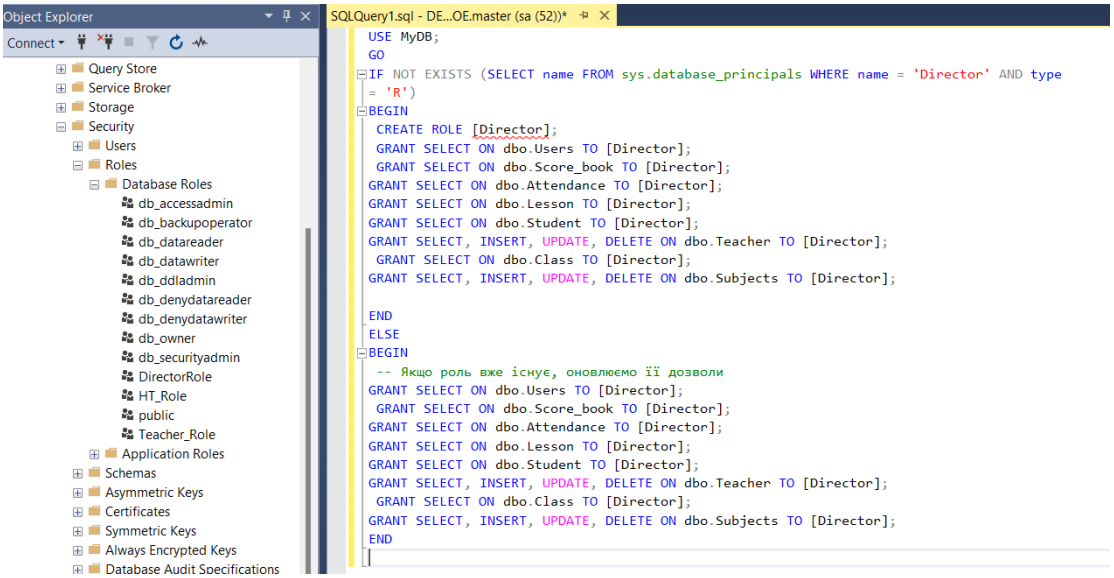


Рис. 3.3 Налаштування ролей та прав доступу в базі даних

В Додатку Б (сторінки 3-7) розміщується повний лістинг коду створення ролей і надання їм прав.

Окрім основної структури таблиць, у базі даних також було реалізовано механізм формування звітів на основі оцінок. Для цього створено три окремі уявлення (views) — це спеціальні об’єкти, які представляють собою збережені SQL-запити. Уявлення не зберігають дані безпосередньо, але дозволяють отримувати необхідну інформацію з однієї або кількох таблиць у зручному вигляді. Вони слугують логічним представленням даних, що часто використовуються, і дають змогу спростити доступ до складних структур, підвищити зручність роботи з базою та забезпечити певний рівень абстракції при обробці запитів.

Уявлення виконують агрегацію та фільтрацію даних відповідно до вимог системи контролю успішності. Вони дозволяють отримувати узагальнену

інформацію без необхідності звертатися до кількох таблиць одночасно, що значно спрощує побудову звітів і підвищує ефективність запитів.

У розробленій системі було створено три ключові уявлення, кожне з яких виконує специфічну функцію у контексті аналітики навчальних результатів:

1. Class_Averages — уявлення, призначене для розрахунку середнього балу по кожному класу. Воно об’єднує інформацію про клас, відповідального викладача та обчислює усереднене значення оцінок на основі журналу оцінювання. Це дозволяє адміністрації закладу освіти здійснювати швидку оцінку загальної динаміки успішності в розрізі класів.

2. Student_Ranking — уявлення, що формує рейтинг учнів відповідно до їхніх результатів з кожного навчального предмета. У запиті агрегуються оцінки, отримані учнями, з урахуванням предметної належності та класної приналежності. Завдяки цьому можливо не лише відстежувати академічну успішність на індивідуальному рівні, але й виявляти потенційні зони для педагогічної інтервенції.

3. Students_Average — уявлення, яке формує список учнів із розрахованим середнім балом на основі їхніх оцінок за всіма предметами. Основна його функція — виведення тих учнів, чий середній бал нижчий або дорівнює значенню, обраному користувачем із випадаючого списку в інтерфейсі. Такий підхід дозволяє швидко виявляти учнів, які потребують додаткової уваги або підтримки, і є зручним інструментом для моніторингу індивідуальної успішності.

На основі цих уявлень у системі реалізовано модулі побудови відповідних звітів, які доступні для користувачів із відповідними правами доступу. Це дозволяє дирекції та вчителям швидко отримувати актуальні аналітичні дані для прийняття управлінських рішень та індивідуальної роботи з учнями. У Додатку Б (сторінки 8-9) наведено SQL-коди створення зазначених уявлень.

Таким чином, реалізація бази даних охопила не лише технічне створення таблиць і встановлення зв’язків між ними, але й заклала основи для надійного, безпечного та контрольованого збереження інформації, що є критичним для стабільної роботи системи контролю успішності учнів у межах шкільної інформаційної екосистеми.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Одним із важливих етапів у розробці системи контролю успішності учнів є вибір мови програмування та середовища розробки. Це визначає не лише технічні можливості системи, але й ефективність її реалізації, підтримки та подальшого масштабування.

На початку були проаналізовані два варіанти — **C++** і **C#**. Хоча C++ надає розробнику широкий контроль над ресурсами та високу продуктивність, складність синтаксису, підвищена вимогливість до управління пам’яттю та відсутність зручних засобів для створення інтерфейсу роблять її менш придатною для розробки прикладного ПЗ освітнього призначення.

Натомість **C#** — об’єктно-орієнтована, сучасна мова програмування, створена компанією **Microsoft**, виявилася оптимальним вибором. Її сильні сторони: простий синтаксис, автоматичне управління пам’яттю, багатий набір бібліотек та висока сумісність із **Microsoft SQL Server**, який використовується як СУБД у проєкті.

Як середовище розробки було обрано **Microsoft Visual Studio**, що забезпечує повноцінну підтримку циклу розробки: написання коду, створення інтерфейсу, компіляцію, налагодження та роботу з базами даних. Особливо цінними для проєкту стали:

- візуальний дизайнер форм;
- підтримка IntelliSense (інтелектуальна система автозаповнення коду);
- інтеграція з Git (система керування версіями);

- зручні засоби для підключення та взаємодії з SQL Server.

Використання **C# у поєднанні з Visual Studio** дозволило швидко реалізувати графічний інтерфейс користувача, забезпечити доступ до бази даних, налаштувати авторизацію за ролями та створити зручну для користувачів архітектуру системи.

Отже, обраний інструментарій повністю відповідає вимогам проєкту, забезпечуючи стабільну, безпечну та гнучку основу для розробки прикладного програмного забезпечення системи контролю успішності учнів.

3.4 Принцип роботи у середовищі візуальної розробки програм

Одним із визначальних факторів у створенні прикладного програмного забезпечення є правильний вибір середовища розробки. Від обраного інструменту залежить ефективність реалізації проєкту, швидкість розробки, зручність налагодження, а також доступність функціональних можливостей, критичних для поставлених завдань. Для розробки системи контролю успішності учнів було проаналізовано переваги двох сучасних середовищ: JetBrains Rider та Microsoft Visual Studio.

JetBrains Rider — це сучасне багатоплатформне IDE, створене компанією JetBrains на основі IntelliJ-платформи, з глибокою інтеграцією інструментів ReSharper. Середовище підтримує розробку .NET-додатків мовою C#, а також проєкти на ASP.NET, Xamarin, Unity і Blazor. Rider вирізняється розвиненим механізмом аналізу коду, автоматичними підказками, рефакторингом, широкими можливостями кастомізації та підтримкою Git, Docker і баз даних.

Перевагою Rider є його кросплатформність: середовище може бути встановлене на Windows, macOS або Linux, що робить його зручним для командної розробки у змішаних середовищах. IDE також містить інструменти для роботи з вебтехнологіями, що важливо при інтеграції з онлайн-кабінетами учнів або формуванням вебзвітів.

Однак середовище Rider має низку особливостей, які слід враховувати:

- платна ліцензія (для навчальних закладів доступна безкоштовно, але з обмеженим терміном);
- відсутність повноцінного візуального редактора Windows Forms, що ускладнює створення десктопного GUI без ручного написання XAML або коду;
- більш високі системні вимоги порівняно з аналогами.

У свою чергу, **Microsoft Visual Studio** є флагманським середовищем для розробки .NET-додатків і надає повний інструментарій для побудови як логічної, так і візуальної частини прикладного ПЗ. Особливо цінною для реалізації проєкту стала підтримка Windows Forms — технології для швидкого створення графічного інтерфейсу з використанням кнопок, таблиць, полів введення та інших візуальних елементів.

Переваги Visual Studio:

- Вбудований візуальний редактор форм, що дозволяє створювати UI у форматі "drag-and-drop" (перетягування-вставлення);
- потужна система автодоповнення IntelliSense;
- нативна інтеграція з Microsoft SQL Server;
- вбудовані засоби налагодження, тестування та профілювання;
- безкоштовна версія Visual Studio Community для навчальних проєктів;
- повна підтримка Windows-платформи.

Це середовище особливо зручне для студентських і освітніх проєктів, оскільки дозволяє зосередитися не на налаштуванні інструментів, а безпосередньо на логіці реалізації функціоналу.

У рамках дипломного проєкту було прийнято рішення на користь Microsoft Visual Studio. Такий вибір обумовлений:

- необхідністю створення багатоформового графічного інтерфейсу;

- використанням C# як основної мови розробки;
- повною сумісністю із Microsoft SQL Server, що використовується як СУБД;
- наявністю готових шаблонів, що пришвидшують створення інтерфейсу для кожного типу користувача (директор, вчитель, завуч).

Середовище забезпечило стабільну реалізацію усіх компонентів: від авторизації до формування звітів, включно з підключенням до бази даних та обробкою дій користувача через обробники подій.

Висновок до розділу 3

У третьому розділі розглянуто практичні аспекти реалізації прикладного програмного забезпечення для автоматизованої системи контролю успішності учнів. На основі аналізу вимог до функціональності та структури даних було обґрунтовано вибір ключових технологічних рішень, необхідних для створення ефективної та зручної в експлуатації системи.

Першим етапом стала побудова структури бази даних на основі Microsoft SQL Server, яка забезпечила централізоване, надійне та структуроване збереження інформації про учнів, вчителів, класи, предмети, оцінки та звіти. Створення таблиць супроводжувалося впровадженням зовнішніх ключів, обмежень цілісності та налаштуванням прав доступу згідно з ролями користувачів.

Для розробки інтерфейсу та логіки системи було обрано мову C# та середовище Microsoft Visual Studio, що надало широкі можливості для створення Windows Forms-додатків, зручну інтеграцію з базою даних, генерацію обробників подій та засоби налагодження. Такий вибір дозволив реалізувати весь функціонал системи у стислий термін і з високою якістю.

Також було порівняно можливості Visual Studio та альтернативного середовища JetBrains Rider. Попри потужність Rider і його багатофункціональність, саме Visual Studio виявилося більш придатним для проєкту завдяки повній підтримці Windows Forms та глибокій інтеграції з Microsoft SQL Server.

Таким чином, результати, отримані у межах третього розділу, підтвердили обґрунтованість обраного технічного підходу до реалізації системи. У процесі розробки було сформовано логічно структуровану тришарову архітектуру: **рівень зберігання даних** (на базі Microsoft SQL Server), **рівень обробки логіки** (реалізований мовою C#) та **рівень користувацького інтерфейсу** (на основі Windows Forms). Така архітектура забезпечує чітке розмежування функціональних обов’язків між складовими системи, сприяє її масштабованості, зручному супроводу та надійній роботі в умовах реального освітнього процесу.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування є важливим етапом розробки прикладного програмного забезпечення, що дозволяє переконатися в коректності реалізованої функціональності, відповідності вимогам технічного завдання та готовності системи до практичного використання. У межах розробки автоматизованої системи контролю успішності учнів було проведено комплексне тестування основних функціональних блоків, з акцентом на стабільність, безпеку та зручність користування.

Тестування охоплювало як **функціональні**, так і **логічні сценарії** використання системи. Зокрема, перевірялась правильність роботи ключових модулів:

- **авторизація та автентифікація** — проведено перевірку процесу входу, реакції системи на помилкові дані та повідомлень про помилки;
- **розмежування доступу за ролями** — протестовано, чи обмежується функціонал відповідно до статуса користувача (директор, завуч, вчитель);
- **операції з даними** — виконано тестування додавання, редагування та видалення об'єктів (учнів, оцінок, предметів, класів) з коректними та некоректними вхідними даними;

- **збереження змін у базі даних** — перевірено, чи система коректно обробляє транзакції та відображає зміни в інтерфейсі;
- **завершення сесії** — протестовано стабільність виходу з системи та відсутність несанкціонованого доступу без повторної авторизації.

У процесі тестування були застосовані два основні підходи:

- **Метод «чорної скриньки»** — тестування поведінки системи з точки зору користувача без аналізу внутрішньої логіки коду;
- **метод «білої скриньки»** — перевірка коректності логіки обробки даних, SQL-запитів, обробки помилок та винятків.

Додатково проводилися:

- **тестування юзабіліті** — оцінювалася інтуїтивність інтерфейсу, зручність навігації, логічність розміщення елементів;
- **обмежене навантажувальне тестування** — перевірялася стійкість системи при збільшенні кількості записів у базі;
- **безпекове тестування** — моделювалися сценарії несанкціонованого доступу, перевірялися обмеження дій користувачів з різними рівнями прав.

Результати тестування свідчать, що програмне забезпечення виконує всі заплановані функції відповідно до вимог, а виявлені недоліки були оперативно усунуті. Система демонструє стабільну роботу, чітко дотримується логіки розмежування доступу, а її інтерфейс є зрозумілим і функціональним.

Таким чином, проведене тестування підтверджує технічну готовність системи до впровадження в навчальному середовищі та її практичну придатність для щоденного використання педагогами та адміністрацією школи.

4.2 Вимоги до апаратного та програмного забезпечення

Для коректного функціонування системи контролю успішності учнів необхідно врахувати як апаратну, так і програмну складову обчислювального

середовища, у якому буде розгортатись система. Програмне забезпечення реалізовано у форматі десктопного застосунку, що функціонує на основі клієнт-серверної архітектури: серверна частина забезпечує обробку запитів і зберігання інформації, а клієнтська — взаємодію з користувачем через графічний інтерфейс.

База даних, реалізована в середовищі Microsoft SQL Server, розміщується на сервері або локальній машині з відповідною продуктивністю. Клієнтські пристрої мають запускати програму, розроблену у Microsoft Visual Studio мовою C# з використанням технології Windows Forms.

Таблиця апаратних і програмних вимог до сервера наведена в таблиці 4.1 і таблиці 4.2

Таблиця 4.1

Апаратні вимоги для сервера

Ресурс	**Мінімальні вимоги**	**Рекомендовані параметри**
Процесор	1.0 ГГц	2.4 ГГц або вище
Оперативна пам’ять	2 ГБ	4 ГБ і вище
Жорсткий диск	40 ГБ	120 ГБ SSD
Мережева інфраструктура	100 Мбіт/с	Гігабітна мережа
Операційна система	Windows Server 2012 або новіше	Windows Server 2019 / 2022
Сервер баз даних	Microsoft SQL Server 2014 або новіше	Microsoft SQL Server 2019

Таблиця 4.2

Програмні вимоги для сервера

Ресурс	**Мінімальні вимоги**	**Рекомендовані характеристики**
Тип системи	IBM PC-сумісний	x64 архітектура
Процесор	Intel Pentium Dual-Core 1.6 ГГц	Intel Core i3 або вище
Оперативна пам’ять	2 ГБ	4–8 ГБ
Жорсткий диск	25 ГБ	60 ГБ вільного місця
Відеоадаптер	512 МБ	1 ГБ та підтримка DirectX
Монітор	1024×768	Full HD (1920×1080)

Закінчення таблиці 4.2

Операційна система	Windows 7/8/10/11	Windows 10 або 11
Периферія	Клавіатура, миша, принтер	+ сканер (за потреби)

Необхідне програмне забезпечення

Для забезпечення стабільної роботи прикладного програмного забезпечення системи контролю успішності учнів необхідно встановити низку допоміжних програмних компонентів, які забезпечують коректне виконання основних модулів та підтримку взаємодії з операційною системою та базою даних. До рекомендованого програмного забезпечення належать:

1. **Microsoft .NET Framework 4.7 або вище** — забезпечує виконання застосунку, реалізованого мовою C# у середовищі Visual Studio; є обов’язковим компонентом для запуску .NET-додатків.
2. **Microsoft Visual C++ Redistributable** — надає доступ до бібліотек, необхідних для функціонування системних компонентів і динамічних модулів, які використовуються в середовищі розробки**.**

3. **Microsoft Excel 2010 або пізніші версії** — використовується для формування, збереження та друку звітів, зокрема у форматі таблиць, що зручно для подальшого аналізу або архівування результатів.
4. **Антивірусне програмне забезпечення** з попереднім виключенням каталогу встановлення програми — для забезпечення захисту системи без впливу на продуктивність додатку, запобігаючи блокуванню критичних процесів**.**
5. **Microsoft SQL Server Management Studio (SSMS)** — клієнтський інструмент для обслуговування та моніторингу бази даних, застосовується адміністраторами для перегляду структур, керування доступом, резервного копіювання та аналізу запитів.

4.3 Опис роботи програми

Після запуску автоматизованої системи контролю успішності учнів користувач спершу бачить вікно початкового завантаження програми, зображене на рис. 4.1. Після завершення ініціалізації система автоматично перенаправляє користувача до форми входу, зображену на рис. 4.2, яка є точкою старту взаємодії з програмним забезпеченням.

Інтерфейс авторизації передбачає введення логіна та пароля, наданих користувачу відповідно до його ролі в системі. Для підвищення рівня безпеки доступ до функціоналу програми реалізовано через ролеву модель доступу. У системі передбачено три основні категорії користувачів:

- **директор** — має доступ до переглядів звітів, загальної інформації;
- **вчитель** — відповідає за створення уроків і заповнення оцінок;
- **завуч** – відповідає за додавання нових предметів, вчителів, створення класів.

Після успішної авторизації графічний інтерфейс автоматично адаптується відповідно до ролі користувача: відображаються лише ті елементи і розділи, які відповідають його правам у системі. Це дозволяє уникнути перевантаження інтерфейсу та забезпечити інтуїтивну навігацію.

Подальша робота із системою — зокрема керування класами, перегляд звітів чи створення уроків, тощо — здійснюється в межах відповідного функціонального модуля, структура яких наочно представлена на рисунках 4.3–4.16.

В Додатку В (сторінки 1 - 22) розміщується повний лістинг коду програми по основним функціям.

Рис. 4.1 Вікно завантаження програми

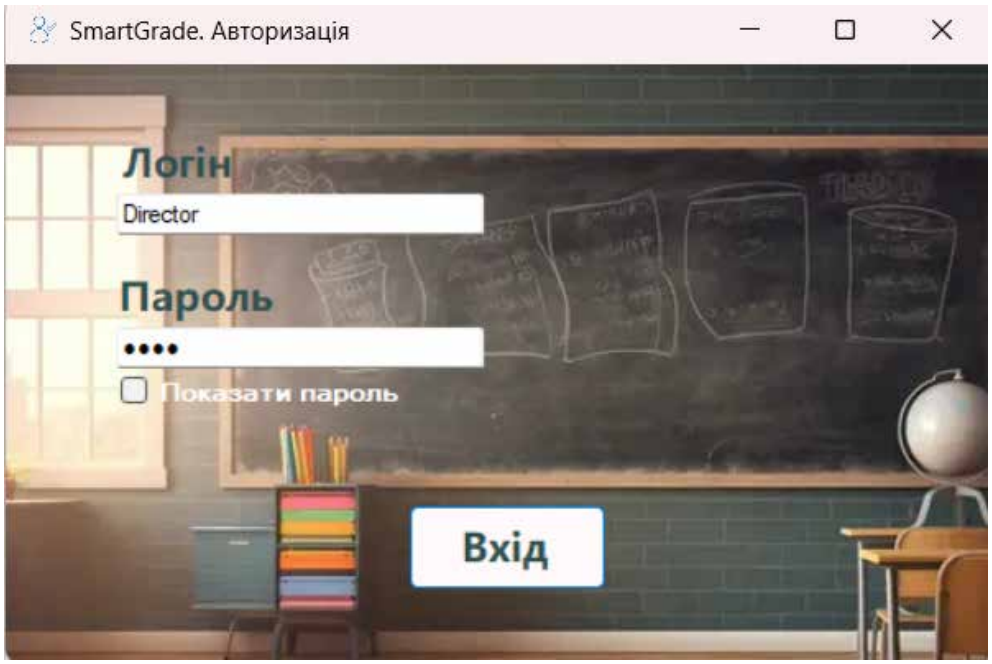


Рис. 4.2 Сторінка входу до системи для ролі «Директор»

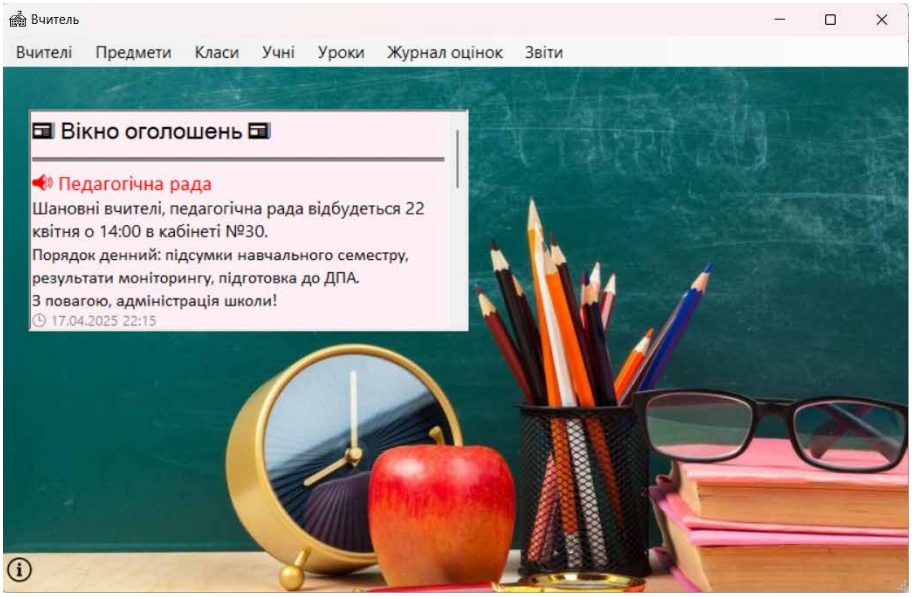


Рис. 4.3 Інтерфейс головного меню для ролі «Вчитель»

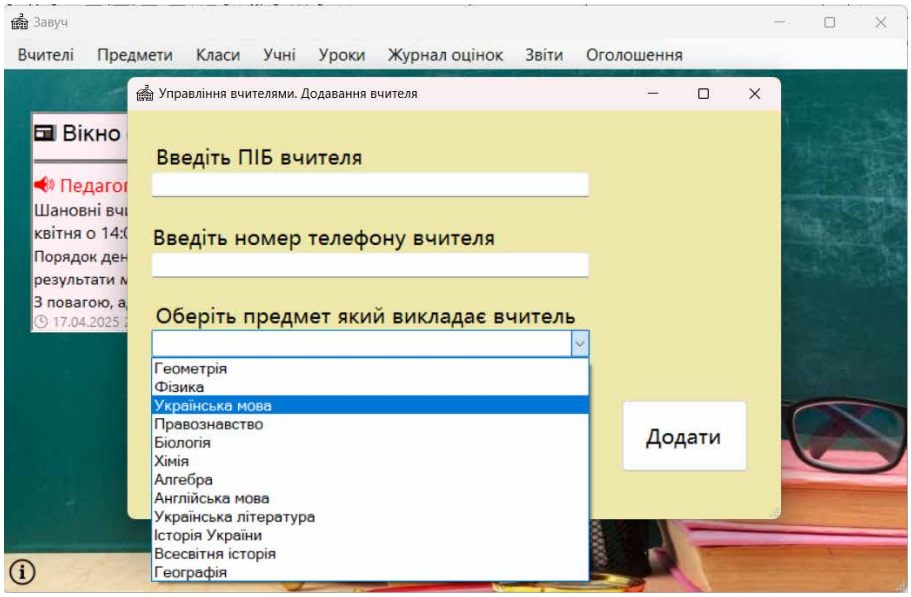


Рис. 4.4 Вікно додавання нового вчителя до системи

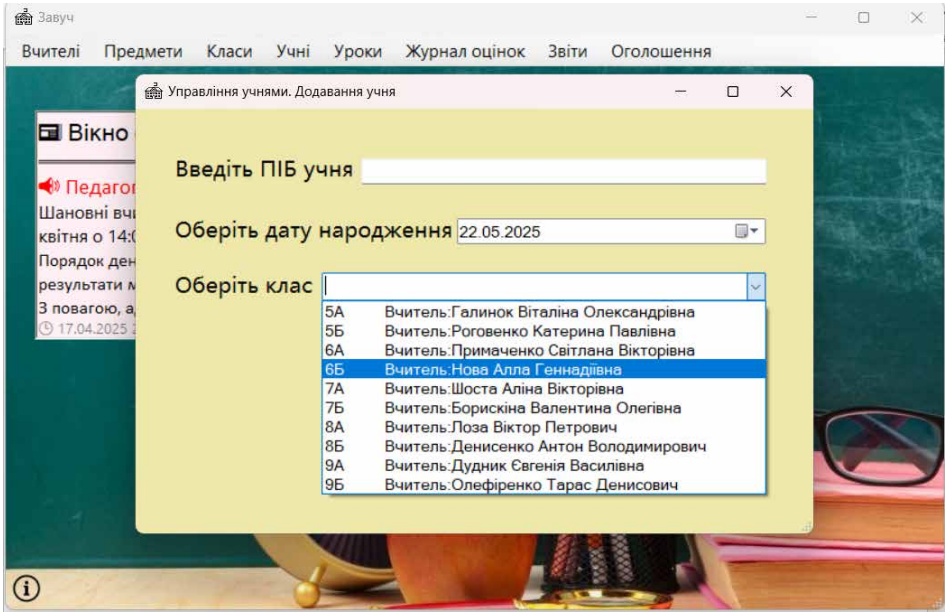


Рис. 4.5 Вікно додавання учня в систему

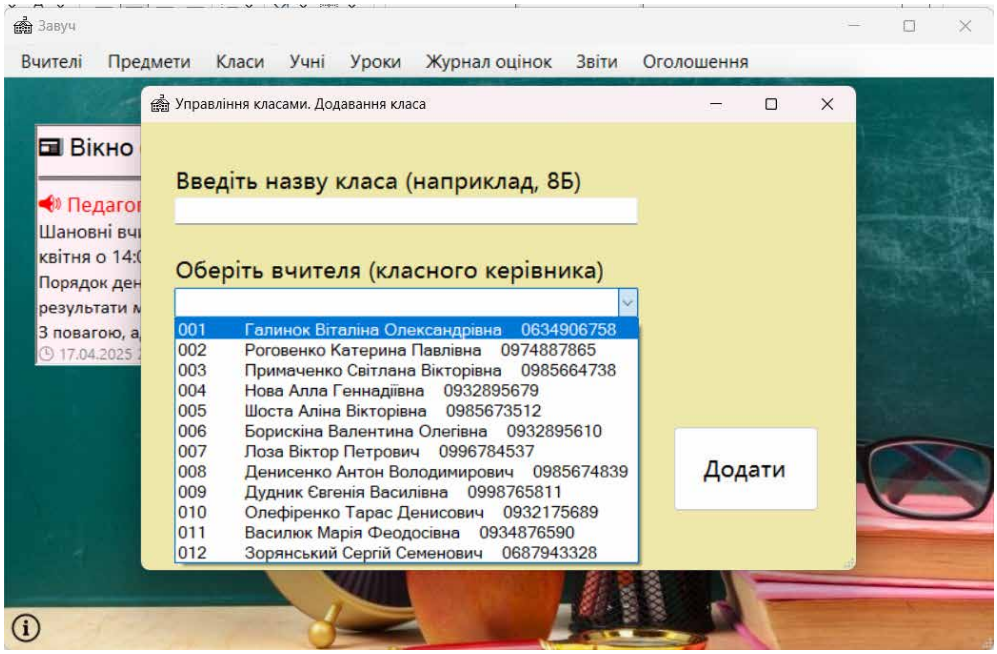


Рис. 4.6 Вікно створення класа

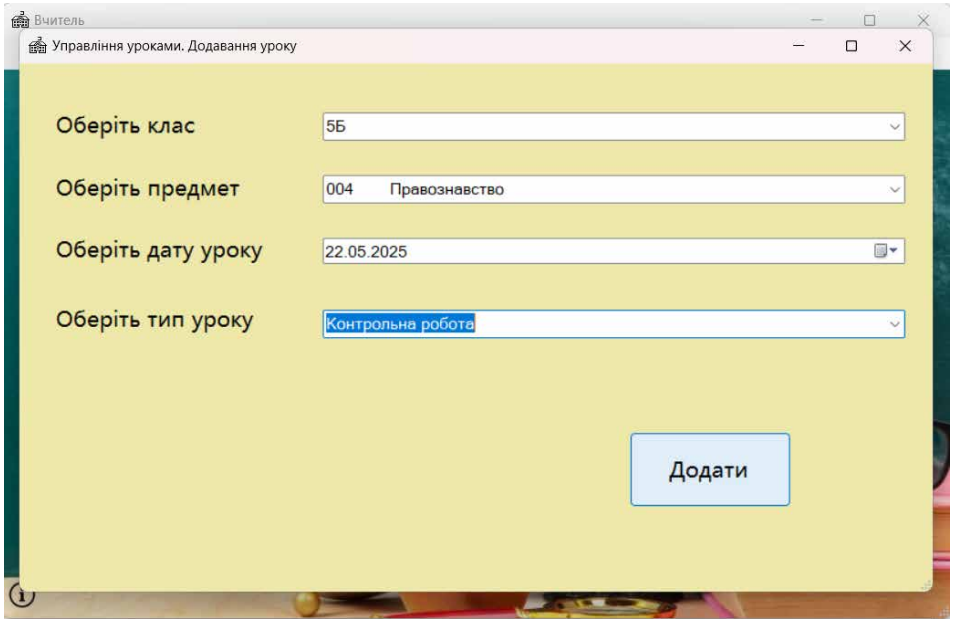


Рис. 4.7 Вікно додавання уроку

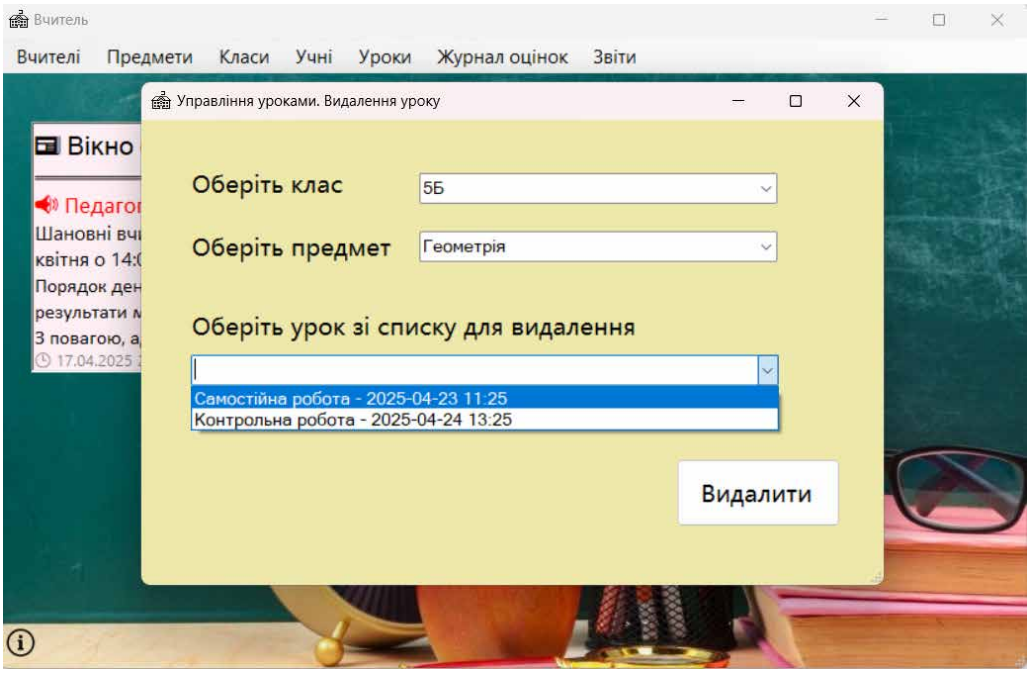


Рис. 4.8 Вікно видалення уроку

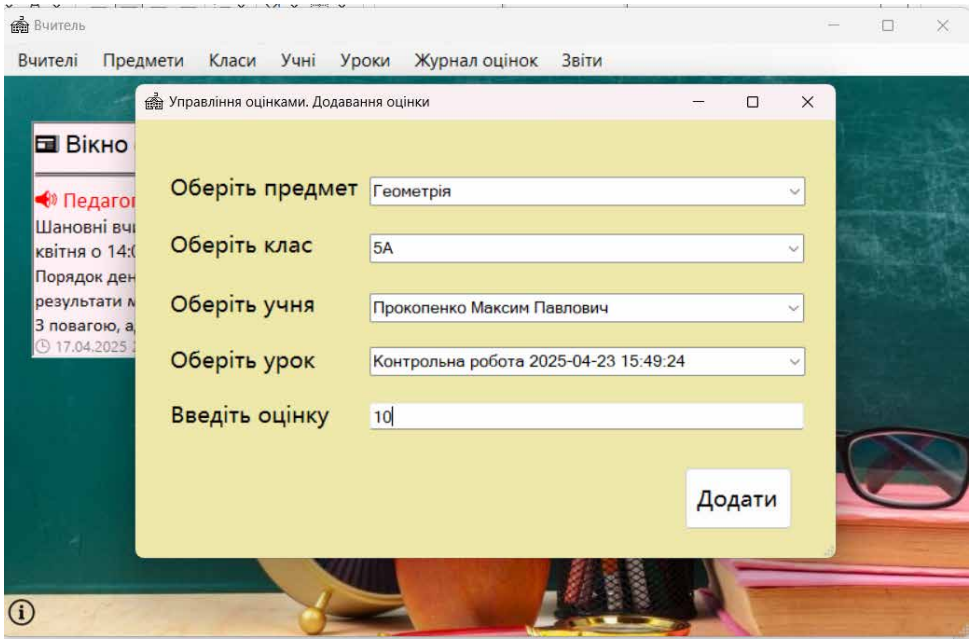


Рис. 4.9 Вікно додавання оцінки

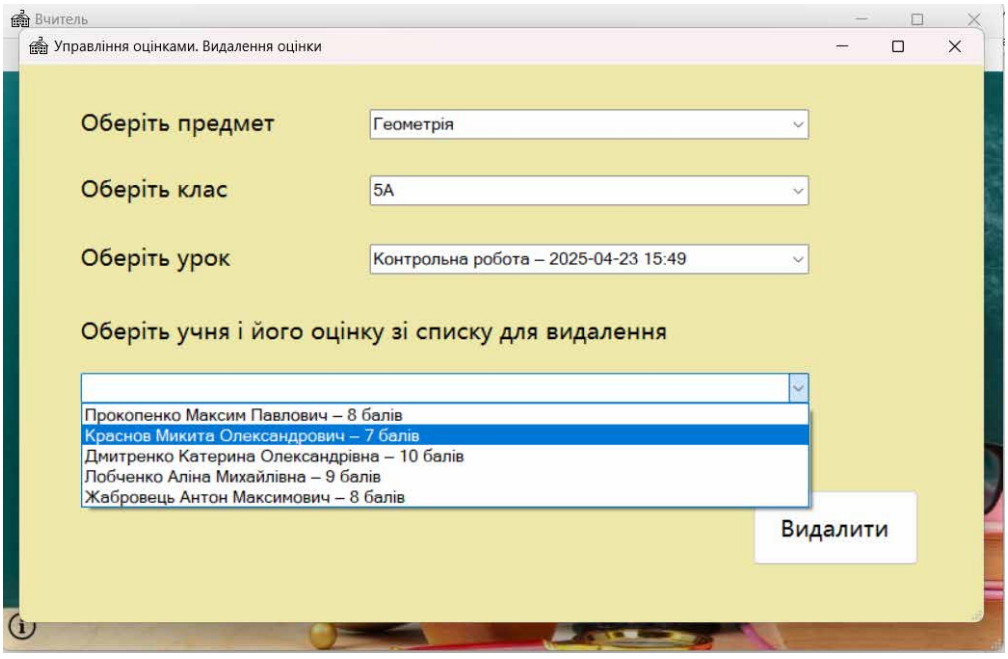


Рис. 4.10 Вікно видалення оцінки

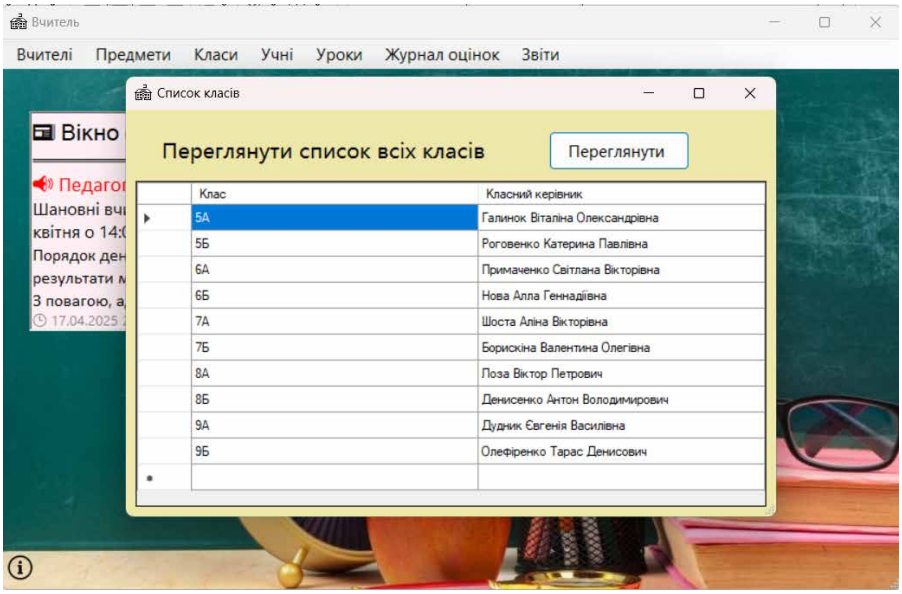


Рис. 4.11 Вікно перегляду списку класів школи

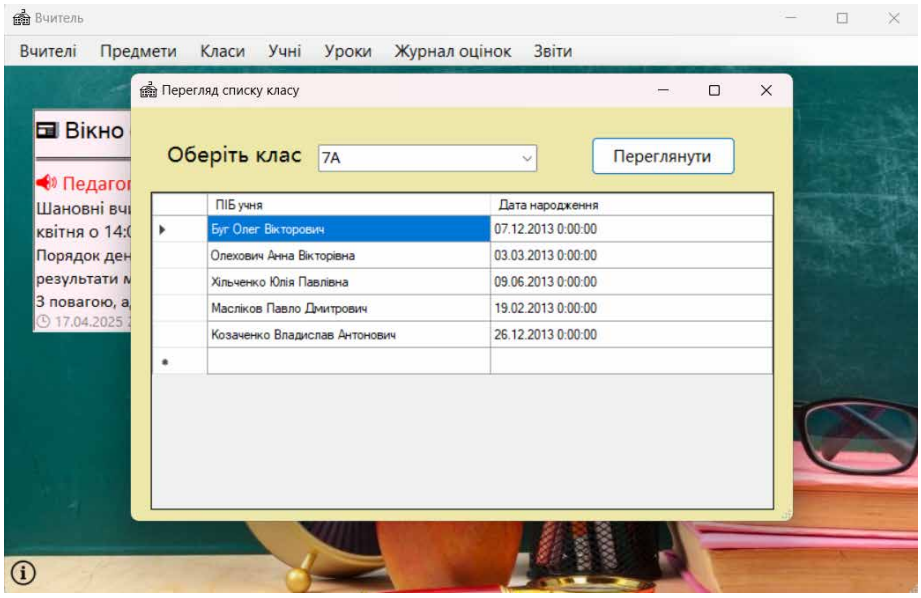


Рис. 4.12 Вікно перегляду списку учнів класу

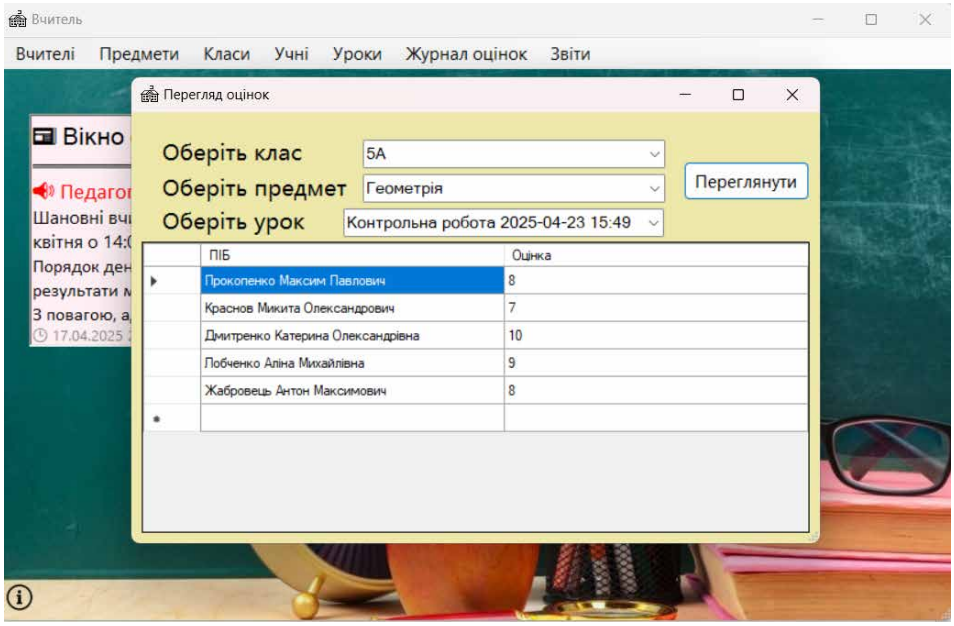


Рис. 4.13 Вікно перегляду списку учнів класу

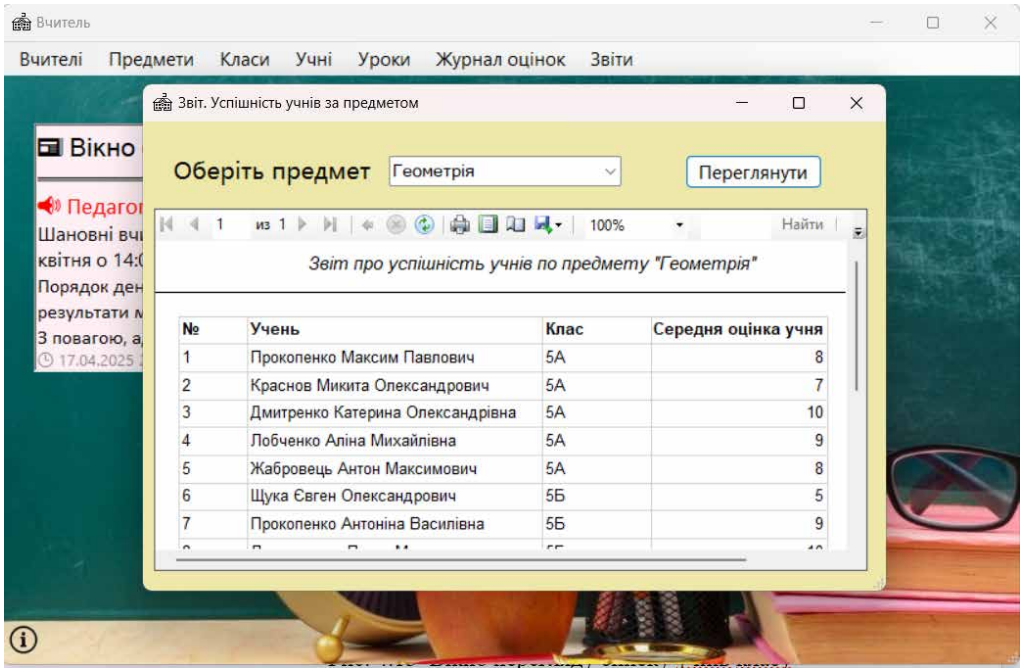


Рис. 4.14 Сформований звіт «Успішність учнів по предмету «Геометрія»»

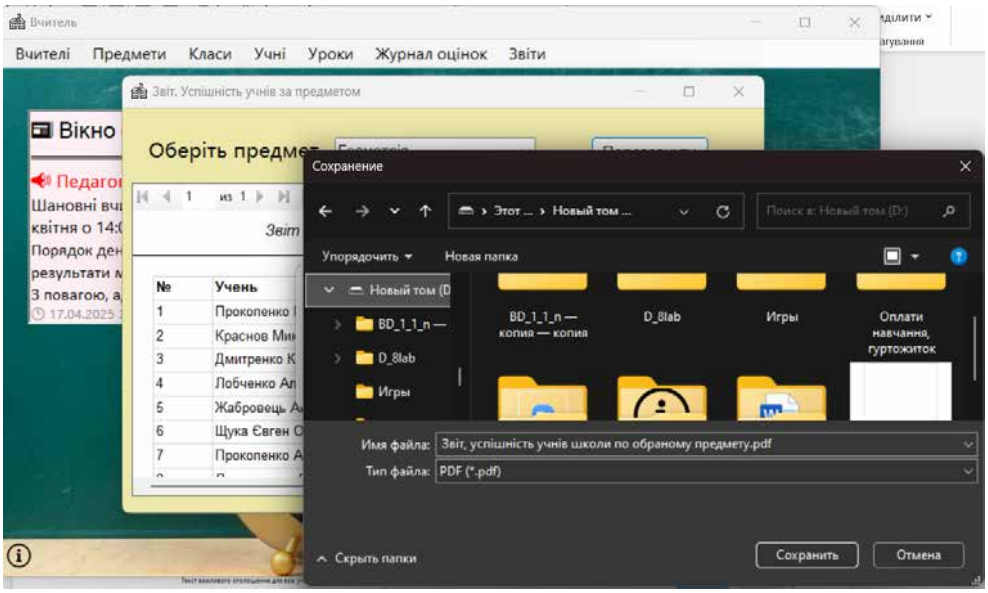


Рис. 4.15 Вікно збереження звіту

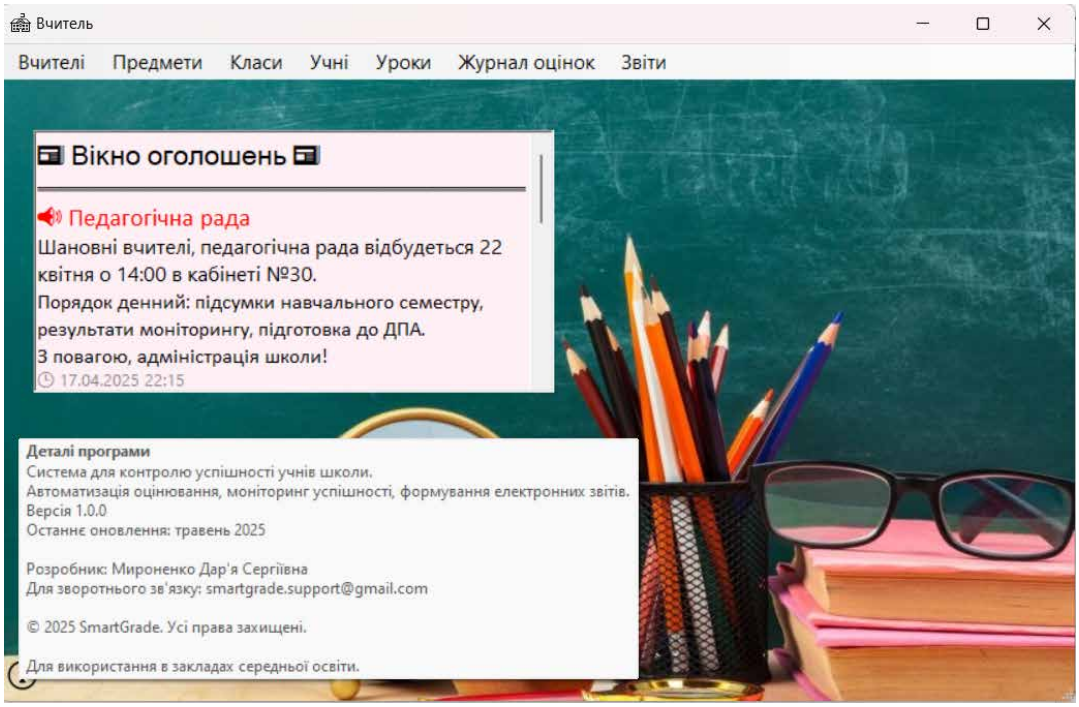


Рис. 4.16 Вікно з спливаючою підказкою з короткою інформацією про програму та технічну підтримку.

Висновки до розділу 4

У межах четвертого розділу було розглянуто ключові аспекти практичного запуску, тестування та експлуатації системи контролю успішності учнів. Проведене функціональне тестування дало змогу перевірити стабільність роботи програмного забезпечення, його відповідність технічному завданню, а також коректність реалізації механізмів розмежування доступу відповідно до ролей користувачів.

Усі основні компоненти системи — від авторизації до роботи з базою даних — були перевірені в межах типових сценаріїв використання, що дозволило своєчасно виявити та усунути незначні помилки. Інтерфейс системи успішно адаптується до ролі користувача (директор, вчитель, завуч), відкриваючи лише дозволений функціонал, що свідчить про правильну реалізацію політики доступу.

Окрему увагу приділено технічним аспектам: наведено апаратні та програмні вимоги як до серверної, так і до клієнтської частини системи. Запропонована конфігурація забезпечує надійність роботи, сумісність із сучасними операційними системами та дає змогу масштабувати рішення відповідно до зростаючих потреб навчального закладу.

Також було описано порядок взаємодії користувача із системою — від завантаження та авторизації до навігації в інтерфейсі та виконання основних операцій. Ілюстративні приклади демонструють доступність функціоналу та інтуїтивну логіку побудови вікон і форм.

У підсумку, проведене тестування, опис вимог до середовища та сценарії використання підтверджують готовність розробленого програмного забезпечення до повноцінного впровадження у навчальній установі. Система забезпечує ефективне управління освітнім процесом, сприяє підвищенню прозорості оцінювання та зручності взаємодії між усіма учасниками навчального середовища.

ВИСНОВКИ

У процесі виконання дипломного проєкту було реалізовано прикладне програмне забезпечення, що виконує функції автоматизованої системи по контролю успішності учнів. Розробка охопила повний цикл створення інформаційної системи — від аналізу предметної області й постановки задач до проєктування архітектури, розробки бази даних, створення інтерфейсу та тестування програмного забезпечення.

Система орієнтована на автоматизацію основних навчально-адміністративних процесів у школі: ведення обліку учнів, викладачів, класів, оцінок і предметів, а також формування звітності. Значну увагу було приділено реалізації ролевої моделі доступу — залежно від ролі (директор, вчитель, завуч), користувач отримує доступ до строго визначеного функціоналу, що забезпечує надійність і структурованість роботи з інформацією.

Для реалізації графічного інтерфейсу користувача було використано мову програмування C# у середовищі Microsoft Visual Studio з використанням технології Windows Forms, що дозволило швидко створити зручний, інтуїтивно зрозумілий інтерфейс. Як платформу для зберігання даних обрано Microsoft SQL Server, що гарантує стабільність, продуктивність і масштабованість бази даних.

У процесі реалізації було створено модулі авторизації, управління класами і учнями, реєстрації оцінок і формування звітів. Всі компоненти були протестовані як з функціональної, так і з безпекової точки зору, що дозволило переконатися у відповідності системи технічному завданню та готовності до впровадження.

Загалом, створена інформаційна система здатна підвищити ефективність управління навчальним процесом, зменшити адміністративне навантаження на педагогів, забезпечити прозорість оцінювання та зручну взаємодію між усіма учасниками освітнього процесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бардус І. О., Лазарєв М. І., Ніценко А. О. Бази даних у схемах (на основі фундаменталізованого підходу): навч. посібник. — Харків: Діса плюс, 2017. — 133 с.
2. Авраменко В. С., Авраменко А. С. Проєктування інформаційних систем: навчальний посібник. — Черкаси: ЧНУ ім. Б. Хмельницького, 2019. – 433.
3. UML: – [Електронний ресурс] – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/
4. Системи управління базами даних: визначення та призначення [Електронний ресурс] – Режим доступу: https://foxminded.ua/systema-upravlinnia-bazamy-danykh/
5. ASP.NET Core Documentation [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/aspnet/core
6. SQL Server Documentation [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/sql/
7. Visual Studio Documentation [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/visualstudio/
8. ERwin Data Modeler [Електронний ресурс] // *Офіційний сайт Erwin*. – Режим доступу: https://www.erwin.com/erwin-data-modeler/
9. Документація JetBrains Rider [Електронний ресурс] – Режим доступу: https://www.jetbrains.com/rider/documentation/

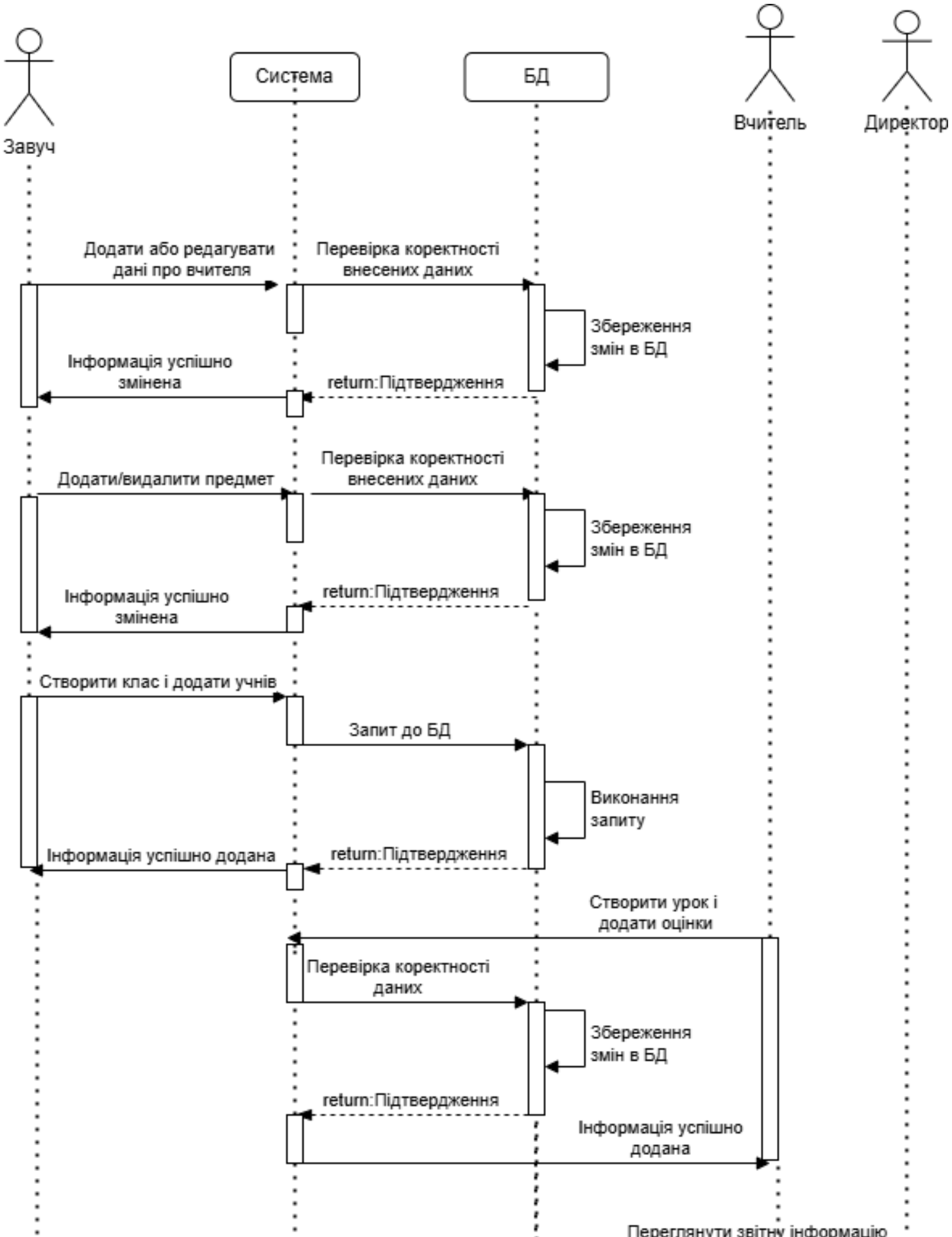
ДОДАТОК А

Діаграми

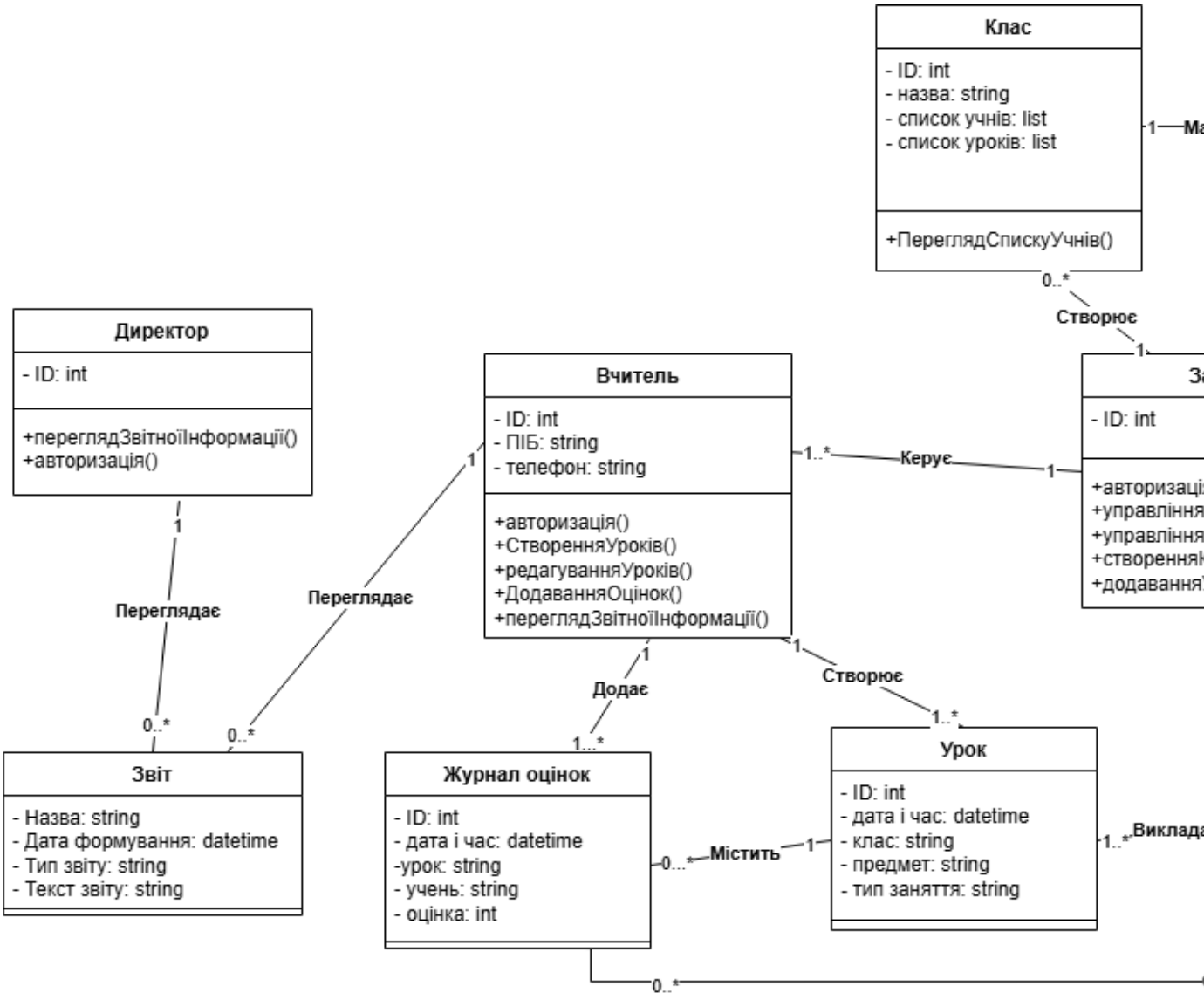
Сторінок 4

Київ - 2025

Діаграма послідовності



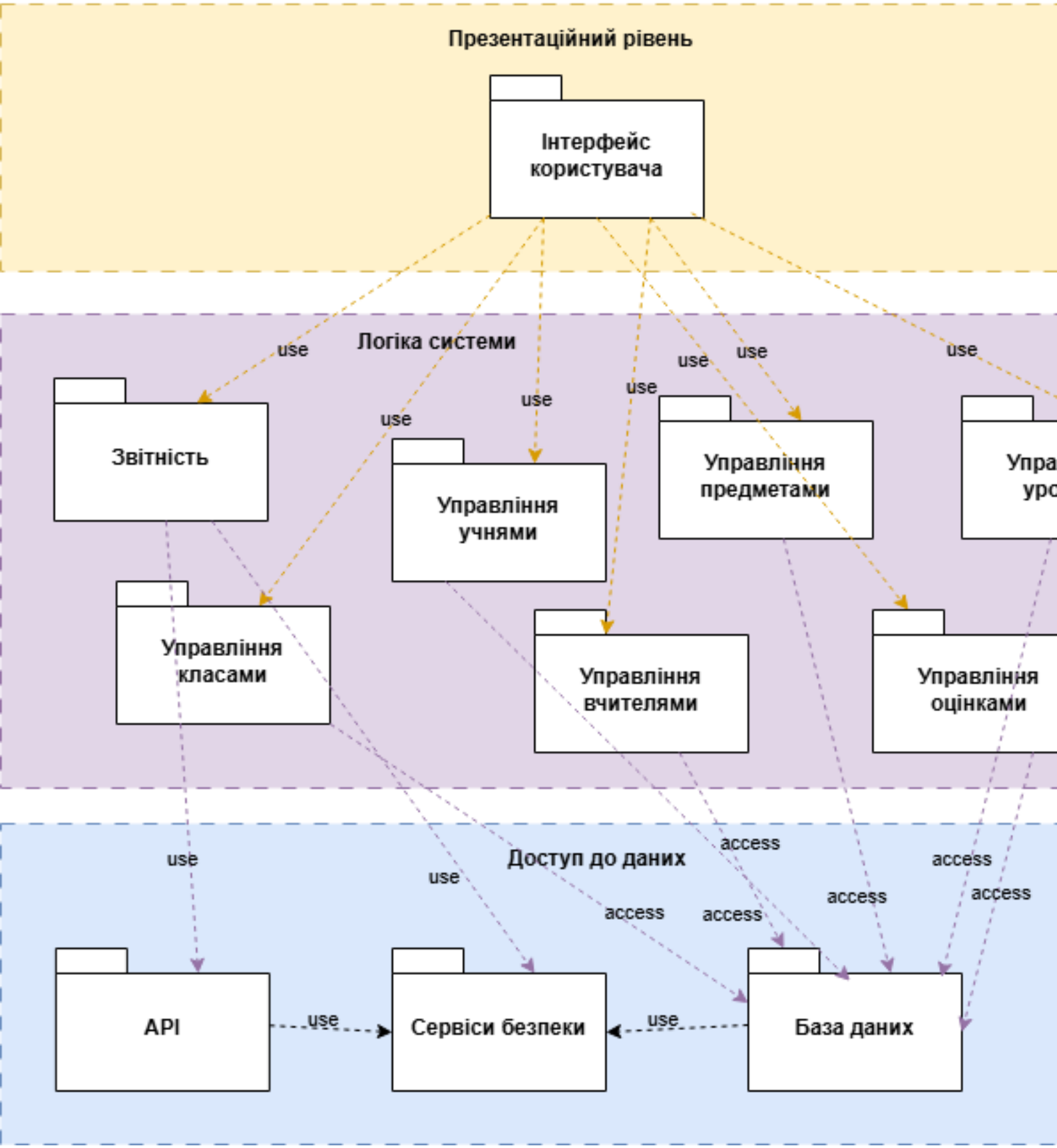
Діаграма класів


Клас
- ID: int
- назва: string
- список учнів: list
- список уроків: list
+ПереглядСпискуУчнів()
1
0..*
Створює
1
Директор
- ID: int
+переглядЗвітноїІнформації()
+авторизація()
1
Переглядає
0..*
Переглядає
1
Вчитель
- ID: int
- ПІБ: string
- телефон: string
+авторизація()
+СтворенняУроків()
+редагуванняУроків()
+ДодаванняОцінок()
+переглядЗвітноїІнформації()
1..*
Керує
1
1
Додає
1...*
1
Створює
1..*
Звіт
- Назва: string
- Дата формування: datetime
- Тип звіту: string
- Текст звіту: string
Журнал оцінок
- ID: int
- дата і час: datetime
-урок: string
- учень: string
- оцінка: int
0...*
Містить
1
Урок
- ID: int
- дата і час: datetime
- клас: string
- предмет: string
- тип заняття: string
1..*
0..*

Сторінка 2

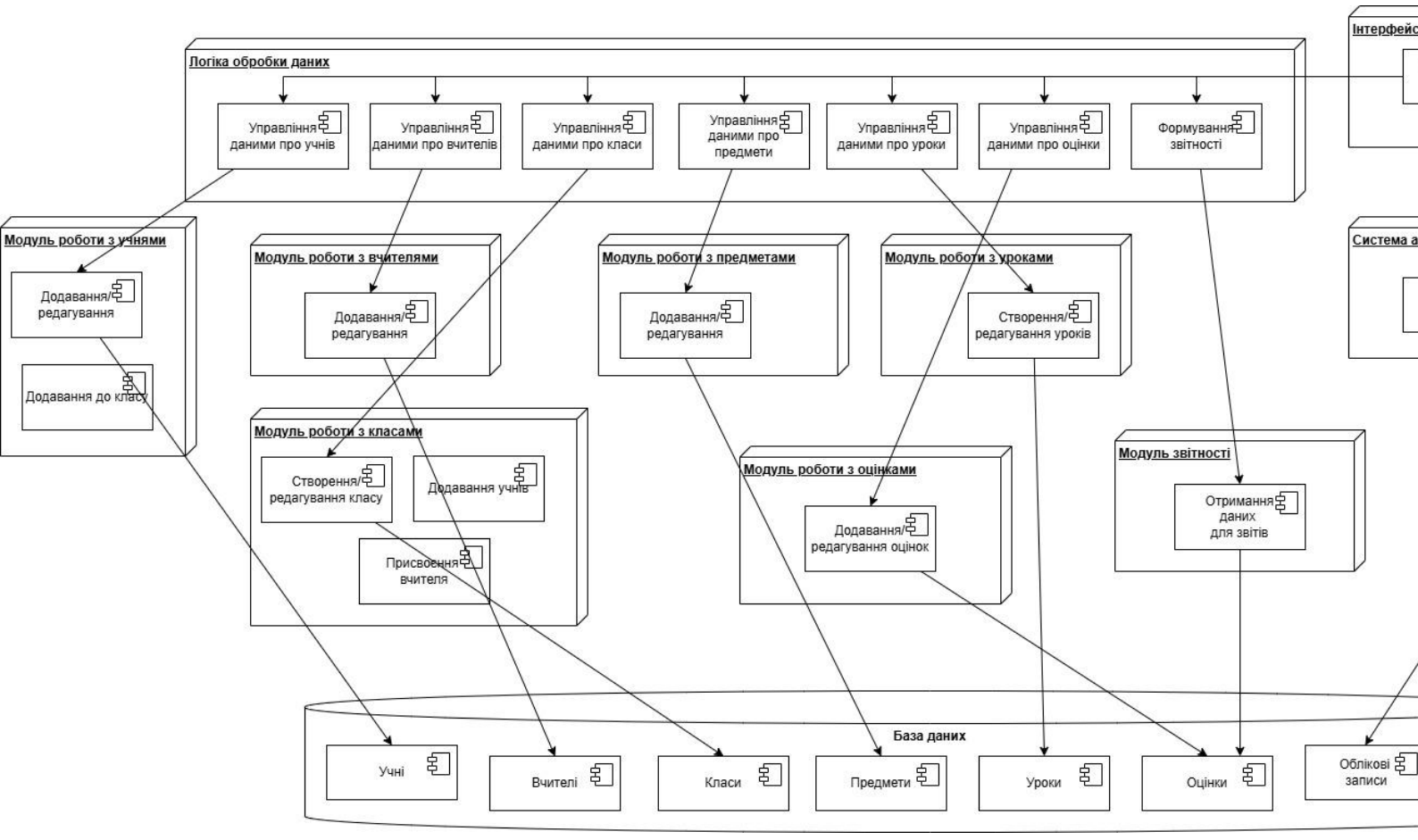
Діаграма пакетів

Діаграма пакетів для ПЗ по контролю успішності учнів школи



Сторінка 3

Діаграма компонентів



ДОДАТОК Б

База даних

Сторінок 9

Київ – 2025

Лістинг коду створення таблиць бази даних

```
use SmartGrade
Go
IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'Score_book' AND type_desc = 'USER_TABLE')
drop table Score_book
IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'Attendance' AND type_desc = 'USER_TABLE')
drop table Attendance
IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'Lesson' AND type_desc = 'USER_TABLE')
drop table Lesson
IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'Student' AND type_desc = 'USER_TABLE')
drop table Student
IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'Class' AND type_desc = 'USER_TABLE')
drop table Class
IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'Teacher' AND type_desc = 'USER_TABLE')
drop table Teacher
IF EXISTS (SELECT name FROM sys.objects
WHERE name = 'Subjects' AND type_desc = 'USER_TABLE')
drop table Subjects
Go
create table Teacher
(
ID_Teacher CHAR(10) NOT NULL PRIMARY KEY,
Name_Teacher VARCHAR(256) NOT NULL,
Telephone Char(10) NULL UNIQUE
);
create table Subjects
(
```

```
ID_Subject CHAR(10) NOT NULL PRIMARY KEY,
Name_Subject VARCHAR(256) NOT NULL,);
create table Class
(
ID_Class CHAR(10) NOT NULL PRIMARY KEY,
Number_of_students INT NOT NULL,
ID_Teacher_FK CHAR(10) not null,
FOREIGN KEY (ID_Teacher_FK) REFERENCES Teacher(ID_Teacher)
);
create table Student
(
ID_Student CHAR(10) NOT NULL PRIMARY KEY,
Name_Student VARCHAR(256) NOT NULL,
Date_birth CHAR(10) not null,
ID_Class_FK CHAR(10) not null,
FOREIGN KEY (ID_Class_FK) REFERENCES Class(ID_Class)
);
create table Lesson
( ID_Lesson CHAR(10) not null primary key,
Type_of_lesson VARCHAR(256) NOT NULL,
Date_lesson DATETIME NOT NULL,
ID_Subject_FK CHAR(10) not null,
ID_Class_FK CHAR(10) not null,
FOREIGN KEY (ID_Subject_FK) REFERENCES Subjects(ID_Subject),
FOREIGN KEY (ID_CLass_FK) REFERENCES Class(ID_Class)
);
create table Score_book
( Score INT NOT NULL,
 ID_Subject_FK CHAR(10) not null,
 ID_Class_FK CHAR(10) not null,
 ID_Student_FK CHAR(10) not null,
 ID_Lesson_FK CHAR(10) not null,
 FOREIGN KEY (ID_Subject_FK) REFERENCES Subjects(ID_Subject),
 FOREIGN KEY (ID_Class_FK) REFERENCES Class(ID_Class),
 FOREIGN KEY (ID_Student_FK) REFERENCES Student(ID_Student),
```

```
FOREIGN KEY (ID_Lesson_FK) REFERENCES Lesson(ID_Lesson));
```


Лістинг коду створення ролей і надання прав в базі даних

```
USE SmartGrade;
GO
-- Створення логінів
IF NOT EXISTS (SELECT name FROM sys.server_principals WHERE name = 'Director')
BEGIN
 CREATE LOGIN [Director] WITH PASSWORD = 'd123';
END
ELSE
BEGIN
 PRINT 'Логін [Director] уже існує.';
END
IF NOT EXISTS (SELECT name FROM sys.server_principals WHERE name =
'TeacherUsername')
BEGIN
 CREATE LOGIN [TeacherUsername] WITH PASSWORD = 'teach123';
END
ELSE
BEGIN
 PRINT 'Логін [TeacherUsername] уже існує.';
END
-- Створення користувачів
IF NOT EXISTS (SELECT name FROM sys.database_principals WHERE name = 'Director' AND
type
= 'U')
BEGIN
 CREATE USER [DirectorUsername] FOR LOGIN [Director];
END
ELSE
BEGIN
 PRINT 'Користувач [DirectorUsername] уже існує.';
END
IF NOT EXISTS (SELECT name FROM sys.database_principals WHERE name =
'TeacherUsername'
AND type = 'U')
BEGIN
 CREATE USER [TeacherUsername] FOR LOGIN [TeacherUsername];
```

```
END
ELSE
BEGIN
 PRINT 'Користувач [TeacherUsername] уже існує.';
END
-- Присвоєння ролей
IF EXISTS (SELECT name FROM sys.database_principals WHERE name = 'Director' AND type
=
'R')
BEGIN
 ALTER ROLE [Director] ADD MEMBER [DirectorUsername];
END
ELSE
BEGIN
 PRINT 'Роль [Director] не існує.';
END
IF EXISTS (SELECT name FROM sys.database_principals WHERE name = 'Teacher' AND type
=
'R')
BEGIN
 ALTER ROLE [Teacher] ADD MEMBER [TeacherUsername];
END
ELSE
BEGIN
 PRINT 'Роль [Teacher] не існує.';
END
GO

-- Створення логіну
IF NOT EXISTS (SELECT name FROM sys.server_principals WHERE name = 'HeadTeacher')
BEGIN
   CREATE LOGIN [HeadTeacher] WITH PASSWORD = 'head123';
END
ELSE
BEGIN
   PRINT 'Логін [HeadTeacher] уже існує.';
END
GO
```

```
-- Створення користувача (перевірка, чи не є це вже роль)
IF NOT EXISTS (
   SELECT name
   FROM sys.database_principals
   WHERE name = 'HeadTeacher' AND type = 'U'
)
BEGIN
   CREATE USER [HeadTeacher] FOR LOGIN [HeadTeacher];
END
ELSE
BEGIN
   PRINT 'Користувач [HeadTeacher] уже існує.';
END
GO

-- Створення ролі з іншим ім’ям (щоб уникнути конфлікту)
IF NOT EXISTS (
   SELECT name
   FROM sys.database_principals
   WHERE name = 'HT_Role' AND type = 'R'
)
BEGIN
   CREATE ROLE [HT_Role];
END
ELSE
BEGIN
   PRINT 'Роль [HT_Role] уже існує.';
END
GO

-- Додавання користувача до ролі
IF NOT EXISTS (
   SELECT rm.role_principal_id
   FROM sys.database_role_members rm
   JOIN sys.database_principals r ON rm.role_principal_id = r.principal_id
   JOIN sys.database_principals u ON rm.member_principal_id = u.principal_id
   WHERE r.name = 'HT_Role' AND u.name = 'HeadTeacher'
)
```

```
BEGIN
   ALTER ROLE [HT_Role] ADD MEMBER [HeadTeacher];                    Сторінка 5
END
ELSE
BEGIN
   PRINT 'Користувач [HeadTeacher] уже є учасником ролі [HT_Role].';
END
GO

-- Надання прав ролі
GRANT SELECT ON dbo.Users TO [HT_Role];
GRANT SELECT ON dbo.Score_book TO [HT_Role];
GRANT SELECT ON dbo.Lesson TO [HT_Role];

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Student TO [HT_Role];
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Teacher TO [HT_Role];
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Class TO [HT_Role];
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Subjects TO [HT_Role];
GO

-- Лістинг коду по створенню ролей
USE SmartGrade;
GO
IF NOT EXISTS (SELECT name FROM sys.database_principals WHERE name = 'Director' AND
type
= 'R')
BEGIN
 CREATE ROLE [Director];
 GRANT SELECT ON dbo.Users TO [Director];
 GRANT SELECT ON dbo.Score_book TO [Director];
 GRANT SELECT ON dbo.Lesson TO [Director];
GRANT SELECT ON dbo.Student TO [Director];
GRANT SELECT ON dbo.Teacher TO [Director];
GRANT SELECT ON dbo.Class TO [Director];
GRANT SELECT ON dbo.Subjects TO [Director];

END
ELSE
BEGIN
```

```
 -- Якщо роль вже існує, оновлюємо її дозволи
CREATE ROLE [Director];                                    Сторінка 6
 GRANT SELECT ON dbo.Users TO [Director];
 GRANT SELECT ON dbo.Score_book TO [Director];
 GRANT SELECT ON dbo.Lesson TO [Director];
GRANT SELECT ON dbo.Student TO [Director];
GRANT SELECT ON dbo.Teacher TO [Director];
GRANT SELECT ON dbo.Class TO [Director];
GRANT SELECT ON dbo.Subjects TO [Director];
END
IF NOT EXISTS (SELECT name FROM sys.database_principals WHERE name = 'Teacher' AND
type =
'R')
BEGIN
 CREATE ROLE [Teacher];
GRANT SELECT ON dbo.Users TO [Teacher];
 GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Score_book TO [Teacher];
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Lesson TO [Teacher];
GRANT SELECT ON dbo.Student TO [Teacher];
GRANT SELECT ON dbo.Teacher TO [Teacher];
 GRANT SELECT ON dbo.Class TO [Teacher];
GRANT SELECT ON dbo.Subjects TO [Teacher];

END
ELSE
BEGIN
 -- Якщо роль вже існує, оновлюємо її дозволи
GRANT SELECT ON dbo.Users TO [Teacher];
 GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Score_book TO [Teacher];
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Lesson TO [Teacher];
GRANT SELECT ON dbo.Student TO [Teacher];
GRANT SELECT ON dbo.Teacher TO [Teacher];
 GRANT SELECT ON dbo.Class TO [Teacher];
GRANT SELECT ON dbo.Subjects TO [Teacher];
END
GO
```

Лістинг коду створення запитів на основі яких формуються звіти

Уявлення 1. Class_Averages

```
CREATE VIEW [dbo].[Class_Averages] AS
SELECT
   Class.ID_Class,
   Teacher.Name_Teacher,
   AVG(Score_book.Score) AS Average_Score
FROM Class
JOIN Teacher ON Class.ID_Teacher_FK = Teacher.ID_Teacher
JOIN Score_book ON Score_book.ID_Class_FK = Class.ID_Class
GROUP BY Class.ID_Class, Teacher.Name_Teacher;
```

Уявлення 2. Student_Ranking

```
CREATE   VIEW Student_Ranking AS
SELECT
S.ID_Student,
S.Name_Student,
C.ID_Class,
L.ID_Subject_FK AS ID_Subject,
MAX(Sub.Name_Subject) AS Name_Subject,
AVG(CAST(SB.Score AS FLOAT)) AS AvgScore
FROM Student S
JOIN Class C ON S.ID_Class_FK = C.ID_Class
JOIN Score_book SB ON S.ID_Student = SB.ID_Student_FK
JOIN Lesson L ON SB.ID_Lesson_FK = L.ID_Lesson
JOIN Subjects Sub ON L.ID_Subject_FK = Sub.ID_Subject
GROUP BY
 S.ID_Student,
S.Name_Student,
 C.ID_Class,
 L.ID_Subject_FK;
```

Уявлення 3. Students_Average

```
CREATE   VIEW Students_Average AS
SELECT
S.ID_Student,
```

```
S.Name_Student,
C.ID_Class,                                                    Сторінка 8
AVG(SB.Score) AS AverageScore
FROM
Student S
JOIN
Score_book SB ON S.ID_Student = SB.ID_Student_FK
JOIN
Class C ON S.ID_Class_FK = C.ID_Class
GROUP BY
S.ID_Student, S.Name_Student, C.ID_Class;
```

ДОДАТОК В

Код програми

Сторінок 22

Київ – 2025

Лістинг коду програми

Лістинг коду функції «Створення класу»

```
namespace BD
{
   public partial class AddClass : Form
   {
      SqlConnection connection;
      public AddClass(SqlConnection connection)
      {
         InitializeComponent();
         this.connection = connection;
         FillComboBox();
      }
      private void FillComboBox()
      {
         try
         {
            string query = "SELECT * FROM Teacher";
            SqlCommand command = new SqlCommand(query, connection);
            SqlDataReader reader = command.ExecuteReader(); // виконуємо запит та отримуємо результат у вигляді об'єкту SqlDataReader
            while (reader.Read())
            {
               string ID = reader.GetString(0);
               string Name = reader.GetString(1);
               string Email = reader.GetString(2
               string comboBoxItem = $"{ID} {Name}    {Email}"; // формуємо рядок, який буде доданий до ComboBox
               comboBox1.Items.Add(comboBoxItem); // додаємо рядок до ComboBox

            }
            reader.Close(); // закриваємо об'єкт SqlDataReader
         }
         catch (Exception ex)
         {
            MessageBox.Show(ex.Message);
}
```

```
        }
        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                //      Перевірка: чи введено ID класу
                if (string.IsNullOrWhiteSpace(textBox1.Text))
                {
                    MessageBox.Show("Будь ласка, введіть назву класу!", "Попередження",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                    return;
                }
                //      Перевірка: чи вибрано вчителя
                if (comboBox1.SelectedIndex == -1)
                {
                    MessageBox.Show("Будь ласка, виберіть вчителя!", "Попередження",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                    return;
                }
                string selectedString = comboBox1.SelectedItem.ToString();
                string[] substring = selectedString.Split(' ');
                string code = substring[0].Trim();
                string id = textBox1.Text;
                string query = "INSERT INTO Class (ID_Class,ID_Teacher_FK) VALUES (@id,
@code)";
                SqlCommand command = new SqlCommand(query, connection);

                command.Parameters.AddWithValue("@id", id);
                command.Parameters.AddWithValue("@code", code);

                int rowsAffected = command.ExecuteNonQuery(); // Повертає кількість вставлених
рядків
                if (rowsAffected > 0)
                {
                    MessageBox.Show("Клас успішно додано!", "Успіх", MessageBoxButtons.OK,
MessageBoxIcon.Information);
                    }
```

```
                else
                {
                    MessageBox.Show("Помилка при додаванні класа!", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
                textBox1.Text = "";
                comboBox1.SelectedIndex = -1;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
```

Лістинг коду функції «Додавання нового учня в систему»

```
namespace BD
{
   public partial class AddStudent : Form
   {
      SqlConnection connection;
      public AddStudent(SqlConnection connection)
      {
         InitializeComponent();
         this.connection = connection;
         FillComboBox();
      }
      private void button1_Click(object sender, EventArgs e)
      {
         try
         {
            //     Перевірка полів
            if (string.IsNullOrWhiteSpace(textBox2.Text))
            {
               MessageBox.Show("Будь ласка, введіть ім’я учня!", "Попередження",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
               return;
            }
            if (comboBox1.SelectedIndex == -1)
                  {
```

```
            MessageBox.Show("Будь ласка, виберіть клас!", "Попередження",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }
        //     Генерація ID для учня
        string getMaxIDQuery = "SELECT MAX(CAST(ID_Student AS INT)) FROM Student";
        SqlCommand getMaxIDCommand = new SqlCommand(getMaxIDQuery, connection);
        var result = getMaxIDCommand.ExecuteScalar();
        int nextID = (result != DBNull.Value) ? Convert.ToInt32(result) + 1 : 1;
        string studentID = nextID.ToString("D3"); // форматування 001, 002...
        //     Отримання даних з форми
        string name = textBox2.Text;
        DateTime birth = dateTimePicker1.Value;
        string selectedClass = comboBox1.SelectedItem.ToString();
        string classID = selectedClass.Split(' ')[0]; // ID класу

        SqlCommand command = new SqlCommand(
            "INSERT INTO Student (ID_Student, Name_Student, Date_birth, ID_Class_FK) VALUES
(@id, @name, @birth, @classID)",
            connection
        );

        command.Parameters.AddWithValue("@id", studentID);
        command.Parameters.AddWithValue("@name", name);
        command.Parameters.AddWithValue("@birth", birth);
        command.Parameters.AddWithValue("@classID", classID);

        int rowsAffected = command.ExecuteNonQuery();

        if (rowsAffected > 0)
        {
            MessageBox.Show("Учня успішно додано!", "Успіх", MessageBoxButtons.OK,
MessageBoxIcon.Information);}
        else
        {
            MessageBox.Show("Помилка при додаванні учня!", "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
```

```
            }

            // Очищення полів
            textBox2.Text = "";
            comboBox1.SelectedIndex = -1;
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }

    }
    private void FillComboBox()
    {
        try
        {
            string query = "SELECT Class.ID_Class, Teacher.ID_Teacher, Teacher.Name_Teacher FROM
Class " +
              "JOIN Teacher ON Class.ID_Teacher_FK = Teacher.ID_Teacher";

            SqlCommand command = new SqlCommand(query, connection);
            SqlDataReader reader = command.ExecuteReader(); // виконуємо запит та отримуємо
результат у вигляді об'єкту SqlDataReader
            while (reader.Read())
            {
                string ID = reader.GetString(0); // отримуємо значення стовпця Company_ID з об'єкту
SqlDataReader
                //string Name = reader.GetString(1);
                string NameTeacher = reader.GetString(2);

                string comboBoxItem = $"{ID} Вчитель:{NameTeacher}"; // формуємо рядок, який буде
доданий до ComboBox
                comboBox1.Items.Add(comboBoxItem); // додаємо рядок до ComboBox

            }
            reader.Close(); // закриваємо об'єкт SqlDataReader
        }
        catch (Exception ex)
        {
```

```
            MessageBox.Show(ex.Message);
        }
    }
```

Лістинг коду функції «Створення уроку»

```
namespace BD
{
    public partial class AddLesson : Form
    {
        SqlConnection connection;
        public AddLesson(SqlConnection connection)
        {
            InitializeComponent();
            this.connection = connection;
            FillComboBox();

            comboBox3.Items.Add("Контрольна робота");
            comboBox3.Items.Add("Самостійна робота");
            comboBox3.Items.Add("Звичайне заняття");
            comboBox3.SelectedIndex = -1; // Немає вибору за замовчуванням
        }
        private void FillComboBox()
        {
            try
            {
                string query1 = "SELECT * FROM Subjects";

                SqlCommand command = new SqlCommand(query1, connection);
                SqlDataReader reader = command.ExecuteReader(); // виконуємо запит та отримуємо
результат у вигляді об'єкту SqlDataReader
                while (reader.Read())
                {
                    string ID = reader.GetString(0); // Додай цей рядок
                    string Name = reader.GetString(1);

                    string comboBoxItem = $"{ID} {Name}"; // формуємо рядок, який буде доданий до
ComboBox
                    comboBox1.Items.Add(comboBoxItem); // додаємо рядок до ComboBox
```

```
            }
            reader.Close(); // закриваємо об'єкт SqlDataReader

            string query2 = "SELECT * FROM Class";

            SqlCommand command1 = new SqlCommand(query2, connection);
            SqlDataReader reader1 = command1.ExecuteReader(); // виконуємо запит та отримуємо результат у вигляді об'єкту SqlDataReader
            while (reader1.Read())
            {
                string ID = reader1.GetString(0); // отримуємо значення стовпця Company_ID з об'єкту SqlDataReader
                string comboBoxItem = $"{ID}"; // формуємо рядок, який буде доданий до ComboBox
                comboBox2.Items.Add(comboBoxItem); // додаємо рядок до ComboBox
            }
            reader1.Close(); // закриваємо об'єкт SqlDataReader
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private void button1_Click(object sender, EventArgs e)
    {
        try
        {
            //     Перевірка заповнення всіх полів
            if (comboBox3.SelectedIndex == -1)
            {
                MessageBox.Show("Будь ласка, виберіть тип уроку!", "Попередження", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }
            if (comboBox1.SelectedIndex == -1)
            {
                MessageBox.Show("Будь ласка, виберіть предмет!", "Попередження", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
```

```
        }

        if (comboBox2.SelectedIndex == -1)
        {
            MessageBox.Show("Будь ласка, виберіть клас!", "Попередження",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }

        //  Отримання ID предмета
        string selectedSubject = comboBox1.SelectedItem.ToString();
        string subjectID = selectedSubject.Split(' ')[0].Trim();

        //  Отримання ID класу
        string selectedClass = comboBox2.SelectedItem.ToString();
        string classID = selectedClass.Split(' ')[0].Trim();

        //  Автоматичне формування ID уроку
        string getMaxIDQuery = "SELECT MAX(CAST(ID_Lesson AS INT)) FROM Lesson";
        SqlCommand getMaxIDCommand = new SqlCommand(getMaxIDQuery, connection);
        var result = getMaxIDCommand.ExecuteScalar();
        int nextID = (result != DBNull.Value) ? Convert.ToInt32(result) + 1 : 1;
        string lessonID = nextID.ToString("D3");

        string lessonType = comboBox3.SelectedItem.ToString(); // Отримуємо тип уроку з
ComboBox
        DateTime lessonDate = dateTimePicker1.Value;

        //  Запит на додавання уроку
        SqlCommand command = new SqlCommand("INSERT INTO Lesson (ID_Lesson,
Type_of_lesson, Date_lesson, ID_Subject_FK, ID_Class_FK) VALUES (@id, @type, @date, @subject,
@class)", connection);
        command.Parameters.AddWithValue("@id", lessonID);
        command.Parameters.AddWithValue("@type", lessonType);
        command.Parameters.AddWithValue("@date", lessonDate);
        command.Parameters.AddWithValue("@subject", subjectID);
        command.Parameters.AddWithValue("@class", classID);
```



```
        int rowsAffected = command.ExecuteNonQuery();
```

```
            if (rowsAffected > 0)
            {
                MessageBox.Show("Урок успішно додано!", "Успіх", MessageBoxButtons.OK,
MessageBoxIcon.Information);
            }
            else
            {
                MessageBox.Show("Помилка при додаванні уроку!", "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
            //      Очищення полів після додавання
            comboBox3.SelectedIndex = -1;
            comboBox1.SelectedIndex = -1;
            comboBox2.SelectedIndex = -1;
        }
        catch (Exception ex)
        {
            MessageBox.Show("Сталася помилка: " + ex.Message, "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }
    }
```

Лістинг коду функції «Виставлення оцінки»

```
namespace BD
{
   public partial class AddScore : Form
   {
      SqlConnection connection;

      private string selectedClassId = null;
      private string selectedSubjectName = null;
      private Dictionary<string, string> lessonDisplayToIdMap = new Dictionary<string, string>();

      public AddScore(SqlConnection connection)
      {                                                                    Сторінка 9
         InitializeComponent();
         this.connection = connection;
```

```
    FillComboBox();

    comboBox1.SelectedIndexChanged += comboBox1_SelectedIndexChanged;
}

private class StudentItem
{
    public string ID { get; set; }
    public string Name { get; set; }

    public StudentItem(string id, string name)
    {
        ID = id;
        Name = name;
    }

    public override string ToString()
    {
        return Name; // Виводимо лише ім’я
    }
}
private void FillComboBox()
{
    try
    {
        string query1 = "SELECT * FROM Subjects";
        SqlCommand command1 = new SqlCommand(query1, connection);
        SqlDataReader reader1 = command1.ExecuteReader();
        while (reader1.Read())
        {
            string Name = reader1.GetString(1);
            comboBox1.Items.Add($"{Name}");
        }
        reader1.Close();

        string query2 = "SELECT * FROM Class";
        SqlCommand command2 = new SqlCommand(query2, connection);
        SqlDataReader reader2 = command2.ExecuteReader();
        while (reader2.Read())
```

```
            {
                string ID = reader2.GetString(0);
                comboBox2.Items.Add($"{ID}");
            }
            reader2.Close();

            string query3 = "SELECT * FROM Student";
            SqlCommand command3 = new SqlCommand(query3, connection);
            SqlDataReader reader3 = command3.ExecuteReader();
            while (reader3.Read())
            {
                string ID = reader3.GetString(0);
                string Name = reader3.GetString(1);
                string IDClass = reader3.GetString(3);
                comboBox3.Items.Add($"{ID} {Name} {IDClass}");
            }
            reader3.Close();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private void button1_Click(object sender, EventArgs e)
    {
        try
        {
            if (comboBox1.SelectedIndex == -1)
            {
                MessageBox.Show("Будь ласка, виберіть предмет!", "Попередження",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }

            if (comboBox2.SelectedIndex == -1)
            {                                                                    Сторінка 11
                MessageBox.Show("Будь ласка, виберіть клас!", "Попередження",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
```

```
            return;
        }

        if (comboBox3.SelectedIndex == -1)
        {
            MessageBox.Show("Будь ласка, виберіть учня!", "Попередження",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }

        if (comboBox4.SelectedIndex == -1)
        {
            MessageBox.Show("Будь ласка, виберіть урок!", "Попередження",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }

        if (string.IsNullOrWhiteSpace(textBox1.Text))
        {
            MessageBox.Show("Будь ласка, введіть оцінку!", "Попередження",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }

        string selectedString1 = comboBox1.SelectedItem.ToString();
        string subjectName = selectedString1.Trim();

        string selectedString2 = comboBox2.SelectedItem.ToString();
        string classId = selectedString2.Trim();

        StudentItem selectedStudent = (StudentItem)comboBox3.SelectedItem;
        string studentId = selectedStudent.ID;

        string selectedLessonText = comboBox4.SelectedItem.ToString();

        // Перевірка, чи є вибір уроку в словнику
        if (!lessonDisplayToIdMap.ContainsKey(selectedLessonText))
        {
```

```
            MessageBox.Show("Урок не знайдений", "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return;
        }

        string lessonId = lessonDisplayToIdMap[selectedLessonText];

        string scoreText = textBox1.Text;
        int score;

        if (!int.TryParse(scoreText, out score) || score < 1 || score > 12)
        {
            MessageBox.Show("Будь ласка, введіть коректну оцінку від 1 до 12.", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            textBox1.Focus();
            return;
        }

        // Отримуємо ID_Subject по імені предмета
        string subjectIdQuery = "SELECT ID_Subject FROM Subjects WHERE Name_Subject =
@subname";
        SqlCommand subjectCmd = new SqlCommand(subjectIdQuery, connection);
        subjectCmd.Parameters.AddWithValue("@subname", subjectName);
        string subjectId = subjectCmd.ExecuteScalar()?.ToString();

        // Перевірка на null для subjectId
        if (string.IsNullOrEmpty(subjectId))
        {
            MessageBox.Show("Не знайдено ID предмета", "Помилка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return;
        }

        SqlCommand command = new SqlCommand(
            "INSERT INTO Score_book (Score, ID_Subject_FK, ID_Class_FK, ID_Student_FK,
ID_Lesson_FK) " +
            "VALUES (@score, @idsub, @idclass, @idstud, @idless)", connection);
```



```
        command.Parameters.AddWithValue("@score", score);
```

```
            command.Parameters.AddWithValue("@idsub", subjectId);
            command.Parameters.AddWithValue("@idclass", classId);
            command.Parameters.AddWithValue("@idstud", studentId);
            command.Parameters.AddWithValue("@idless", lessonId);

            int rowsAffected = command.ExecuteNonQuery();

            if (rowsAffected > 0)
            {
                MessageBox.Show("Оцінка успішно додана!", "Успіх", MessageBoxButtons.OK,
MessageBoxIcon.Information);
            }
            else
            {
                MessageBox.Show("Помилка при додаванні оцінки!", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
    private void comboBox2_SelectedIndexChanged(object sender, EventArgs e)
    {
        selectedClassId = comboBox2.SelectedItem.ToString().Split(' ')[0].Trim();
        UpdateStudentComboBox();
        UpdateLessonsComboBox();
    }
    private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
        selectedSubjectName = comboBox1.SelectedItem.ToString().Trim();
        UpdateLessonsComboBox();
    }
    private void UpdateStudentComboBox()
    {
        comboBox3.Items.Clear();
```

```
        string query = "SELECT ID_Student, Name_Student FROM Student WHERE ID_Class_FK =
@idclass";
        SqlCommand command = new SqlCommand(query, connection);
        command.Parameters.AddWithValue("@idclass", selectedClassId);

        SqlDataReader reader = command.ExecuteReader();
        while (reader.Read())
        {
            string ID = reader.GetString(0);
            string Name = reader.GetString(1);
            comboBox3.Items.Add(new StudentItem(ID, Name));
        }
        reader.Close();
    }

    private void UpdateLessonsComboBox()
    {
        comboBox4.Items.Clear();
        lessonDisplayToIdMap.Clear();

        if (string.IsNullOrEmpty(selectedClassId) || string.IsNullOrEmpty(selectedSubjectName))
            return;

        string query = @"SELECT Lesson.ID_Lesson, Lesson.Type_of_lesson, Lesson.Date_lesson,
                    Subjects.Name_Subject
                    FROM Lesson
                    JOIN Subjects ON Lesson.ID_Subject_FK = Subjects.ID_Subject
                    WHERE Lesson.ID_Class_FK = @idclass AND Subjects.Name_Subject = @subname";

        SqlCommand command = new SqlCommand(query, connection);
        command.Parameters.AddWithValue("@idclass", selectedClassId);
        command.Parameters.AddWithValue("@subname", selectedSubjectName);

        SqlDataReader reader = command.ExecuteReader();
        while (reader.Read())
        {
            string ID = reader.GetString(0);
            string Typeoflesson = reader.GetString(1);
            DateTime Datelesson = reader.GetDateTime(2);
```

```
            string displayText = $"{Typeoflesson} {Datelesson:yyyy-MM-dd HH:mm:ss}";
            comboBox4.Items.Add(displayText);
            lessonDisplayToIdMap[displayText] = ID;
        }
        reader.Close();
    }
```

Лістинг коду функції «Перегляд звіту 1. Середній бал класів»

```
namespace BD
{
    public partial class Report1 : Form
    {
        SqlConnection connection;

        public Report1(SqlConnection connection)
        {
            InitializeComponent();
            this.connection = connection;
        }

        private void Report1_Load(object sender, EventArgs e)
        {
            // При завантаженні форми — нічого не заповнюємо
            reportViewer1.Reset(); // очищення повністю
        }

        private void reportViewer1_Load(object sender, EventArgs e)
        {
            // Також очищаємо на всякий випадок
            reportViewer1.Reset();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                string query = "SELECT * FROM Class_Averages";
                SqlDataAdapter adapter = new SqlDataAdapter(query, connection);
```

```
            DataSet ds = new DataSet();
            adapter.Fill(ds);

            // Якщо немає даних — повідомлення
            if (ds.Tables[0].Rows.Count == 0)
            {
                MessageBox.Show("Немає даних для формування звіту.", "Інформація",
MessageBoxButtons.OK, MessageBoxIcon.Information);

                // Очистити ReportViewer
                reportViewer1.Clear();
                return;
            }

            Class_AveragesBindingSource.DataSource = ds.Tables[0];

            reportViewer1.LocalReport.DataSources.Clear();
            reportViewer1.LocalReport.DataSources.Add(
                new ReportDataSource("DataSet1", Class_AveragesBindingSource));

            reportViewer1.LocalReport.ReportEmbeddedResource = "BD.Report1.rdlc"; // Шлях до звіту

            reportViewer1.RefreshReport();
        }
        catch (Exception ex)
        {
            MessageBox.Show("Помилка при завантаженні звіту: " + ex.Message, "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error); }}}}
```

Лістинг коду функції «Перегляд звіту 2. Рейтинг учнів за окремими предметами»

```
namespace BD
{
   public partial class Report2 : Form
   {
      SqlConnection connection;

      public Report2(SqlConnection connection)
```

```
        {
            InitializeComponent();
            this.connection = connection;}

        // При завантаженні форми завантажуємо список предметів у ComboBox
        private void Report2_Load(object sender, EventArgs e)
        {
            this.dataSet2.EnforceConstraints = false;
            // TODO: данная строка кода позволяет загрузить данные в таблицу
"dataSet2.Student_Ranking". При необходимости она может быть перемещена или удалена.
            this.student_RankingTableAdapter1.Fill(this.dataSet2.Student_Ranking);
            // Завантажуємо предмети в ComboBox
            LoadSubjects();
        }
        // Завантаження предметів в ComboBox
        private void LoadSubjects()
        {
            string query = "SELECT ID_Subject, Name_Subject FROM Subjects";
            SqlDataAdapter adapter = new SqlDataAdapter(query, connection);
            DataTable subjects = new DataTable();
            adapter.Fill(subjects);

            subjectComboBox.Items.Clear();
            subjectComboBox.DisplayMember = "Name_Subject";  // що показувати
            subjectComboBox.ValueMember = "ID_Subject";      // справжнє значення
            subjectComboBox.DataSource = subjects;

            subjectComboBox.SelectedIndex = -1; // прибирає автоматичний вибір
        }

        // Кнопка для перегляду звіту по обраному предмету
        private void viewButton_Click(object sender, EventArgs e)
        {
            if (subjectComboBox.SelectedIndex == -1)
            {
                MessageBox.Show("Будь ласка, виберіть предмет для перегляду звіту.", "Увага",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }
```

```
            int selectedSubjectId = Convert.ToInt32(subjectComboBox.SelectedValue);

            string query = @"
        SELECT ID_Student, Name_Student, ID_Class, AvgScore
        FROM Student_Ranking
        WHERE ID_Subject = @SubjectId";

            SqlDataAdapter adapter = new SqlDataAdapter(query, connection);
            adapter.SelectCommand.Parameters.AddWithValue("@SubjectId", selectedSubjectId);

            DataSet ds = new DataSet();
            adapter.Fill(ds);

            // якщо дані відсутні
            if (ds.Tables[0].Rows.Count == 0)
            {
                MessageBox.Show("Немає даних для вибраного предмету.", "Порожній звіт",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                reportViewer1.Clear(); // очистити попередній звіт
                return;
            }

            studentRankingBindingSource.DataSource = ds.Tables[0];

            reportViewer1.LocalReport.ReportEmbeddedResource = "BD.Report2.rdlc"; // Шлях до звіту
            reportViewer1.LocalReport.DataSources.Clear();
            reportViewer1.LocalReport.DataSources.Add(new ReportDataSource("DataSet2",
studentRankingBindingSource));
            // Додаємо параметр
            string selectedSubjectName = subjectComboBox.Text;
            ReportParameter subjectParam = new ReportParameter("SubjectName", selectedSubjectName);
            reportViewer1.LocalReport.SetParameters(subjectParam);
            reportViewer1.RefreshReport();
        }}
```

Лістинг коду функції «Перегляд звіту 3. Список учнів з середньою оцінкою нижче обраної»

```
namespace BD
{
   public partial class Report3 : Form
   {
      SqlConnection connection;

      public Report3(SqlConnection connection)
      {
         InitializeComponent();
         this.connection = connection;
         reportViewer1.LocalReport.ReportEmbeddedResource = "BD.Report3.rdlc"; // важливо правильно вказати шлях
      }

      private void Report3_Load(object sender, EventArgs e)
      {
         try
         {
            this.myDBDataSet11.EnforceConstraints = false;

            // Заповнюємо комбобокс оцінками
            comboBox1.Items.Clear();
            for (int i = 1; i <= 12; i++)
            {
               comboBox1.Items.Add(i);
            }

            comboBox1.SelectedIndex = -1; // Спочатку нічого не вибрано

            // !!! НЕ викликаємо reportViewer1.RefreshReport() тут !!!

         }
         catch (Exception ex)
         {
```

```
            MessageBox.Show(ex.Message);
        }
    }

    private void button1_Click(object sender, EventArgs e)
    {
        try
        {
            // Перевірка: чи обрана оцінка
            if (comboBox1.SelectedIndex == -1)
            {
                MessageBox.Show("Будь ласка, оберіть оцінку перед формуванням звіту!", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }

            decimal threshold = Convert.ToDecimal(comboBox1.SelectedItem);

            string query = @"SELECT ID_Student, Name_Student, ID_Class, AverageScore
                FROM Students_Average
                WHERE AverageScore <= @threshold";

            SqlDataAdapter adapter = new SqlDataAdapter(query, connection);
            DataSet ds = new DataSet();
            adapter.SelectCommand.Parameters.AddWithValue("@threshold", threshold);
            adapter.Fill(ds);
            adapter.Dispose();

            // Перевірка чи є результати
            if (ds.Tables[0].Rows.Count == 0)
            {
                MessageBox.Show("Немає учнів із середнім балом менше або рівним обраній оцінці.",
"Інформація", MessageBoxButtons.OK, MessageBoxIcon.Information);

                studentsAverageBindingSource.DataSource = null; // Очистити попередній вміст
                reportViewer1.Clear(); // Очистити старий звіт
                return;
            }
```

```
            studentsAverageBindingSource.DataSource = ds.Tables[0];

            reportViewer1.LocalReport.DataSources.Clear();
            reportViewer1.LocalReport.DataSources.Add(new
Microsoft.Reporting.WinForms.ReportDataSource("DataSet3", studentsAverageBindingSource));

            // Передаємо параметр у заголовок
            ReportParameter param = new ReportParameter("ThresholdScore", threshold.ToString());
            reportViewer1.LocalReport.SetParameters(param);

            reportViewer1.RefreshReport();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);}}}
```