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 управління роботи студентськими науковими гуртками»

Спеціальність 121 – «Інженерія програмного забезпечення» ОПП - «Інженерія програмного забезпечення»

Гарант освітньої програми

_____к.т.н., доцент_____ _____ _____Голуб .Б.Л.

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

Керівник бакалаврської кваліфікаційної роботи

_____ _____ _____ _____Панкратьєв В.О

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

Виконала _____Мірошниченко А.В_____

(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 студентці
Мірошниченко Аліні Вадимівні

Спеціальність 121 – «Інженерія програмного забезпечення»

1. Тема бакалаврської кваліфікаційної роботи «Програмне забезпечення автоматизованої системи управління роботи студентськими науковими гуртками»

Затверджена наказом ректора НУБіП України №2248 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 прийняла до виконання_____/А.В. Мірошниченко/
підпис ініціали та прізвище

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

ЗМІСТ

ВСТУП

Актуальність. Освітній процес у закладах вищої освіти є одним із найважливіших напрямів розвитку сучасного суспільства. У зв'язку з цифровою трансформацією освіти, вища освіта зазнає суттєвих змін, зокрема у сфері позанавчальної діяльності студентів. Студентські наукові гуртки відіграють ключову роль у цій діяльності, оскільки сприяють розвитку наукового потенціалу студентів, формуванню дослідницьких навичок та кращій підготовці до майбутньої професійної діяльності.

Для ефективної роботи таких гуртків необхідно належним чином організувати облік, управління інформацією та документацією. З огляду на зростання кількості гуртків і учасників, а також обсягу звітної документації, виникає потреба у впровадженні сучасних інформаційних технологій. Значну частину часу керівників займає оформлення документації та підготовка звітів, що потребує автоматизації.

Впровадження автоматизованої системи управління діяльністю студентських гуртків дозволяє зменшити часові витрати на обробку даних, уникнути помилок при формуванні звітності, покращити контроль та доступ до інформації. Ключову роль у цьому відіграє база даних, що забезпечує надійне зберігання та швидкий доступ до всієї необхідної інформації.

Заклади вищої освіти потребують простого, безкоштовного та зручного у використанні програмного забезпечення для обліку діяльності студентських наукових гуртків і оцінки їхньої результативності. Така система має вирішувати комплекс завдань: спростити роботу з базами учасників, гуртків, керівників і використаних матеріалів; забезпечити ефективне планування заходів через інтегрований календар подій; реалізувати оперативне інформування через систему новин та оголошень. Саме тому було поставлене завдання розробити систему, яка зробить роботу керівників гуртків більш продуктивною завдяки централізованому управлінню всіма аспектами діяльності.

Основною метою даної бакалаврської кваліфікаційної роботи є розробка програмного забезпечення автоматизованої системи управління роботою студентських наукових гуртків. Система повинна забезпечити зручне ведення обліку гуртків, учасників, занять, матеріалів та звітів, а також надати інструменти для ефективного планування та аналізу діяльності.

Для досягнення мети необхідно вирішити такі завдання:

1. Дослідити предметну область та визначити ключові процеси й інформаційні потоки.
2. Проаналізувати наявні програмні рішення у цій сфері.
3. Сформулювати функціональні та нефункціональні вимоги до системи.
4. Побудувати UML-діаграми для візуалізації архітектури та процесів.
5. Обрати інструменти та технології розробки.
6. Розробити інтерфейс користувача з урахуванням зручності та доступності.
7. Спроєктувати та реалізувати структуру бази даних Microsoft SQL.
8. Розробити програмний продукт.
9. Провести тестування розробленої системи.

Кінцевий продукт дозволить зберігати, обробляти, редагувати, переглядати та швидко знаходити потрібну інформацію про наукові гуртки, їхніх учасників та керівників, вести облік матеріалів, планувати події через вбудований календар, публікувати та отримувати новини й оголошення. Все це має максимально спростити роботу користувачів і надати їм потрібну інформацію у зручному форматі в будь-який момент.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка задачі

Система для управління студентськими науковими гуртками призначена для автоматизації процесів організації, обліку та моніторингу гуртkової діяльності у навчальних закладах. Вона сприяє ефективній взаємодії між директором, керівниками та учасниками гуртків, забезпечує зручне ведення документації, планування подій та контроль за виконанням поставлених завдань.

Основні цілі системи:

1. Спростити роботу керівників гуртків та директора.
2. Забезпечити зручний облік учасників, гуртків і керівників.
3. Впорядкувати додавання, видалення та перегляд новин, оголошень і подій.
4. Створити єдину базу даних усіх учасників, гуртків, керівників та матеріалів.
5. Надати директору інструменти для формування звітної інформації.

У процесі роботи студентських наукових гуртків часто виникають труднощі, зумовлені відсутністю централізованої системи обліку, необхідністю вручну складати звітну документацію, а також великою кількістю рутинних однотипних операцій — таких як додавання чи редагування інформації про учасників або гуртки.

Розроблена система хоч і не повністю автоматизує всі адміністративні процеси, проте значно покращує ведення обліку та управління гурткoвою діяльністю, надаючи зручний інтерфейс і набір інструментів для щоденної роботи.

Функціональні вимоги:

1. Облік гуртків, їх керівників і учасників.
2. Створення, редагування та видалення новин і оголошень.
3. Керування календарем подій (додавання, видалення, перегляд).
4. Облік матеріалів, що використовуються в гуртковій діяльності.
5. Генерація та експорт звітів щодо діяльності гуртків у формати docx або pdf.
6. Розмежування прав доступу до функцій системи відповідно до ролей користувачів.

Нефункціональні вимоги:

1. Безпека: Захист даних (авторизація, автентифікація, обмежений доступ).
2. Надійність: Стабільна робота, резервне копіювання та відновлення даних.
3. Масштабованість: Здатність до зростання без втрати продуктивності.
4. Юзабіліті: Інтуїтивно зрозумілий інтерфейс, легка навігація.
5. Легкість обслуговування: Зрозуміла архітектура для легкого внесення змін та розширення.

Уся інформація в системі зберігається у базі даних, що дозволяє отримувати актуальні дані у реальному часі. Завдяки реалізації захисту за допомогою авторизації та розмежування прав доступу, забезпечується безпека даних.

Програма розроблена з урахуванням зручності використання, легкості навчання та максимальної адаптації до потреб кінцевих користувачів.

1.2 Моделювання предметної області

Моделювання предметної області є важливим етапом у процесі проєктування та розробки інформаційних систем. Сучасний ринок інструментальних засобів пропонує широкий спектр технологій для підтримки

цього процесу, від традиційних CASE-інструментів до новітніх систем із застосуванням штучного інтелекту.

Ефективне моделювання предметної області вимагає комплексного розуміння як технічних аспектів, так і відповідних бізнес-процесів. Предметна область є ключовою частиною реальності, що визначає задачу, яку має вирішити система. Моделі предметної області є важливим інструментом для налагодження комунікації та досягнення спільного розуміння системних вимог між усіма зацікавленими сторонами проєкту.

Неправильне розуміння потреб користувачів та нечітке визначення вимог часто призводять до значних проблем у процесі розробки програмного забезпечення. Як наслідок, якісне моделювання предметної області стає критично важливим фактором для успішної реалізації проєкту. Очевидно, що витрати на виправлення помилок, допущених на початкових етапах аналізу вимог, є значно меншими порівняно з їх усуненням на пізніших стадіях.

Модель предметної області в інформаційних системах — це спосіб описати, як працює певна сфера діяльності. Вона показує основні поняття, їхні характеристики та те, як вони пов'язані між собою. Така модель допомагає розробникам краще зрозуміти, що саме потрібно створити — базу даних, логіку роботи системи та зручний інтерфейс для користувача.

Для створення якісних моделей предметної області часто використовують мову UML (Unified Modeling Language) — універсальну графічну нотацію для візуалізації, проєктування та документування компонентів програмних систем. UML дозволяє описувати основні об'єкти предметної області, їхні властивості, дії та взаємозв'язки за допомогою різних типів діаграм, таких як діаграми класів, об'єктів та інші. Це допомагає краще зрозуміти структуру системи ще до початку її розробки.

У контексті розробки автоматизованої системи управління роботою студентських наукових гуртків, модель предметної області відіграє ключову роль у визначенні основних сутностей та їх взаємозв'язків. Аналіз предметної області, проведений на першому етапі розробки (згідно з поставленими завданнями бакалаврської кваліфікаційної роботи), дозволив виділити наступні ключові сутності, які є центральними для функціонування системи:

1. **Гурток:** характеризується назвою, описом, керівником, списком учасників та переліком використовуваних матеріалів.
2. **Керівник:** представляє особу, відповідальну за організацію та діяльність гуртка. Має ідентифікаційні дані та пов'язаний з одним або кількома гуртками.
3. **Учасник:** студент, який є членом одного або кількох наукових гуртків. Має ідентифікаційні дані та пов'язаний з гуртками.
4. **Заняття (подія):** представляє планові заходи гуртка, такі як зустрічі, семінари, майстер-класи. Має назву, дату, час, місце проведення та пов'язане з конкретним гуртком.
5. **Матеріал:** ресурси, обладнання та інструменти, що використовуються в діяльності гуртка. Це фізичні об'єкти (наприклад, пробірки, хімічні реактиви, 3d-принтери, інструменти для дослідів). Кожен матеріал пов'язаний з конкретним гуртком та може мати опис, кількість або інші характеристики.
6. **Новина/оголошення:** інформаційні повідомлення, що публікуються для інформування учасників та керівників про діяльність гуртків.

Для візуалізації статичної структури предметної області розробленої системи була використана діаграма класів UML. Ця діаграма відображає перелічені вище сутності, їхні атрибути та взаємозв'язки (наприклад, зв'язок "один-до-багатьох" між керівником та гуртками, "багато-до-багатьох" між учасниками та гуртками). Діаграми діяльності UML були застосовані для моделювання основних бізнес-процесів, таких як реєстрація учасника в гуртку, додавання нової події до календаря, публікація новини тощо. Діаграми

послідовності використовувалися для відображення взаємодії між користувачами та системою при виконанні ключових функціональних вимог.

Застосування UML дозволило не лише візуалізувати архітектуру системи на ранніх етапах розробки, але й забезпечити єдине розуміння предметної області між розробником та потенційними користувачами, що є важливим аспектом успішного проєктування інформаційної системи управління студентськими науковими гуртками.

При створенні моделі предметної області, важливо дотримуватися кількох ключових підходів. По-перше, варто зосереджуватись лише на головному — на тих особливостях сфери, які справді впливають на роботу системи. Це і є суть абстракції. По-друге, якщо система складна, її краще поділити на менші частини — так простіше зрозуміти, як усе працює. Це називається декомпозицією. І ще один важливий принцип — інкапсуляція: його суть у тому, щоб приховати внутрішню логіку об'єктів і показати зовні лише те, що необхідне для взаємодії з ними.

Моделювання предметної області тісно пов'язане з використанням різних діаграмних технік. UML, як стандартизована мова моделювання, пропонує широкий набір діаграм для відображення різних аспектів системи. Діаграми класів використовуються для моделювання статичної структури предметної області, діаграми діяльності відображають бізнес-процеси, а діаграми послідовності демонструють взаємодію між об'єктами.

Використання формальних методів моделювання предметної області забезпечує чітку та структуровану документацію системи. UML-діаграми не лише візуалізують поточний стан проєкту, але й слугують основою для подальшої розробки, тестування та підтримки програмного забезпечення. Детальна модель полегшує розуміння архітектури системи, спрощує внесення змін та розширення функціоналу в майбутньому.

Підсумовуючи, можна стверджувати, що моделювання предметної області є критичним елементом успішної розробки програмного забезпечення. Це не просто технічний етап, а складний інтелектуальний процес, який вимагає глибокого розуміння бізнес-контексту та ефективної комунікації між усіма зацікавленими сторонами. Інвестиції в якісне моделювання значно знижують ризики проєкту та підвищують ймовірність створення програмного продукту, що відповідає реальним потребам користувачів.

1.3 Діаграма прецендентів

Діаграма прецедентів (або діаграма варіантів використання, англ. *Use Case Diagram*) є одним із найважливіших інструментів мов моделювання UML (Unified Modeling Language), що дозволяє описувати функціональність програмної системи з точки зору кінцевих користувачів. Вона використовується на ранніх етапах проєктування для виявлення та візуалізації вимог до системи, забезпечуючи спільне розуміння її можливостей між розробниками, замовниками та іншими зацікавленими сторонами.

Основні елементи діаграми:

1. Межа системи (System Boundary).

Межа системи відображається прямокутником, у якому розміщуються всі варіанти використання. Все, що знаходиться поза цією межею — не є частиною реалізації програмного забезпечення.

2. Актор (Actor).

Актор — це будь-яка зовнішня по відношенню до системи сутність, яка взаємодіє з нею: користувач, інша система або пристрій. Актори зображуються у вигляді стилізованих фігурок людини. Один актор може взаємодіяти з кількома варіантами використання, і навпаки — один варіант може бути доступний кільком акторам.

3. Варіант використання (Use Case).

Варіант використання — це функціональна одиниця, що описує взаємодію між актором і системою для досягнення певної цілі. Кожен варіант позначається еліпсом і має короткий опис, наприклад, "Авторизація", "Редагування профілю" тощо .

4. Сценарії (Scenarios).

Сценарій є конкретною реалізацією варіанту використання — послідовністю дій, які виконуються під час взаємодії актора з системою. Він може бути основним (успішне завершення задачі) або альтернативним (наприклад, при помилках або виняткових ситуаціях).

5. Зв’язки (Relationships):

- **асоціація (association):** зв’язок між актором і варіантом використання;
- **include:** використовується, коли один варіант використання обов’язково включає поведінку іншого. наприклад, «увійти в систему» обов’язково включає «ввести логін і пароль»;
- **extend:** використовується для опису розширених, необов’язкових або умовних сценаріїв. наприклад, «здійснити оплату» може розширюватися варіантом «отримати бонус» лише в окремих випадках.

Призначення та переваги

Головна мета діаграми варіантів використання — надати спільне бачення функціональності системи. Вона є зручною формою комунікації між усіма учасниками проєкту: розробниками, аналітиками, дизайнерами, замовниками та кінцевими користувачами. Вона дозволяє:

- визначити, що система повинна робити (а не як саме вона це робить);

- зосередити увагу на поведінці з точки зору користувача;
- уникнути надлишкового технічного занурення на ранніх етапах проєкту;
- підготувати основу для подальшого проєктування класів, діаграм активності або послідовностей.

Діаграми прецедентів є ефективним способом формалізації функціональних вимог до системи. Легкість у використанні та наочність роблять їх відмінним інструментом для початкового аналізу вимог, показу архітектури системи та обговорень у команді.За допомогою елементів, таких як актори, варіанти використання та зв’язки, можна точно і зрозуміло описати можливості та обмеження системи.

У системі , яка буде розроблятися, можна виокремити три основні актори:

1. **Директор**: керує інформацією про гуртки та учасників, може додавати, редагувати або видаляти ці дані. Він також займається управлінням інформацією про керівників гуртків, переглядає списки учасників та керівників, додає події до календаря, переглядає новини та оголошення, а також формує звітну інформацію щодо гуртків, їх учасників, керівників та бюджету.
2. **Керівник**: має можливість переглядати списки учасників, гуртків і керівників, додавати події до календаря. Він також займається управлінням новинами та оголошеннями — додає, видаляє або редагує їх. Крім того, керівник веде облік матеріалів, що використовуються в діяльності гуртка, і має можливість додавати чи видаляти ці матеріали.
3. **Учасник**: може переглядати новини та оголошення, події в календарі, а також списки гуртків, учасників і керівників. Учасник має лише доступ до перегляду наявної інформації, без можливості внесення змін.

На рис. 1.1 зображено діаграму прецедентів, яка відображає основні сценарії взаємодії користувачів із системою.

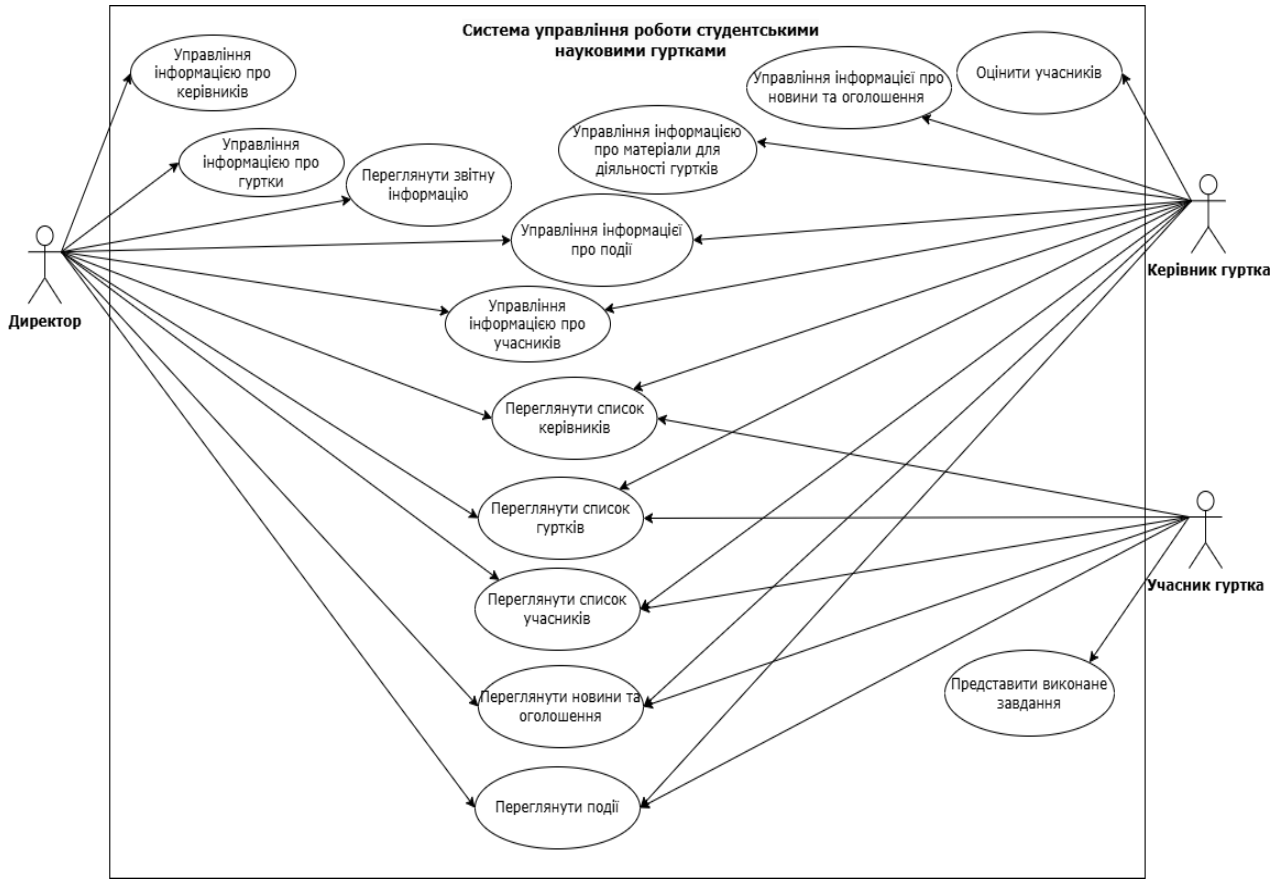


Рис.1.1 Діаграма прецендентів

Для предметної області даної системи виділено наступних акторів, які описані в таблиці 1.1.

Таблиця 1.1

Опис акторів

Актор	Короткий опис
Директор	Особа, яка здійснює загальне управління системою, визначає політику та організує діяльність гуртків і керівників.
Керівник	Особа, що організовує та координує роботу гуртка, взаємодіє з учасниками та забезпечує виконання задач і обов'язків.
Учасник	Особа, яка бере участь у діяльності гуртка, взаємодіє з іншими учасниками та керівником, отримує доступ до інформації.

1.4 Діаграма активності

Діаграма активності є одним з найбільш зрозумілих і візуально зручних інструментів у моделюванні поведінки системи. Вона дає змогу не лише описати логіку бізнес-процесів, а й виявити потенційні помилки чи неефективність ще до початку реалізації програмного продукту. Ці діаграми є частиною формалізованої мови UML (Unified Modeling Language), що забезпечує їхню універсальність та однозначність у розумінні між аналітиками, розробниками та замовниками.

Зовні діаграма активності нагадує блок-схему, однак ключова відмінність полягає в тому, що вона описує не лише послідовність дій, а й дозволяє моделювати паралельне виконання процесів, що особливо важливо для сучасних інформаційних систем. Діаграми активності є зручними для опису складних логік поведінки, особливо коли йдеться про паралелізм або альтернативні шляхи виконання.

Головною складовою діаграми активності є власне активність, тобто логічний стан системи, у якому вона виконує певну дію. Після завершення дії система переходить до іншої активності. Такі переходи можуть бути умовними,

залежними від зовнішньої події або значення параметра. У діаграмі також використовуються символи початку та завершення процесу, логічні розгалуження, об'єднання, що дозволяє повноцінно охопити складні бізнес-сценарії.

Ці особливості роблять діаграми активності ефективним інструментом під час підготовки до автоматизації бізнес-процесів. За допомогою діаграми діяльності (або активності) зручно зображувати процеси у вигляді алгоритмів, за якими функціонує програмне забезпечення автоматизованої системи управління роботою студентських наукових гуртків.

На практиці дуже важливо ще на етапі аналізу чітко визначити, які дії відбуваються у межах кожного процесу, хто є відповідальним за кожну дію, які ресурси потрібні. Розробник повинен мати повне уявлення про логіку роботи підприємства. Недостатнє розуміння процесів може призвести до розробки неефективного або неповного рішення, що в свою чергу призведе до додаткових витрат або навіть провалу проекту.

Таким чином, діаграма активності — це не просто візуальний засіб. Вона є ключовим інструментом для системного аналізу, розуміння та проєктування поведінки програмних рішень. Її застосування забезпечує глибше розуміння бізнес-логіки, дозволяє заздалегідь врахувати можливі помилки, оптимізувати процеси і в результаті — створити більш ефективну систему.

На рис. 1.2 зображено діаграму активності, що демонструє послідовність виконання основних дій користувача в межах системи.

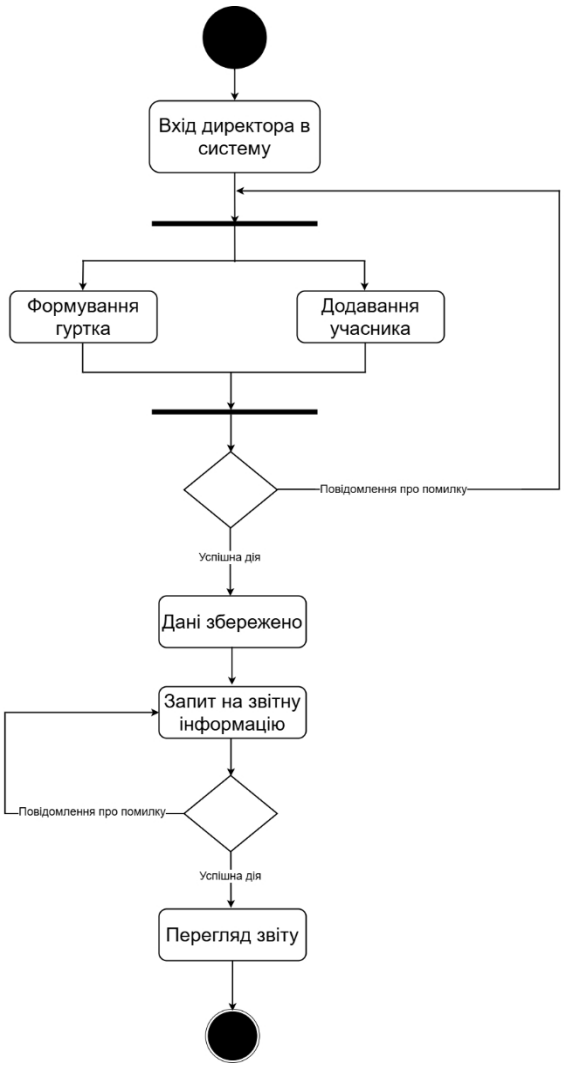


Рис.1.2 Процес управління студентськими науковими гуртками (діаграма активності)

На діаграмі активності зображено алгоритм роботи директора в системі управління студентськими науковими гуртками. Процес починається з входу директора в систему, після чого виконуються паралельні процеси формування гуртка та додавання учасника. Система перевіряє коректність введених даних, і якщо виникає помилка, повертає директора до початку процесу з відповідним повідомленням. При успішному виконанні дані зберігаються в системі. Далі директор може зробити запит на звітну інформацію, яка після успішної перевірки виводиться у вигляді звіту. Процес завершується після перегляду звіту.

1.5 Діаграма послідовності

Діаграма послідовності (Sequence Diagram) — це один із типів діаграм UML, що використовується для відображення послідовності взаємодій між об'єктами програмної системи під час виконання певного сценарію. Вона дозволяє описати динаміку системи, демонструючи, як об'єкти взаємодіють між собою у часовому порядку.

На такій діаграмі зображуються як зовнішні користувачі (актори), так і внутрішні компоненти системи, між якими відбувається обмін повідомленнями для досягнення певної мети. Це допомагає чітко визначити, які саме об’єкти залучені до реалізації сценарію, у якій послідовності відбуваються дії, та яку інформацію вони передають.

Ключовою перевагою діаграм послідовності є те, що вони дозволяють на етапі проєктування виявити логіку взаємодії компонентів ще до початку реалізації. Це створює основу для побудови майбутніх класів, методів і обробників подій.

У центрі діаграми — **часова вісь**, яка йде згори вниз. Вона дозволяє побачити, у якому порядку відбуваються виклики: чим вище на діаграмі розміщене повідомлення, тим раніше воно було передано. Кожен учасник взаємодії має **лінію життя** — вертикальну пунктирну лінію, яка позначає період існування об'єкта під час сценарію.

Обмін повідомленнями між об’єктами зображується у вигляді горизонтальних стрілок, що позначають виклик методів або передачу даних. Для позначення моментів активності об'єкта використовується **фокус управління** (activation bar) — витягнутий прямокутник, який вказує, коли об'єкт виконує певну дію.

Таким чином, діаграма послідовності є потужним інструментом моделювання, який дозволяє візуалізувати поведінку системи, виявити потенційні помилки у логіці взаємодій та краще підготуватися до реалізації програмного забезпечення.

Діаграма послідовності, що наведена в Додатку В (сторінка 1), ілюструє послідовність взаємодії між ключовими об’єктами програмного забезпечення, розробленого для автоматизованого управління діяльністю студентських наукових гуртків.

Актори діаграми

1. **Директор** - керівник організації, що ініціює процеси та отримує звітність.
2. **Керівник** - співробітник, що працює з системою та виконує операції обробки даних.
3. **Учасник** - користувач, який взаємодіє з системою на нижчому рівні доступу.

Системні компоненти

1. **Система** - програмна платформа, що координує взаємодію між акторами та базою даних.
2. **БД (База даних)** - сховище інформації, з яким відбувається обмін даними.

Основні процеси взаємодії

1. **Додавання/редагування даних**:
   - процес починається з директора, який ініціює додавання/редагування запису;
   - система обробляє запит та перевіряє коректність вхідних даних;
   - відбувається передача коректності введених даних до бд;
   - бд виконує операції збереження даних;

- інформація про успішне завершення операції повертається через систему до директора.

2. **Перевірка та обробка даних різними акторами**:
   - кожен актор (директор, керівник, учасник) має різні рівні доступу та різні операції;
   - для кожного з них відбувається обмін повідомленнями з системою та бд;
   - основні типи повідомлень включають:
     - "інформація успішно збережена";
     - "підтвердження отримання запиту";
     - "передача коректності введених даних";
     - "збереження даних".
3. **Процеси запитів та отримання даних**:
   - актори можуть запитувати інформацію з бд;
   - процес проходить через систему, яка формує запит до бд;
   - бд обробляє запит та повертає дані;
   - система форматує та презентує дані відповідному актору.
4. **Заключні операції**:
   - на завершальних етапах діаграми відображаються операції з запитом до бд;
   - відбувається отримання даних, їх відображення та підтвердження завершення операцій;
   - остаточні результати передаються відповідним акторам.

Особливості взаємодії

1. **Послідовність взаємодії** - діаграма чітко демонструє часову послідовність обміну повідомленнями між акторами та системними компонентами.
2. **Валідація даних** - на діаграмі представлені процеси перевірки коректності введених даних перед їх збереженням у БД.
3. **Зворотній зв'язок** - система забезпечує акторів підтвердженнями про успішне виконання операцій.
4. **Рівні доступу** - різні актори мають різні права та можливості взаємодії з системою та даними.

   На цій діаграмі послідовності можна побачити "кроки" взаємодії між користувачем, системою та базою даних. Вона показує порядок дій і обмін інформацією, що необхідно для розробки компонентів програми.

Висновок до розділу 1

У результаті проведеного системного аналізу предметної області було досліджено ключові аспекти функціонування системи управління студентськими науковими гуртками, орієнтованої на автоматизацію адміністративних процесів та покращення комунікації між учасниками освітнього процесу. Проаналізовано існуючі підходи до організації гурткової діяльності та виявлено потребу в автоматизації рутинних операцій, централізованому зберіганні даних та спрощенні формування звітності.

Було визначено три основні ролі користувачів системи (директор, керівник гуртка та учасник), їх функціональні обов'язки та потреби. На основі цього сформовано комплекс функціональних вимог, що охоплюють облік гуртків та учасників, управління новинами та оголошеннями, календарне планування, облік матеріалів та формування звітності. Також визначено нефункціональні вимоги, що стосуються зручності використання, безпеки, надійності, масштабованості та документування системи.

Розроблені UML-діаграми (прецедентів, активності, послідовності) дозволили візуалізувати динамічні та статичні аспекти майбутньої системи, забезпечити чітке розуміння взаємодії компонентів та послідовності виконання основних операцій. Завдяки цьому створено надійне підґрунтя для подальшого проєктування архітектури програмного забезпечення, розробки бази даних та реалізації користувацького інтерфейсу системи управління студентськими науковими гуртками.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

ER-модель (Entity-Relationship Model) — це концептуальна модель даних, яка дозволяє відобразити предметну область у вигляді сукупності сутностей, їх атрибутів та взаємозв'язків між ними. Завдяки своїй візуальній природі ця модель є ефективним інструментом для аналітичного осмислення структури даних ще до початку технічної реалізації. Вона допомагає чітко сформулювати бізнес-вимоги до бази даних і виявити потенційні проблеми, пов'язані з надлишковістю або неоднозначністю інформації.

У рамках проєкту з розробки системи обліку та управління діяльністю гуртків було створено логічну модель, що охоплює такі ключові елементи: **керівники, гуртки, учасники, новини, матеріали та події**. Кожна з цих сутностей представляє реальні об'єкти, з якими взаємодіє користувач або адміністратор системи. Взаємозв'язки між цими об'єктами дозволяють коректно організувати логіку доступу, обліку та звітності, забезпечуючи при цьому гнучкість масштабування і простоту супроводу системи.

Інструмент моделювання: CA ERwin Data Modeler

Для побудови ER-діаграми було використано **CA ERwin Data Modeler** — потужне програмне середовище для створення структурованих моделей баз даних. Цей інструмент дозволяє проєктувати логічні та фізичні моделі, автоматизуючи процес побудови бази даних з урахуванням усіх вимог і зв'язків.

Серед основних переваг ERwin варто відзначити:

1. **Глибоку кастомізацію сутностей**: можливість точно визначати всі атрибути, типи даних, обмеження та правила валідації.

2. **Автоматичне генерування SQL-структур**: інструмент може створити готові до використання сценарії створення бази даних.
3. **Документування моделі**: система дозволяє зберігати повну документацію до кожної частини моделі, що зручно для командної роботи та підтримки проєкту в майбутньому.
4. **Можливість трасування зв'язків**: це особливо корисно при аналізі складних залежностей або зміні бізнес-вимог.

У процесі реалізації дипломного проєкту **ERwin** був використаний для побудови графічного відображення логічної структури бази даних, що дозволило на етапі проєктування забезпечити правильну організацію даних, дотримання нормалізації та забезпечення зв'язності таблиць.

Таким чином, застосування **CA ERwin Data Modeler** не лише прискорило процес створення бази даних, але й підвищило якість проєктування, що в довгостроковій перспективі позитивно впливає на стабільність та розширюваність інформаційної системи.

ER-діаграму, що ілюструє основні сутності та їх взаємозв'язки в розробленій системі, наведено на рис. 2.1.

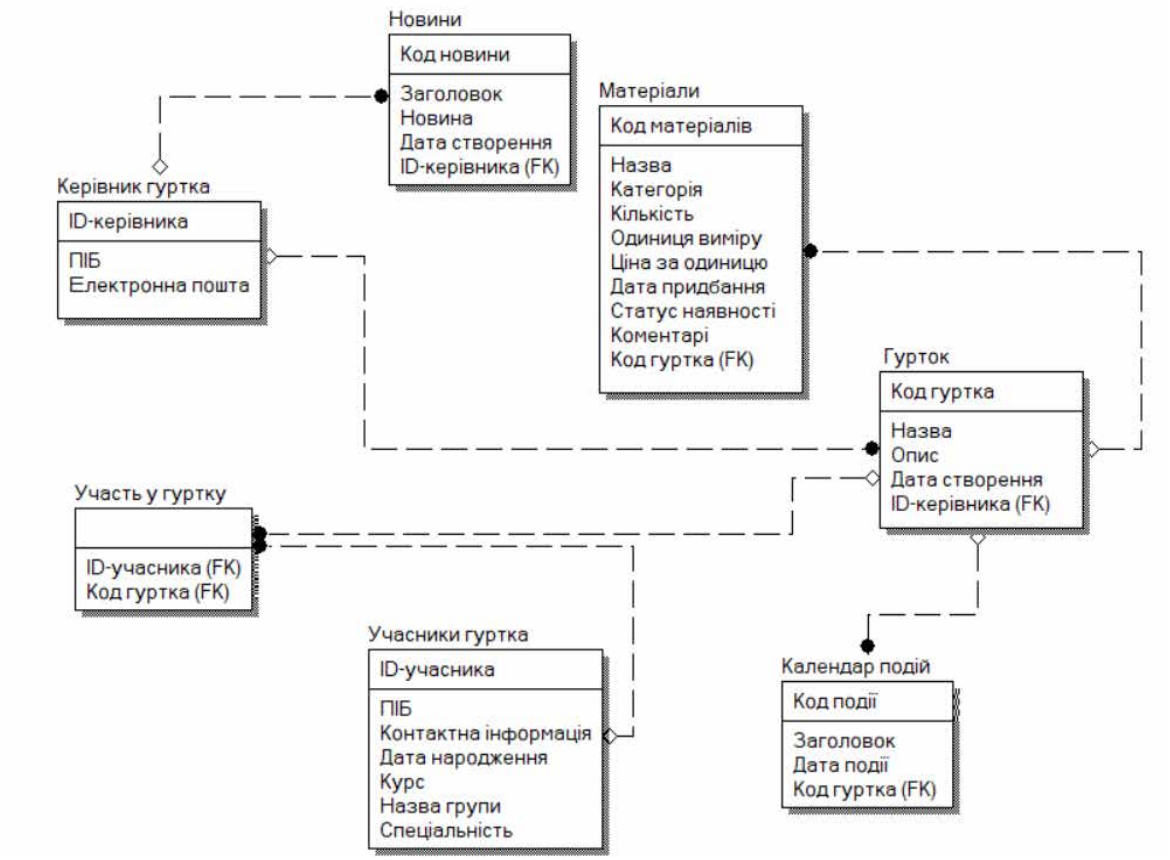


Рис. 2.1 ER-діаграма

Опис основних сутностей системи

1. Керівник гуртка: ця сутність зберігає інформацію про осіб, відповідальних за управління гуртками. Вона включає унікальний ідентифікатор, повне ім’я керівника та його електронну пошту. Один керівник може бути відповідальним за кілька гуртків, що реалізується через зв’язок один-до-багатьох із сутністю "гурток". Також керівник асоціюється із публікацією новин, виступаючи автором інформаційних повідомлень у системі.

2. Гурток: центральна сутність, яка визначає організаційно-структурну одиницю системи. містить назву гуртка, його опис, дату створення та посилання на керівника, що ним опікується. гурток є пов’язуючим вузлом для багатьох інших сутностей — зокрема, до нього належать події з календаря, матеріали, новини, а також участь студентів.

3. Учасник гуртка: ця сутність відображає персоналізовану інформацію про осіб, які беруть участь у гуртках. До атрибутів належать ПІБ, дата народження, контактна інформація, курс, група та спеціальність. Учасник може бути залучений до декількох гуртків, що реалізовано через проміжну таблицю "Участь у гуртку".

4. Участь у гуртку: допоміжна сутність, яка моделює зв’язок "багато-до-багатьох" між учасниками та гуртками. Вона містить два зовнішні ключі — на учасника та гурток, — що дозволяє системі реєструвати належність студентів до кількох об’єднань одночасно. Такий підхід виключає дублювання даних і забезпечує гнучке управління списками учасників.

5. Матеріали: сутність описує ресурси, що використовуються в межах діяльності гуртків: канцелярія, обладнання, техніка тощо. Атрибути включають назву, категорію, одиницю виміру, кількість, ціну за одиницю, дату придбання, статус наявності та коментар. Кожен запис прив’язаний до конкретного гуртка, що дозволяє вести облік матеріального забезпечення окремо по кожному напрямку.

6. Календар подій: ця сутність акумулює інформацію про заходи, які організовуються в межах певного гуртка. Містить унікальний код події, її назву, дату проведення та зовнішній ключ до гуртка. Завдяки цьому можна легко фіксувати та відображати графік активностей для кожного гуртка.

7. Новини: інформаційна сутність, призначена для створення повідомлень, що відображають новини, оголошення або досягнення гуртка. Складається із заголовка, тексту новини, дати публікації та посилання на керівника, який є її автором. Це дозволяє зберігати контекст і походження кожної інформаційної одиниці в системі.

Аналіз відповідності моделі вимогам третьої нормальної форми (ЗНФ)

ER-модель відповідає усім вимогам нормалізації баз даних, зокрема принципам третьої нормальної форми (ЗНФ):

Перша нормальна форма (1НФ)

Усі таблиці містять атомарні значення: кожне поле зберігає лише одне значення, без списків або вкладених структур. Наприклад, контактна інформація учасника представлена як окремий рядок, а не масив контактів. Повторювані групи відсутні.

Друга нормальна форма (2НФ)

Модель не містить часткових залежностей неключових атрибутів від частини складеного ключа. У таблиці "Участь у гуртку", яка має складений ключ із двох зовнішніх ключів (ID-учасника та Код гуртка), не зберігається жодної інформації, окрім ключових полів, що виключає порушення 2НФ. Усі інші таблиці мають прості первинні ключі, і всі їх атрибути напряму залежать від них.

Третя нормальна форма (3НФ)

У системі не виявлено транзитивних залежностей — усі неключові атрибути залежать безпосередньо від первинного ключа своєї таблиці:

- у "Учасник гуртка" спеціальність напряму залежить від ID-учасника, а не, наприклад, від назви групи;
- у "Матеріали" такі поля, як ціна, кількість, статус і коментар, залежать тільки від коду матеріалу, не між собою;
- у "Гурток" назва, опис і дата створення не залежать один від одного, а лише від ключа "Код гуртка".

Таким чином, структура бази даних:

- мінімізує дублювання інформації;
- дотримується принципів логічної узгодженості;
- легко масштабована та підтримувана;
- не містить надлишкових або транзитивних залежностей.

Побудована логічна модель бази даних є чітко структурованою, узгодженою з вимогами предметної області й відповідає третій нормальній формі. Це забезпечує ефективну роботу системи, спрощує її підтримку та дає можливість легко адаптувати її до нових функціональних вимог.

2.2 Діаграма класів та кооперацій

Діаграма класів — це один із типів діаграм у мові UML (Unified Modeling Language), який служить для графічного відображення структури програмної системи. Вона допомагає показати основні класи, які входять до складу системи, а також їхні властивості та методи. За допомогою такої діаграми можна чітко побачити, які дані зберігає кожен клас і які операції він виконує.

Крім того, діаграма класів дозволяє візуалізувати різні типи взаємозв’язків між класами — такі як асоціації, агрегації, композиції та наслідування. Це сприяє кращому розумінню архітектури системи як окремим розробникам, так і всій команді загалом.

Вона також виконує роль важливого інструменту при безпосередній розробці — класова діаграма служить своєрідним шаблоном для написання програмного коду. Завдяки цьому процес створення класів і їх взаємодії стає більш структурованим і послідовним.

Нарешті, така діаграма допомагає систематизувати логіку взаємодії об’єктів, що є корисним як для документування системи, так і для подальшого тестування, забезпечуючи більш якісну розробку програмного продукту.

Подана діаграма класів, розміщена в Додатку В (сторінка 2), детально ілюструє архітектуру інформаційної системи для адміністрування діяльності гуртків, охоплюючи ключових учасників процесу, їхні функціональні обов'язки та логічні взаємозв'язки між об'єктами системи. До основних класів моделі належать: *Директор, Керівник гуртка*, *Учасник гуртка*, *Гурток*, *Матеріали*, *Звіт*, *Календар подій* і *Новини/оголошення*. Ці класи спільно забезпечують логіку

обліку користувачів, наповнення подій, зберігання звітної інформації та управління ресурсами гуртків.

Таблиця основних класів та їх функціональності наведена нижче в табл. 2.1.

Таблиця 2.1

Таблиця основних класів та їх функціональності

№	**Клас**	**Основні властивості**	**Ключові методи**
1	Директор	ID	авторизація, реєстрація користувача, керування правами доступу, створення звітів
2	Керівник гуртка	ID, ПІБ, Email	формування складу гуртка, створення новин, формування подій
3	Учасник гуртка	ID, ПІБ, контактні дані, дата народження, курс, група, спеціальність	перегляд гуртків, керівників, новин, календаря подій
4	Гурток	Назва, опис, дата створення, кількість учасників	додавання матеріалів, отримання списку учасників
5	Матеріали	ID, назва, категорія, одиниця, ціна, дата, статус тощо	отримання доступу до матеріалів
6	Звіт	Назва, дата формування, тип, текст звіту	генерація звіту
7	Календар подій	ID, опис, час	отримання доступу до подій
8	Новини/оголошення	ID, назва, опис, дата	перегляд новин

Першим ключовим класом є **«Директор»**, який виконує роль адміністратора системи. Він має таку властивость, як ID. До основних методів цього класу належать: авторизація користувача, його реєстрація, управління правами доступу, формування подій та створення звітів. Директор має повний контроль над користувачами та інформаційною структурою системи.

Наступним є клас **«Керівник гуртка»**, який відповідає за безпосереднє управління гуртком. Серед його атрибутів — ID, ПІБ і Email. Методи класу дозволяють формувати склад гуртка, створювати новини або оголошення, додавати події та оцінювати учасників. Цей клас забезпечує внутрішню координацію діяльності гуртка.

«Учасник гуртка» — ще один важливий клас, що представляє звичайного користувача системи. Він має базові властивості: ID, ПІБ , контактні дані, дату народження, курс, група та спеціальність. Основні методи передбачають перегляд інформації про гуртки, їхніх керівників, новини, розклад занять та інші події. Цей клас є кінцевим споживачем даних у системі.

Клас **«Гурток»** зберігає інформацію про кожен гурток: його назву, опис, дату створення та кількість учасників. До методів цього класу належать додавання матеріалів, що використовуються в гуртку, та отримання списку його учасників. Він є основною одиницею організаційної структури системи.

Окремим важливим компонентом є клас **«Матеріали»**, який містить детальну інформацію про ресурси, що використовуються в межах гуртків: назву, категорію, одиниці виміру, кількість, ціну, дату придбання, статус наявності та інше.

Клас **«Звіт»** формується директором і включає такі поля, як назва звіту, дата формування, тип і текст. Метод дозволяє генерувати унікальний ідентифікатор звіту, що забезпечує збереження історії діяльності гуртків.

«Календар подій» — клас, що зберігає інформацію про події: їх опис і час проведення. Завдяки методам цього класу користувачі можуть переглядати заплановані заходи.

Останнім є клас **«Новини/оголошення»**, який містить ID, назву, опис і дату створення оголошення. Основний метод класу забезпечує перегляд новин усіма учасниками системи.

Діаграма кооперації призначена для відображення конкретної взаємодії між об’єктами системи в межах певного сценарію або функціонального процесу. Вона дозволяє зрозуміти, як екземпляри класів взаємодіють у часі та в якому порядку передають повідомлення одне одному, щоб досягти визначеної мети.

На діаграмі зображуються об’єкти (що є реалізацією класів) і зв’язки між ними, які вказують на можливість взаємодії. Повідомлення, що передаються між об’єктами, позначаються стрілками з номерами або порядком викликів, що дозволяє простежити послідовність дій. Такі стрілки можуть містити підписи, що описують методи або операції, які викликаються в процесі.

Особливістю цієї діаграми є її динамічний характер — вона не демонструє всю архітектуру системи, як це робить діаграма класів, а концентрується лише на окремому випадку взаємодії, який є релевантним для певної задачі. При цьому структура зв’язків може змінюватися залежно від контексту — той самий об’єкт може брати участь у кількох кооперaціях з різними ролями.

Такий тип діаграми є корисним для моделювання логіки процесів, деталізації варіантів використання та верифікації сценаріїв взаємодії. Для аналітиків, розробників і тестувальників це — наочний інструмент, що дозволяє побачити, як система працює в конкретних ситуаціях, де важлива не лише структура, а й послідовність кроків.

Нижче наведено діаграми кооперацій, які ілюструють взаємодію між основними об’єктами системи (рис. 2.3–2.5).

Рис. 2.3 Сценарій реєстрації учасника в системі

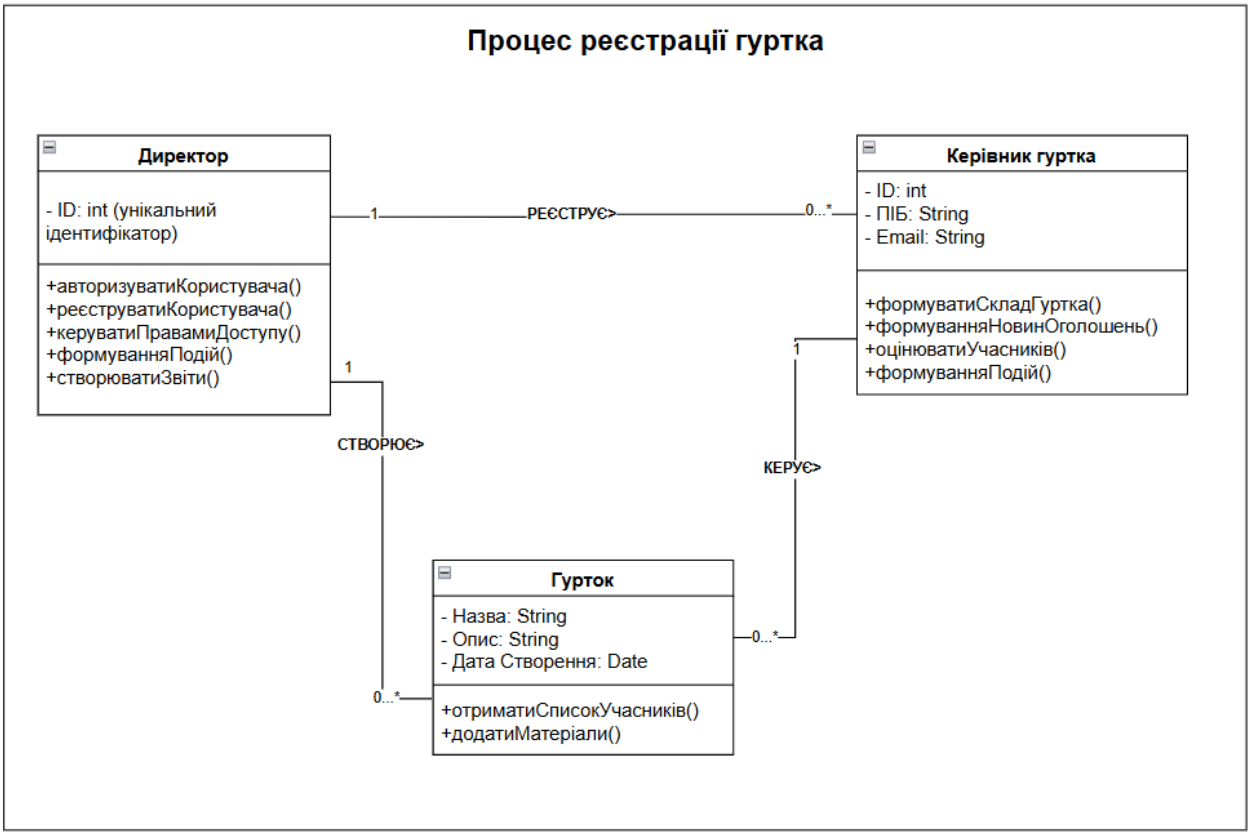


Рис. 2.4 Сценарій реєстрації гуртка в системі

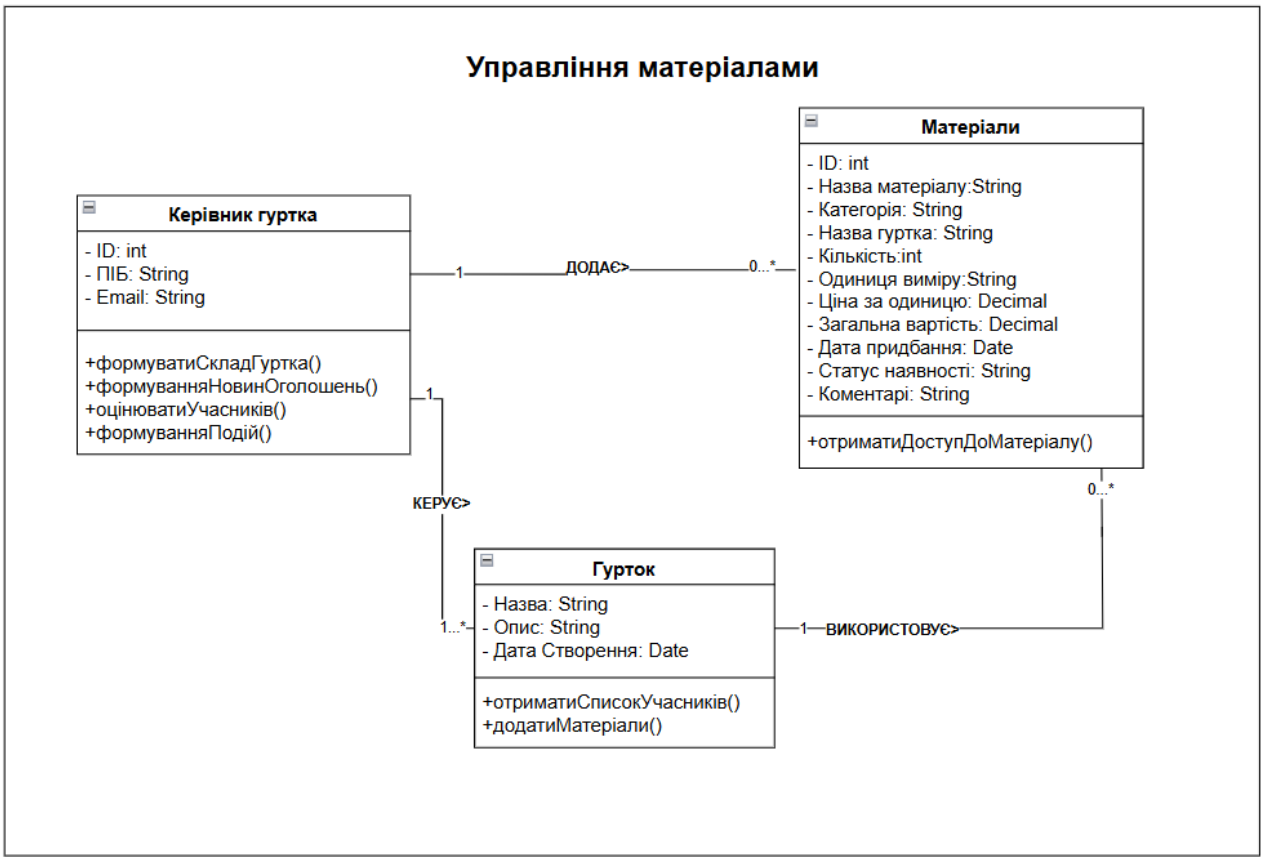


Рис. 2.5 Сценарій управління матеріалами

2.3 Діаграма пакетів

Мова моделювання UML надає засоби для моделювання складних систем, зокрема через діаграми пакетів, які дозволяють структуризувати модель, об’єднавши пов’язані елементи у тематичні блоки. Кожен пакет виступає як логічна область, яка містить класи, інтерфейси або інші компоненти системи, що мають спільне призначення або функціональність. Елементи, розміщені всередині пакета, є його складовими частинами. Якщо сам пакет видаляється, автоматично зникають і всі об'єкти, які він охоплює.

Під час проектування масштабних програмних рішень надзвичайно важливо розділяти систему на ізольовані функціональні частини. Завдяки пакетам забезпечується логічне групування об’єктів, що полегшує аналіз архітектури, підвищує зручність навігації та підтримки коду.

На представленій діаграмі пакетів, яка розміщена у Додатку В (сторінка 3), відображається логічна структура програмного забезпечення для керування студентськими науковими гуртками, в якій чітко виділено окремі рівні взаємодії: від користувацького інтерфейсу до доступу до бази даних та сервісів безпеки.

Вся система згрупована в межах одного основного пакета, що включає декілька логічно впорядкованих рівнів, кожен із яких відповідає за окрему частину функціональності системи адміністрування студентських наукових гуртків.

1. Рівень інтерфейсу (презентаційний)

Цей шар призначений для взаємодії користувача з програмою. Він містить модуль Інтерфейс, який реалізує візуальні компоненти системи та забезпечує відображення даних, отриманих із нижчих рівнів. Через цей інтерфейс користувачі можуть запускати функції програми, наприклад, переглядати події, редагувати матеріали чи створювати звіти. Інтерфейс має залежності до усіх основних модулів логіки системи.

2. Рівень логіки (бізнес-логіка програми)

Тут зосереджена основна функціональна частина системи, що реалізує правила роботи гуртків. Цей рівень включає такі підсистеми:

- **управління користувачами** — обробка інформації про директорів, керівників, учасників;
- **звітність** — генерація звітів на основі інформації з бази даних;
- **управління матеріалами** — облік і контроль ресурсів, необхідних для гуртків;
- **управління гуртками** — створення, редагування та адміністрування гуртків;

- **управління новинами / оголошеннями** — створення інформаційних повідомлень;
- **управління подіями** — ведення календаря та запланованих заходів.

Кожен з цих модулів взаємодіє як з інтерфейсом, так і з системою збереження даних.

3. Рівень доступу до даних

Цей рівень відповідає за зв'язок із системами зберігання інформації. Складається з таких елементів:

- **база даних** — зберігає всю необхідну інформацію: дані про користувачів, матеріали, гуртки, події тощо;
- **API** — програмний інтерфейс, який надає доступ до деяких функцій або даних у структурований спосіб, зокрема для модулів звітності;
- **сервіси безпеки** — забезпечують контроль доступу, автентифікацію користувачів та захист даних.

Усі компоненти логіки програми взаємодіють із цим рівнем через запити до бази даних або використання API у випадках, де необхідна проміжна обробка чи контроль доступу.

2.4 Діаграма компонентів

UML-діаграма компонентів є важливим інструментом для моделювання архітектури складних інформаційних систем. Вона надає можливість візуалізувати, з яких саме частин складається програмне забезпечення, як ці частини взаємодіють одна з одною, та яким чином реалізується функціональність системи через розподіл відповідальностей між окремими модулями.

Кожен компонент у такій діаграмі являє собою автономний фрагмент системи, який виконує певну логічну роль. Такі компоненти можуть включати в себе підсистеми, бізнес-логіку або інтерфейсні елементи — усе залежить від

того, яку функцію виконує модуль. Однією з ключових властивостей компонентів є їхня взаємодія через чітко визначені точки доступу — інтерфейси. Це дозволяє зберігати слабке зв’язування між модулями, що позитивно впливає на гнучкість і масштабованість системи.

Діаграма компонентів не лише демонструє структуру системи, а й відображає зв’язки між окремими її частинами, їхню ієрархію, залежності та потоки даних. Такий підхід дозволяє ефективно організувати взаємодію між модулями, спростити процес розробки та обслуговування програмного забезпечення, а також мінімізувати потенційні конфлікти при внесенні змін.

У контексті розробки автоматизованої системи управління студентськими науковими гуртками діаграма компонентів, розміщена в Додатку В (сторінка 4), описує поділ архітектури на логічні рівні, що включають: інтерфейс користувача, прикладну логіку, модулі доступу до даних та сервіси авторизації. Кожен компонент системи має власну роль, чітко обмежену зоною відповідальності, і взаємодіє з іншими модулями згідно визначених сценаріїв.

Опис архітектурних рівнів програмного забезпечення

Архітектура автоматизованої системи управління студентськими науковими гуртками реалізована з урахуванням модульного підходу та розподілена на окремі логічні блоки, кожен з яких виконує певну функцію в межах загальної системи. Всі модулі згруповані відповідно до призначення, що дозволяє забезпечити гнучкість, розширюваність і зручне обслуговування.

1. Інтерфейс користувача

Цей блок забезпечує взаємодію кінцевого користувача з системою. Він включає:

- **введення логіна і пароля** — базова форма автентифікації користувача;
- **взаємодія з формами** — робота з графічним інтерфейсом, що дозволяє подавати запити до системи та відображати відповіді.

Через інтерфейс користувача здійснюється доступ до основних функцій: створення подій, керування учасниками, перегляд новин тощо.

2. Система авторизації

Відповідає за перевірку облікових даних користувачів та передачу інформації про них у відповідні модулі. Складається з:

- **перевірки облікових даних** — автентифікація за логіном і паролем;
- **передачі інформації про користувача** — для подальшої ідентифікації в системі.

Це ізольований модуль без доступу до бізнес-логіки, який виконує роль шлюзу до системи.

3. Модулі роботи з даними

Ці компоненти реалізують конкретну функціональність системи. Кожен модуль має свою зону відповідальності:

- **модуль роботи з гуртками**: додавання, редагування, перегляд списку гуртків;
- **модуль роботи з учасниками**: управління даними учасників (створення, видалення, перегляд);
- **модуль роботи з керівниками**: керування записами про керівників гуртків;
- **модуль роботи з подіями**: створення, редагування та перегляд подій;
- **модуль роботи з новинами**: публікація, редагування та перегляд інформаційних повідомлень;
- **модуль звітності**: формування та отримання звітної інформації.

Кожен з цих модулів працює з відповідними таблицями в базі даних і пов’язаний із логікою обробки даних.

4. Логіка обробки даних

Це внутрішній рівень, який відповідає за управління інформаційними об'єктами та бізнес-правилами:

- **управління даними про події, гуртки, учасників, керівників, новини** — виконує перевірку, обробку й оновлення інформації;
- **формування звітної інформації** — агрегування даних для подальшого виводу.

Цей рівень діє як “мозок” системи, який приймає запити від модулів і трансформує їх у відповідні операції над даними.

5. База даних

На рівні зберігання реалізовано взаємодію з такими сутностями: Гуртки, Учасники, Керівники, Події, Новини, Звіти, Облікові записи. Кожен модуль системи має зв’язок із відповідною таблицею, забезпечуючи повноцінну роботу з актуальною інформацією.

Потік взаємодії:

1. Користувач вводить дані через інтерфейс.
2. Система авторизації перевіряє облікові дані.
3. Обробка дій виконується у відповідному модулі.
4. Модуль звертається до логіки обробки даних.
5. Після обробки запит передається до бази даних.
6. Результат повертається до користувача через інтерфейс.

Перевагами такої архітектурної структури є чітке розмежування функціональності між окремими модулями, що сприяє прозорості реалізації системи; зручність у підтримці та масштабуванні, яка дозволяє безболісно розширювати можливості програмного забезпечення; потенціал для повторного використання окремих компонентів у майбутніх проєктах; а також централізоване управління як даними, так і безпековими механізмами, що забезпечує контрольовану та захищену роботу всієї системи.

Висновки до розділу 2

Використання сучасних моделей представлення даних, таких як ER-моделі та діаграми уніфікованої мови моделювання (UML), є важливою частиною процесу проєктування складних інформаційних систем. Ці інструменти дозволяють на етапі аналізу структурувати вимоги, виявити потенційні проблеми в логіці системи й створити прозору архітектуру ще до написання коду.

Зокрема, ER-моделювання дозволяє ефективно спроектувати базу даних: воно допомагає логічно впорядкувати сутності, визначити зв’язки між ними та створити коректну структуру сховища даних. Це знижує ризик логічних помилок у майбутньому та полегшує спільну роботу команди розробників.

UML забезпечує потужний набір візуальних засобів для моделювання як статичних, так і динамічних аспектів системи. Використання діаграм класів, кооперації, компонентів чи пакетів дає змогу розбити систему на чітко структуровані частини, виявити залежності та краще зрозуміти логіку взаємодії між модулями.

Діаграми компонентів, зокрема, дозволяють наочно зобразити архітектуру програмного забезпечення, описати взаємозв’язки між його модулями та оцінити рівень ізольованості й повторного використання окремих частин. Така візуалізація є корисною не лише для розробників, а й для аналітиків, тестувальників і всіх учасників розробки, оскільки дає цілісне уявлення про структуру системи.

Водночас діаграми пакетів забезпечують логічне групування елементів системи, що сприяє кращій організації коду, полегшує тестування, а також спрощує підтримку та масштабування. Хоча вони не зменшують складність залежностей, пакети дозволяють зробити їх очевидними та керованими.

Загалом, застосування ER-моделей та UML-діаграм у процесі розробки ПЗ, зокрема для побудови системи управління студентськими науковими гуртками,

сприяє створенню продуманої, масштабованої та зрозумілої архітектури, яка забезпечує стабільну роботу та гнучке вдосконалення системи в майбутньому.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління базою даних

Система управління базами даних (СУБД) — це спеціалізоване програмне забезпечення, яке дозволяє створювати, змінювати, зберігати та ефективно управляти великими обсягами структурованої інформації. Вона забезпечує надійне збереження даних, захист доступу до них, а також дозволяє здійснювати швидкий пошук і обробку інформації через зручний інтерфейс взаємодії.

Основна роль СУБД у будь-якій інформаційній системі полягає в централізованому управлінні даними, уникненні дублювання інформації, забезпеченні її узгодженості, цілісності та безпеки. Для системи автоматизованого управління студентськими науковими гуртками це означає точне ведення обліку учасників, керівників, гуртків, подій, новин та звітів, а також контроль доступу до кожного з цих об’єктів.

СУБД дозволяє реалізувати всі ключові операції роботи з даними — збереження, оновлення, вибірку, сортування та фільтрацію. Наприклад, під час формування звітів або пошуку подій у межах конкретного гуртка система звертається до бази даних, де знаходить та обробляє потрібну інформацію. Завдяки оптимізації запитів забезпечується швидкий відгук навіть при великому обсязі записів.

Крім того, СУБД підтримує механізми контролю доступу — автентифікацію користувача, авторизацію за ролями, а також шифрування та журналювання дій. Це особливо важливо у випадках, коли йдеться про персональні дані учасників, контактну інформацію чи управлінські звіти, що повинні бути захищені від несанкціонованого доступу.

СУБД поділяються за різними ознаками: за моделлю даних (реляційні, документоорієнтовані, графові тощо), за принципом розгортання (локальні або хмарні), за призначенням (оперативні та аналітичні) та за способом доступу до даних (SQL і NoSQL). У межах даного проєкту використовується **реляційна система керування базами даних Microsoft SQL Server**, яка є одним з найбільш стабільних і продуктивних рішень для організації роботи з табличними структурами.

Microsoft SQL Server забезпечує широкі можливості для реалізації логіки запитів, перевірки цілісності даних, збережених процедур та автоматизованих операцій резервного копіювання. Саме завдяки цим функціям дана СУБД була обрана для реалізації системи: вона дозволяє ефективно обслуговувати всі основні функції проєкту, включаючи обробку користувачів, керівників, подій, гуртків, новин і звітності.

Таким чином, Microsoft SQL Server у цьому випадку виступає як надійна основа для збереження і управління критично важливою інформацією в автоматизованій системі, забезпечуючи високу продуктивність, масштабованість та захист даних.

Переваги MS SQL Server:

- інтеграція з продуктами Microsoft (Visual Studio, .NET);
- підтримка збережених процедур і тригерів;
- висока продуктивність і стабільність;
- зручне графічне середовище (SSMS);
- гнучка система доступу та безпеки.

Недоліки:

- платна ліцензія (для повної версії);
- велике споживання ресурсів;
- обмежена підтримка не-Windows платформ;
- потреба в спеціальних знаннях для налаштування та адміністрування.

Як засіб проєктування бази даних у даній системі було обрано **СУБД Microsoft SQL Server**, оскільки вона забезпечує стабільну роботу, зручність у використанні та достатню функціональність для реалізації завдань автоматизованої системи управління студентськими науковими гуртками.

3.2 Створення бази даних

На етапі реалізації структури бази даних для системи управління студентськими науковими гуртками першочергово було виконано створення нової бази з назвою Art_clubs (*Automated Research and Training Clubs System*), яка згодом стала основою для всієї системи збереження інформації (рис. 3.1).

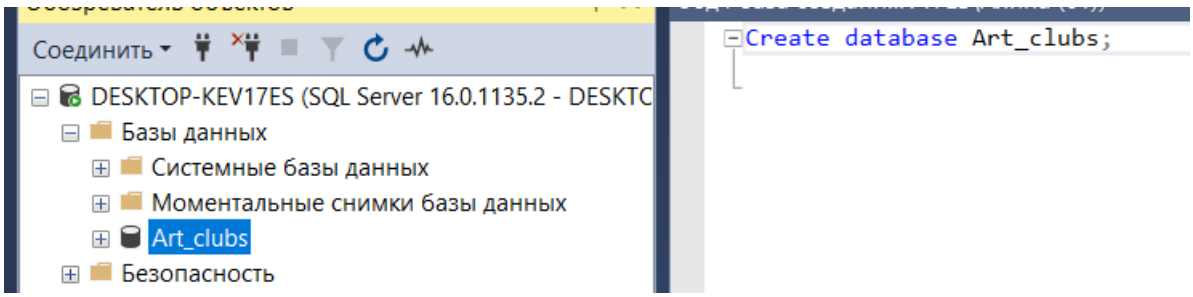


Рис. 3.1 Створення бази даних Art_clubs

Після створення бази даних було розроблено набір таблиць, у яких зберігаються всі ключові сутності — гуртки, учасники, керівники, матеріали, новини тощо. Створення таблиць здійснювалося за допомогою SQL-запитів, де були прописані необхідні поля, типи даних, обмеження (наприклад, первинні ключі та унікальні значення). В Додатку А (сторінки 1-3) розміщується повний лістинг коду створення таблиць бази даних. На рис. 3.2 представлено фрагмент коду, що демонструє створення таблиці Gourt, яка зберігає інформацію про гуртки, включаючи їхній код, назву, опис, дату створення та кількість учасників.

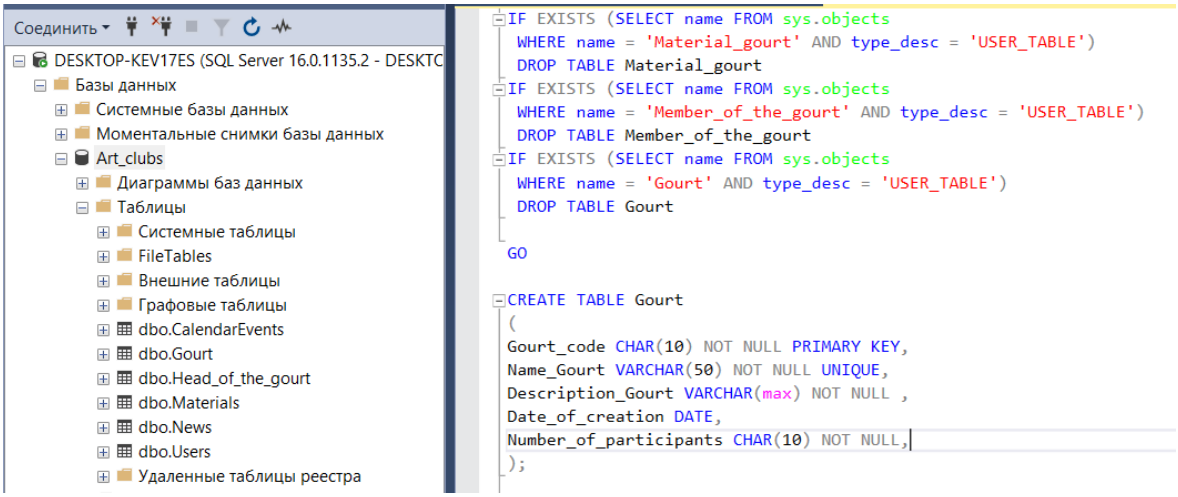


Рис. 3.2 SQL-запит для створення таблиці Gourt

Наступним етапом стала організація системи безпеки доступу. У середовищі Microsoft SQL Server для цього були створені ролі з різними рівнями доступу до таблиць. Наприклад, для ролі Director було налаштовано права на вибірку, вставку, оновлення та видалення даних у конкретних таблицях, що відповідають функціональності директора системи (рис. 3.3). Це дозволяє розмежувати дії користувачів залежно від їхньої ролі в системі та запобігти несанкціонованому доступу до критичних даних.

У Додатку А (сторінки 4–8) подано повний код для створення ролей та призначення їм відповідних прав доступу.

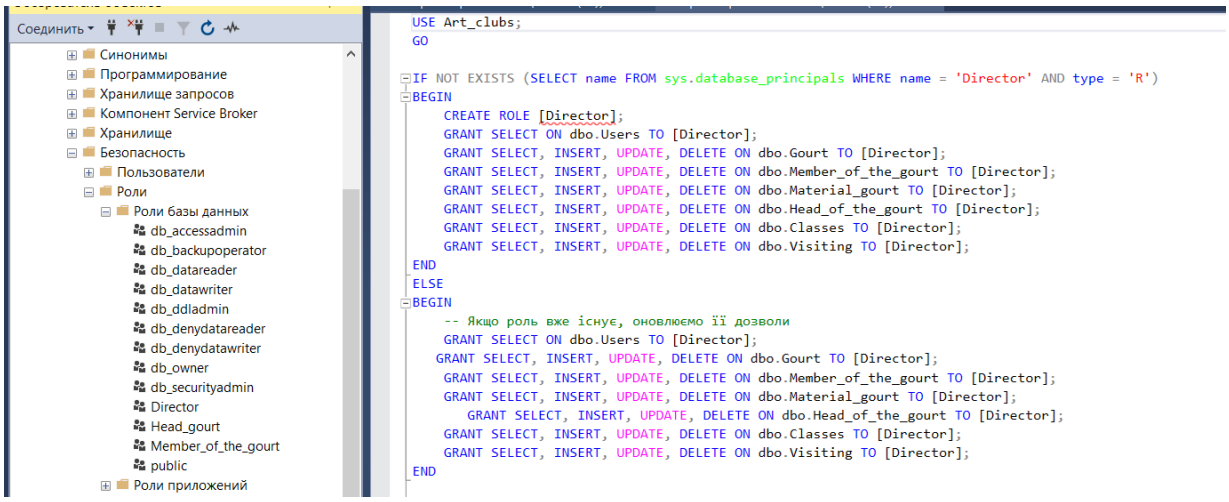


Рис. 3.3 Налаштування ролей та прав доступу в базі даних

Таким чином, у процесі реалізації бази даних було забезпечено не лише логічну структуру даних, але й основи для їхньої безпечної та контрольованої обробки.

До основної схеми таблиць, у базі даних реалізовано набір уявлень (views), які відіграють важливу роль у логічному представленні даних. Уявлення — це збережені SQL-запити, які формують зручну для аналізу інформацію без дублювання самих даних. Вони дозволяють отримувати потрібні зведення та аналітичні зрізи, спрощують доступ до взаємопов’язаних таблиць та підвищують ефективність взаємодії з інформаційною системою.

У контексті побудови автоматизованої системи обліку гуртків і пов’язаних з ними ресурсів, було створено три основні уявлення:

1. **Gourt_Expenses** — уявлення, що здійснює підрахунок загальних витрат на матеріали для кожного гуртка. Воно агрегує дані з таблиці матеріалів і дозволяє адміністрації оперативно контролювати фінансові витрати, пов’язані з конкретними гуртками. Така звітність є важливою для формування бюджету та прозорого обліку.
2. **GourtMembers** — уявлення, яке відображає інформацію про гуртки та їх учасників. Зокрема, воно поєднує дані про назву гуртка, ім’я та дату народження учасника. Це уявлення надає зручний доступ до

персоналізованої інформації, що використовується для аналітики складу, формування списків або комунікації з учасниками.

3. **GourtHead** — уявлення, що містить інформацію про кожен гурток та його керівника. Виводяться назва гуртка, ім’я голови та його електронна адреса. Уявлення дозволяє легко ідентифікувати відповідальних осіб і використовується при формуванні повідомлень, погодженні заходів або розподілі завдань.

Реалізація цих уявлень дозволяє зручно витягувати найнеобхіднішу інформацію для управління гуртками без необхідності складати багаторівневі запити вручну. Вони створюють зручний логічний рівень над фізичною структурою бази даних, що є особливо корисним для звітності, контролю та оптимізації рутинних адміністративних процесів. SQL-коди створення уявлень наведено у Додатку А (сторінка 8).

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

При створенні програмного забезпечення одним із ключових рішень є вибір мови програмування, оскільки від цього залежать функціональні можливості, продуктивність, зручність розробки та масштабованість проєкту. На початковому етапі були проаналізовані дві мови програмування — C++ та C#, які мають широке поширення та активно використовуються в галузі розробки прикладних систем.

Мова C++ є потужним інструментом для створення продуктивного програмного забезпечення з можливістю глибокого контролю над ресурсами. Її сильними сторонами є підтримка як процедурного, так і об’єктно-орієнтованого підходу, доступ до низькорівневих операцій, шаблонне програмування, а також кросплатформеність. Разом із тим, C++ вимагає від розробника глибоких знань, має складний синтаксис, високу ймовірність помилок при роботі з пам’яттю, а

також не завжди очевидну поведінку типів даних. Вивчення цієї мови потребує значних зусиль, особливо з урахуванням різноманіття стандартів і реалізацій.

C# — сучасна, об’єктно-орієнтована мова програмування, розроблена компанією Microsoft. Вона була створена як частина платформи .NET для розробки як десктопних, так і вебзастосунків. На відміну від C++, C# має більш простий та зрозумілий синтаксис, що знижує поріг входження. До її переваг можна віднести автоматичне управління пам’яттю, широку підтримку з боку Microsoft, великий вибір бібліотек, можливість кросплатформної розробки через Xamarin, а також потужний інструментарій у вигляді Visual Studio. Недоліком C# залишається орієнтація на Windows-середовище та залежність від .NET-екосистеми.

З огляду на специфіку задач, які ставилися під час розробки системи управління студентськими науковими гуртками, мова C# виявилася найбільш оптимальним вибором. Вона дозволила швидко реалізувати графічний інтерфейс користувача, організувати роботу з базою даних, реалізувати логіку доступу до ролей та забезпечити підтримку масштабування.

У парі з C# було обрано інтегроване середовище розробки **Microsoft Visual Studio**, яке є одним із найпотужніших засобів для створення застосунків будь-якої складності. Visual Studio забезпечує зручний текстовий редактор з підсвіткою синтаксису, автодоповненням (IntelliSense), графічним дизайнером форм, вбудованим компілятором, налагоджувачем та інструментами для роботи з базами даних. Ця платформа дозволяє ефективно реалізовувати як логіку програми, так і візуальну частину інтерфейсу, що робить її ідеальним вибором для розробки десктопного застосунку.

Крім того, Visual Studio має гнучку систему розширень, підтримку Git, автоматичну перевірку помилок і можливість створення як класичних .NET-додатків, так і сучасних кросплатформених рішень.

Таким чином, з урахуванням функціональних можливостей, гнучкості, доступності навчальних матеріалів і сумісності з Microsoft SQL Server, **для розробки програмного забезпечення було обрано мову C# та середовище**

Microsoft Visual Studio, які найкраще відповідають вимогам поставленого проєкту.

3.4 Принцип роботи у середовищі візуальної розробки програм

При розробці прикладного програмного забезпечення важливо обрати оптимальне середовище розробки (IDE), яке б максимально відповідало вимогам проєкту. Серед доступних рішень було розглянуто два сучасних інструменти: **JetBrains Rider** та **Microsoft Visual Studio**.

JetBrains Rider — це потужне кросплатформне IDE, створене компанією JetBrains, яке підтримує розробку .NET-додатків, включно з C#, ASP.NET, Xamarin та Unity. IDE базується на рушії ReSharper і забезпечує глибокий аналіз коду, автоматичні підказки, рефакторинг і зручну навігацію. Rider підтримує розробку на Windows, macOS і Linux, що робить його гнучким вибором для команд із різними операційними системами. Однак, незважаючи на свої функціональні можливості, середовище Rider є платним (лише з пробним періодом) і має дещо вищі вимоги до ресурсів системи.

Натомість **Microsoft Visual Studio** — це офіційне середовище розробки від компанії Microsoft, яке активно підтримується та є провідним інструментом для створення додатків на платформі .NET. Воно надає широкий спектр можливостей: розробку інтерфейсу, редагування коду, відлагодження, тестування та інтеграцію з системами контролю версій. Visual Studio має зручну інфраструктуру для роботи з **Windows Forms** — технологією, що дозволяє швидко створювати візуальні додатки з використанням кнопок, списків, таблиць та інших елементів керування.

Windows Forms є одним з найпопулярніших способів створення графічного інтерфейсу для настільних застосунків у Windows-середовищі. Цей підхід поєднує простоту реалізації з широкими можливостями налаштування взаємодії з користувачем. Середовище Visual Studio забезпечує візуальний редактор форм,

автогенерацію коду подій, інтеграцію з базою даних та ефективне управління проєктами.

У процесі розробки системи управління студентськими науковими гуртками було прийнято рішення використовувати мову програмування **C#** та середовище **Microsoft Visual Studio**, оскільки воно забезпечує максимальну зручність, потужний функціонал і повну сумісність із платформою **Windows Forms** — основною технологією побудови графічного інтерфейсу в даному проєкті.

Висновок до розділу 3

У цьому розділі було розглянуто практичні аспекти розробки інформаційної системи управління студентськими науковими гуртками. Першочергово було обґрунтовано вибір СУБД — **Microsoft SQL Server**, що стала надійною основою для роботи з даними. Завдяки своїм можливостям — підтримці транзакцій, збережених процедур, гнучкій системі ролей і безпеки — вона дозволила реалізувати структуроване, безпечне та масштабоване зберігання інформації про користувачів, гуртки, керівників, події, новини та звіти.

На наступному етапі було реалізовано створення самої бази даних, проєктування таблиць і налаштування доступу відповідно до логіки прав користувачів у системі. SQL-запити були використані для формування структури таблиць, а засоби управління ролями — для організації контрольованого доступу до окремих даних.

Особливу увагу приділено вибору мови програмування та середовища розробки. З урахуванням поставлених вимог — реалізації візуального інтерфейсу, зручного управління ролями та стабільної інтеграції з базою даних — було обрано **мову програмування C#** у поєднанні з середовищем **Microsoft**

Visual Studio. Такий вибір забезпечив ефективну побудову графічного інтерфейсу на основі **Windows Forms**, спрощив інтеграцію з SQL Server, а також надав розширений функціонал для налагодження, тестування та масштабування додатку.

Таким чином, розділ 3 підтвердив технічну доцільність і практичну реалізованість архітектурного підходу до побудови системи. Була сформована чітка трирівнева структура: **рівень збереження даних (SQL Server)**, **рівень обробки (бізнес-логіка на C#)** і **рівень представлення (Windows Forms UI)**. Такий підхід забезпечує розділення відповідальностей між модулями, гнучкість у розширенні функціоналу та стабільність при реальній експлуатації системи.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування є обов’язковим етапом життєвого циклу створення програмного забезпечення, який забезпечує перевірку відповідності реалізованого функціоналу вимогам технічного завдання, а також виявлення помилок, які могли виникнути в процесі розробки. Метою тестування є забезпечення стабільної, безпечної та коректної роботи програмної системи, а також підвищення її загальної якості.

Під час розробки автоматизованої системи управління студентськими науковими гуртками тестування проводилося з акцентом на функціональні, логічні та безпекові аспекти роботи системи. Тестування здійснювалося з використанням елементів як структурного підходу (внутрішня логіка), так і функціонального (взаємодія користувача з інтерфейсом).

У ході тестування перевірялися наступні ключові компоненти:

- **розмежування доступу відповідно до ролей користувачів:** було протестовано доступ до функціоналу в залежності від ролі користувача (директор, керівник гуртка, учасник). Система успішно обмежує доступ до редагування або перегляду певних модулів згідно з заданими правами;
- **механізми автентифікації та авторизації:** проведено перевірку процедури входу в систему, обробки помилкових облікових даних, а також відображення відповідних повідомлень про помилки;
- **функціональність додавання, редагування та видалення даних:** виконувалося тестування роботи з основними сутностями системи (гуртки, користувачі, події, матеріали, новини), зокрема — при коректних та некоректних вхідних даних;

- **збереження даних і резервне копіювання:** оцінено роботу збереження змін у базі даних, а також тестування сценаріїв з автоматичним та ручним резервуванням інформації;
- **завершення сеансу роботи користувача:** перевірена стабільність виходу з програми та запобігання повторному доступу без повторної авторизації.

У процесі тестування були використані два підходи:

- **тестування за методом "білої скриньки"** — передбачало перевірку внутрішньої логіки обробки даних, правильності побудови запитів до бази даних, обробки виняткових ситуацій;
- **тестування за методом "чорної скриньки"** — проводилося без аналізу внутрішньої структури коду, шляхом взаємодії з інтерфейсом користувача і перевірки поведінки системи в різних ситуаціях.

Крім основного функціонального тестування, були виконані наступні види додаткових перевірок:

- **тестування зручності інтерфейсу (юзабіліті):** перевірялася логічність розміщення елементів керування, зручність навігації, наявність підказок та повідомлень про помилки;
- **обмежене тестування навантаження:** система перевірялась на коректну роботу при значному обсязі даних у базі, з метою визначення стабільності виконання запитів;
- **тестування безпеки:** моделювались ситуації спроб доступу до заборонених дій, тестувався механізм захисту паролів та блокування дій користувачів з обмеженими правами.

За результатами проведеного тестування можна зробити висновок, що програмне забезпечення функціонує відповідно до поставлених вимог. Основні модулі системи працюють стабільно, помилки, виявлені в процесі тестування,

були своєчасно усунуті. Система успішно пройшла перевірку на функціональність, надійність та зручність використання.

Таким чином, проведене тестування підтверджує готовність системи до подальшого впровадження та практичного застосування.

4.2 Вимоги до апаратного та програмного забезпечення

Конфігурація обчислювальної системи, необхідної для функціонування програмного забезпечення, включає **апаратну** та **програмну** складові. Всі сучасні комп'ютерні системи мають модульну (блокову) структуру, яка забезпечує можливість масштабування та обслуговування. Взаємодія між окремими компонентами забезпечується апаратними інтерфейсами, а передача даних здійснюється через стандартизовані мережеві порти.

Для забезпечення коректної роботи автоматизованої системи управління студентськими науковими гуртками передбачено **серверно-клієнтську архітектуру**. На сервері розміщується база даних Microsoft SQL Server, до якої підключаються клієнтські машини, що запускають десктопний застосунок, розроблений у середовищі Microsoft Visual Studio на мові C# з використанням Windows Forms.

У таблиці 4.1 подано апаратні вимоги до сервера для коректної роботи системи.

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

У таблиці 4.2 наведено програмні вимоги до сервера, необхідні для встановлення та стабільної роботи системи.

Таблиця 4.2

Програмні вимоги для сервера

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

Необхідне програмне забезпечення

1. **Microsoft .NET Framework 4.7 або вище** — для запуску програми, розробленої на C#.
2. **Microsoft Visual C++ Redistributable** — для підтримки бібліотек середовища розробки.

3. **Microsoft Excel 2010 або новіший** — для експорту та друку звітів у табличному вигляді.
4. **Антивірусне ПЗ** з виключенням робочого каталогу — для захисту без впливу на продуктивність системи.
5. **Клієнт Microsoft SQL Server Management Studio (SSMS)** — для адміністрування бази даних (на сервері або у системного адміністратора).

4.3 Опис роботи програми

На початку роботи з інформаційною системою управління студентськими науковими гуртками користувач спершу потрапляє до вікна завантаження програми (рис. 4.1), після чого система автоматично переходить до форми авторизації (рис. 4.2). У вікні авторизації необхідно ввести ім’я користувача (логін) та пароль згідно з індивідуально наданими правами доступу.

Вхід до системи реалізовано з урахуванням ролей — **директор**, **керівник гуртка** та **учасник гуртка**. Кожна з ролей має власні облікові дані та різний рівень доступу до функціоналу. Наприклад, директор має можливість формувати звіти, керівник — додавати новини, події та керувати складом гуртка, а учасник — переглядати інформаційні матеріали, події та новини.

Після авторизації інтерфейс програми адаптується до ролі користувача: змінюється набір доступних розділів і дій, відповідно до прав, наданих у системі. Подальша взаємодія з функціоналом інформаційної системи — від керування гуртками до перегляду подій — представлена на рисунках 4.3 – 4.22.

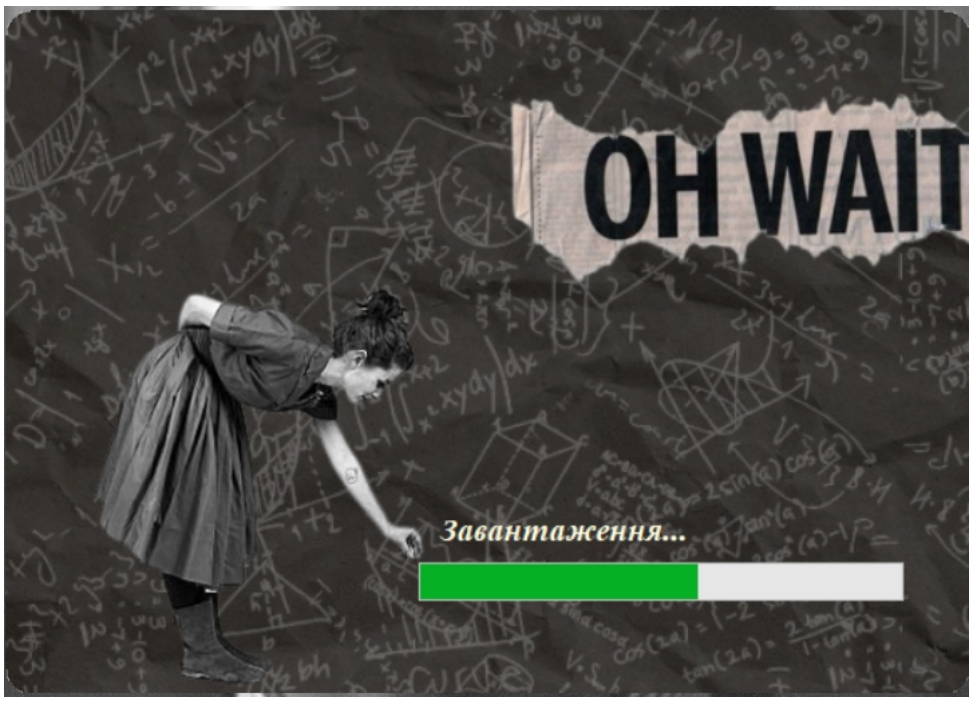


Рис. 4.1 Вікно завантаження програми

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

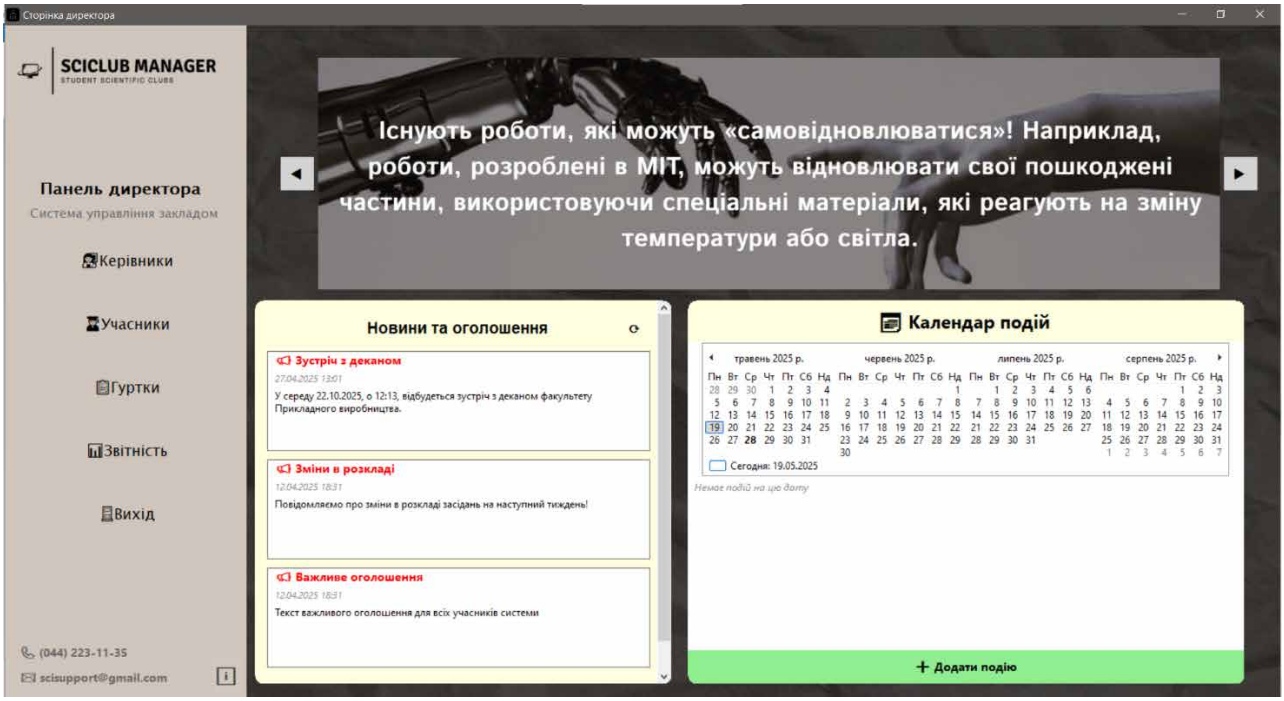


Рис. 4.3 Інтерфейс головного меню для ролі «Директор»

Рис. 4.4 Вікно додавання нового керівника до системи

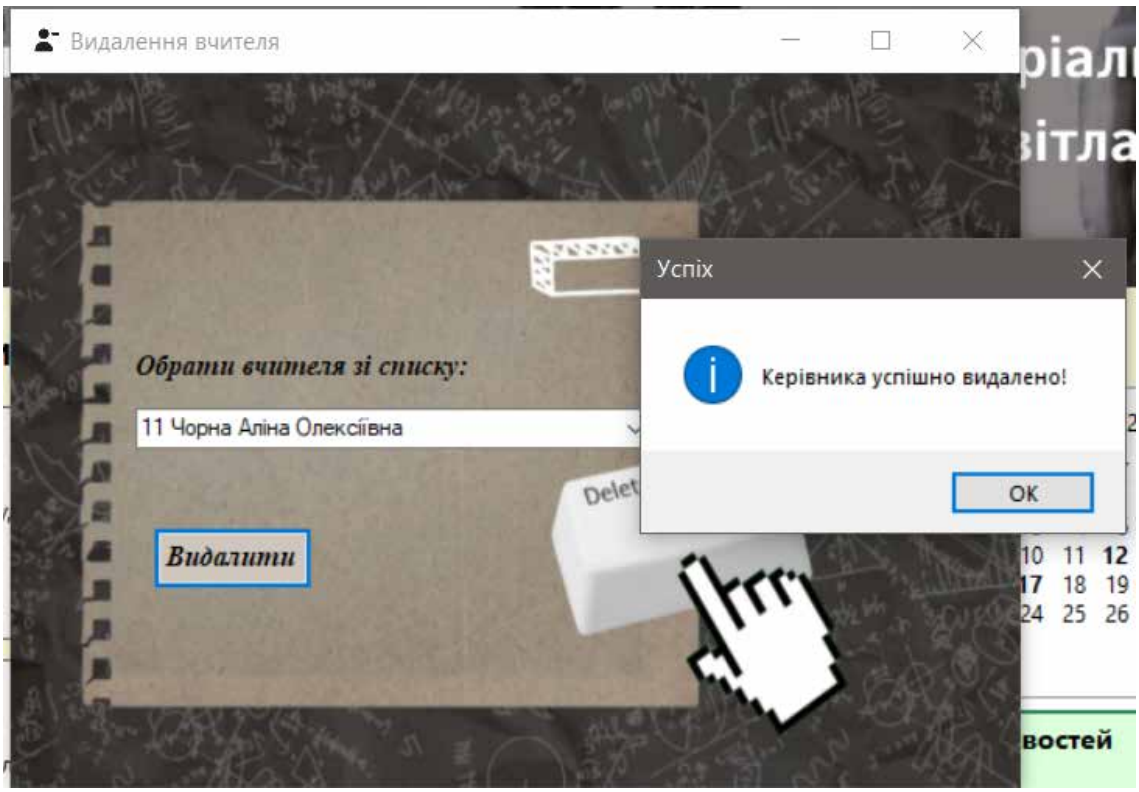


Рис. 4.5 Вікно видалення керівника з системи

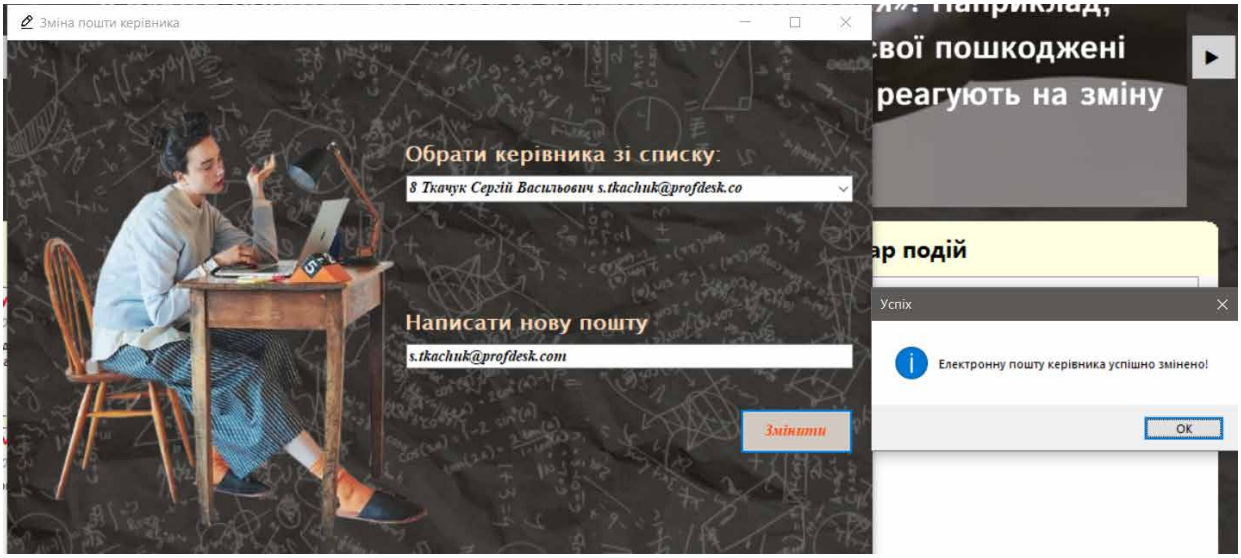


Рис. 4.6 Вікно редагування електронної пошти керівника

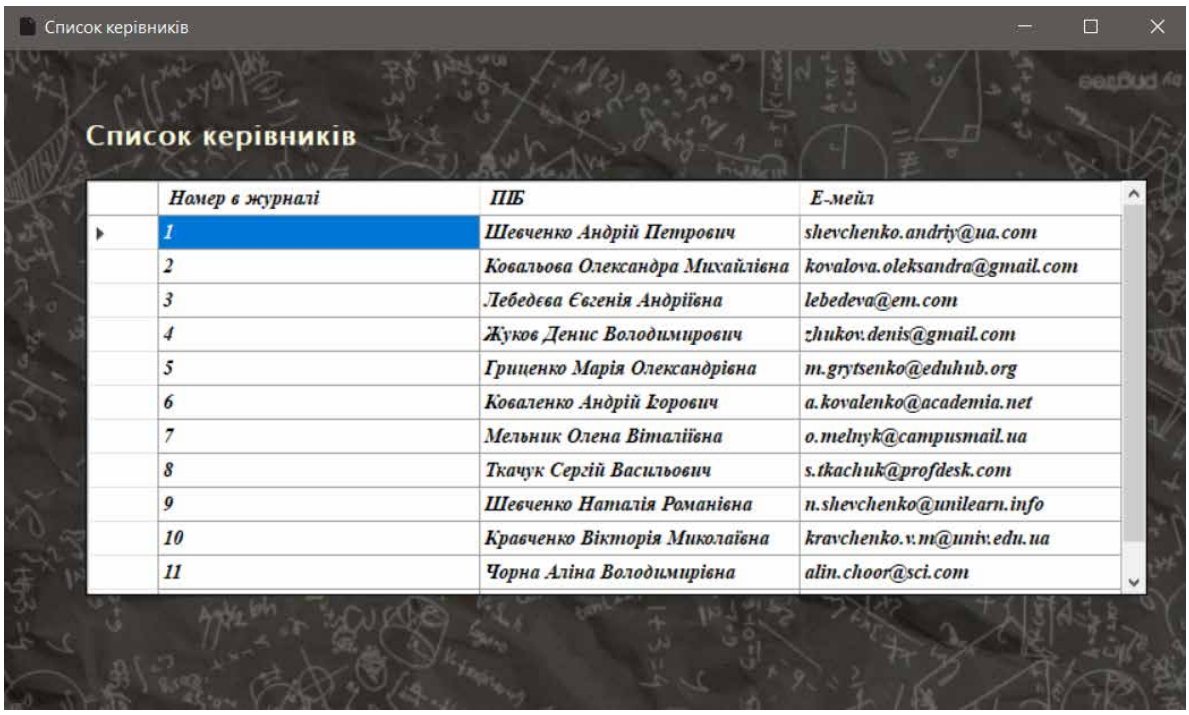


Рис. 4.7 Вікно відображення списку керівників

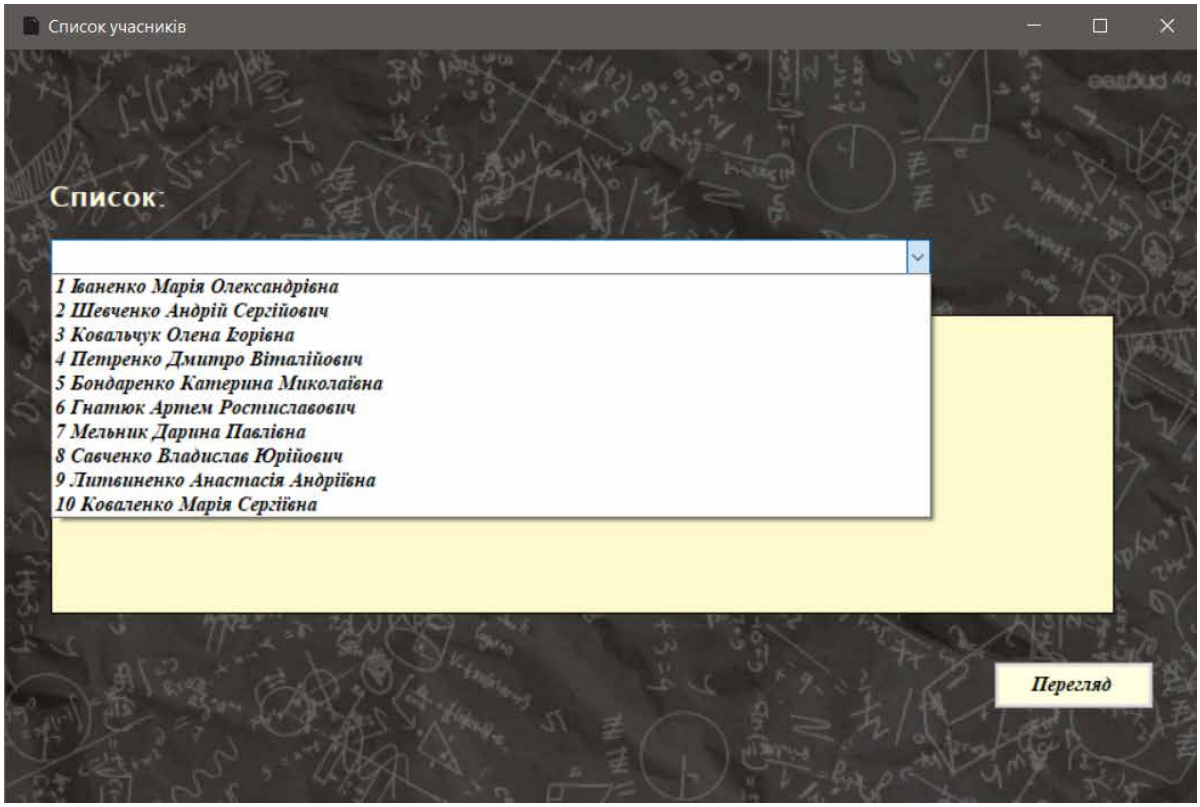


Рис. 4.8 Вікно вибору учасника зі списку

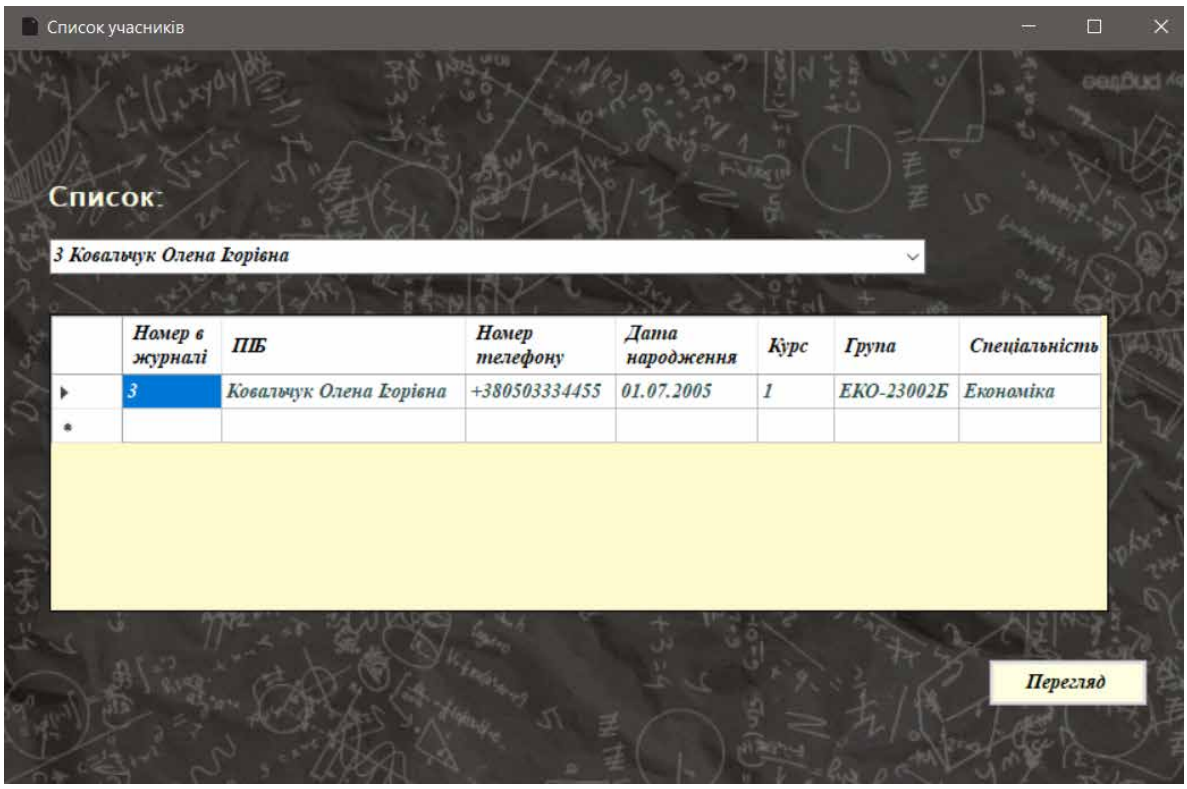


Рис. 4.9 Перегляд інформації про учасника

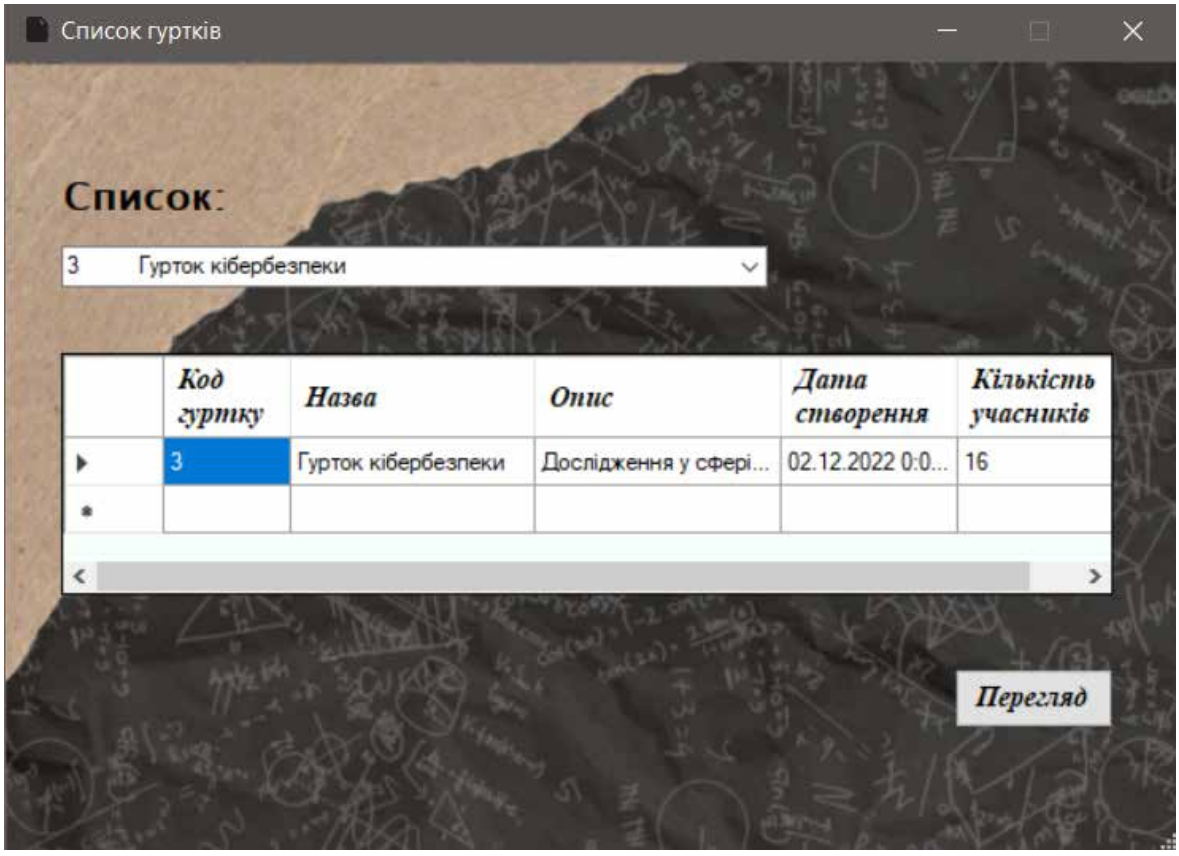


Рис. 4.10 Перегляд списку гуртків

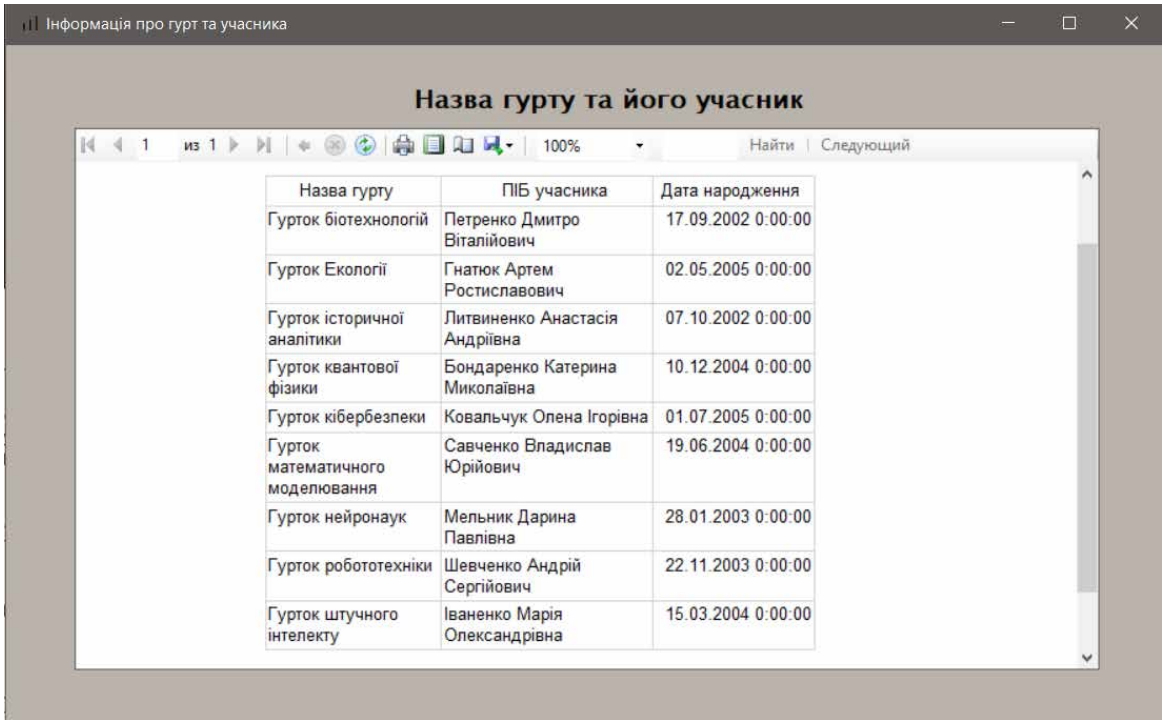


Рис. 4.11 Сформований звіт «Інформація про гурт та його учасника»

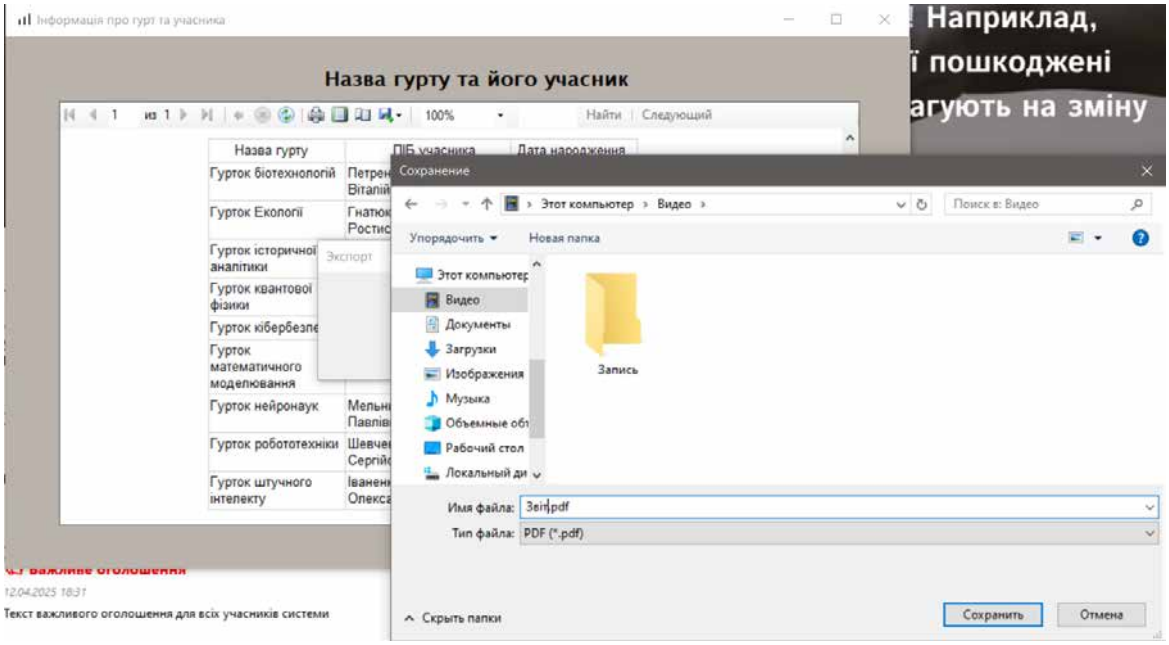
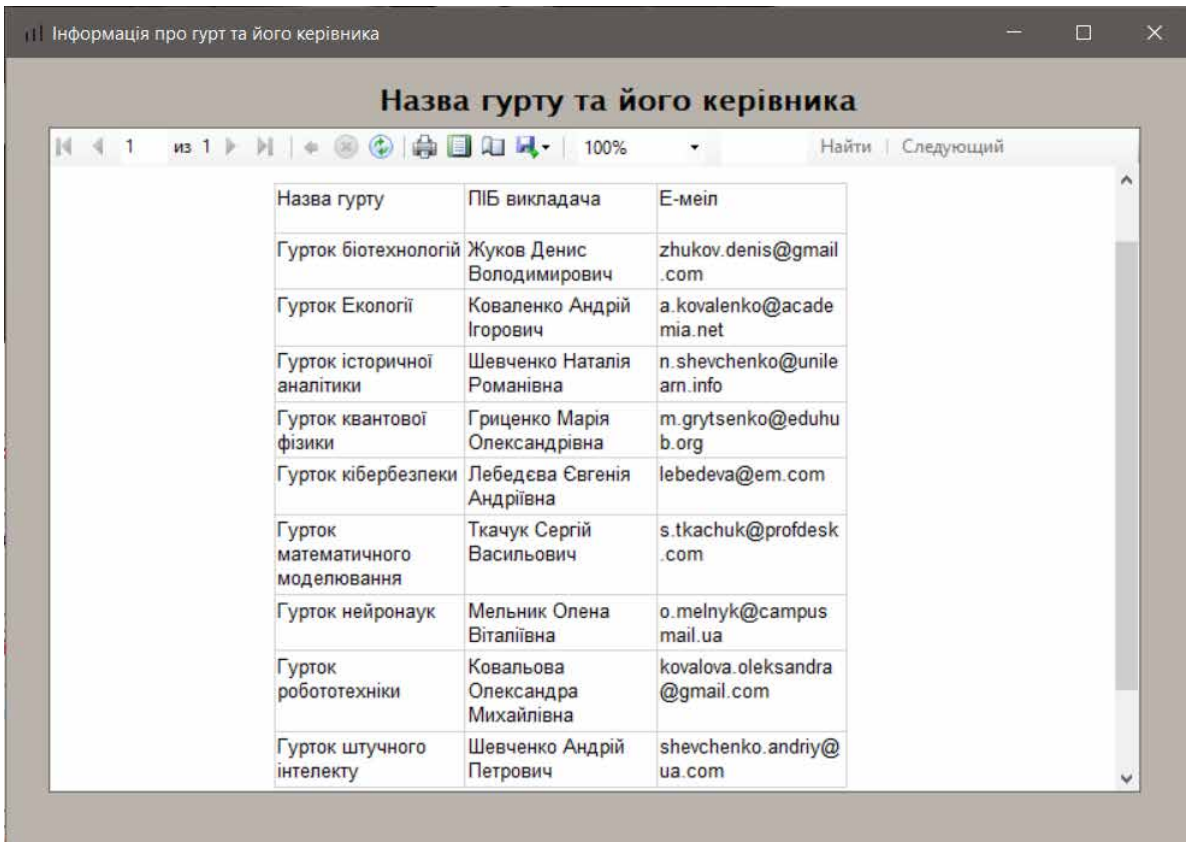
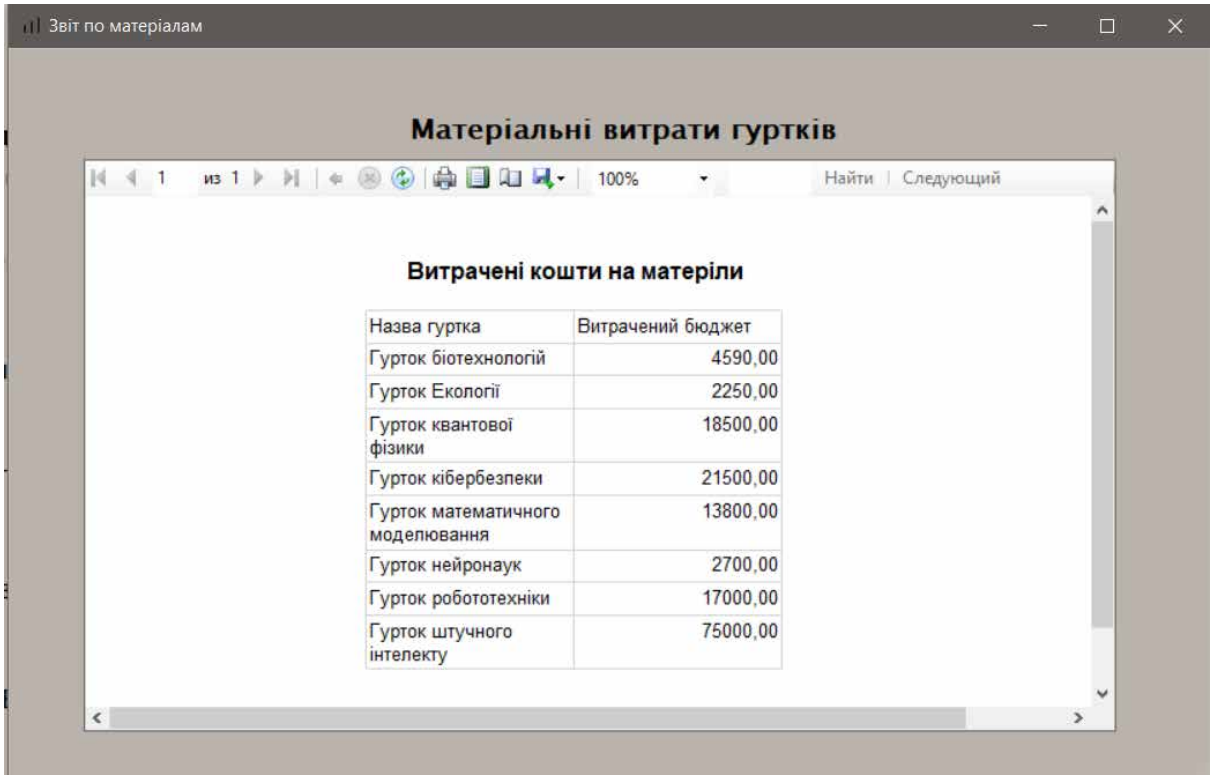


Рис. 4.12 Вікно збереження звіту



Назва гурту	ПІБ викладача	E-меіл
Гурток біотехнологій	Жуков Денис Володимирович	zhukov.denis@gmail.com
Гурток Екології	Коваленко Андрій Ігорович	a.kovalenko@academia.net
Гурток історичної аналітики	Шевченко Наталія Романівна	n.shevchenko@unilearn.info
Гурток квантової фізики	Гриценко Марія Олександрівна	m.grytsenko@eduhub.org
Гурток кібербезпеки	Лебедєва Євгенія Андріївна	lebedeva@em.com
Гурток математичного моделювання	Ткачук Сергій Васильович	s.tkachuk@profdesk.com
Гурток нейронаук	Мельник Олена Віталіївна	o.melnyk@campusmail.ua
Гурток робототехніки	Ковальова Олександра Михайлівна	kovalova.oleksandra@gmail.com
Гурток штучного інтелекту	Шевченко Андрій Петрович	shevchenko.andriy@ua.com

Рис. 4.13 Сформований звіт «Інформація про гурт та його керівника»



Назва гуртка	Витрачений бюджет
Гурток біотехнологій	4590,00
Гурток Екології	2250,00
Гурток квантової фізики	18500,00
Гурток кібербезпеки	21500,00
Гурток математичного моделювання	13800,00
Гурток нейронаук	2700,00
Гурток робототехніки	17000,00
Гурток штучного інтелекту	75000,00

Рис. 4.14 Сформований звіт «Матеріальні витрати гуртків»

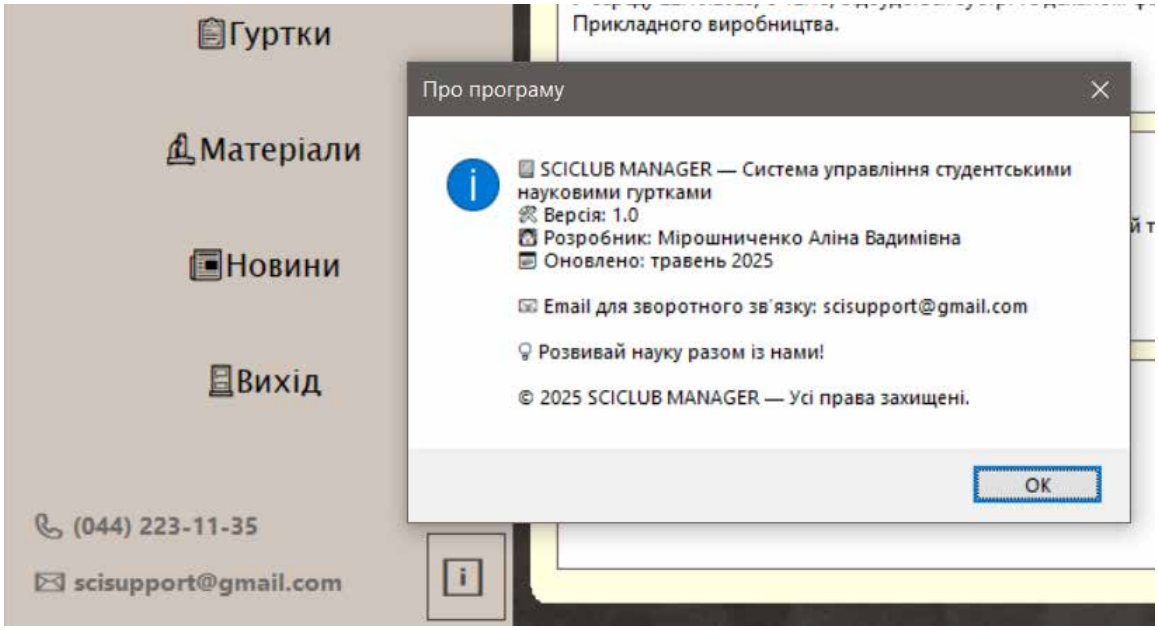


Рис. 4.15 Вікно з короткою інформацією про програму та технічну підтримку

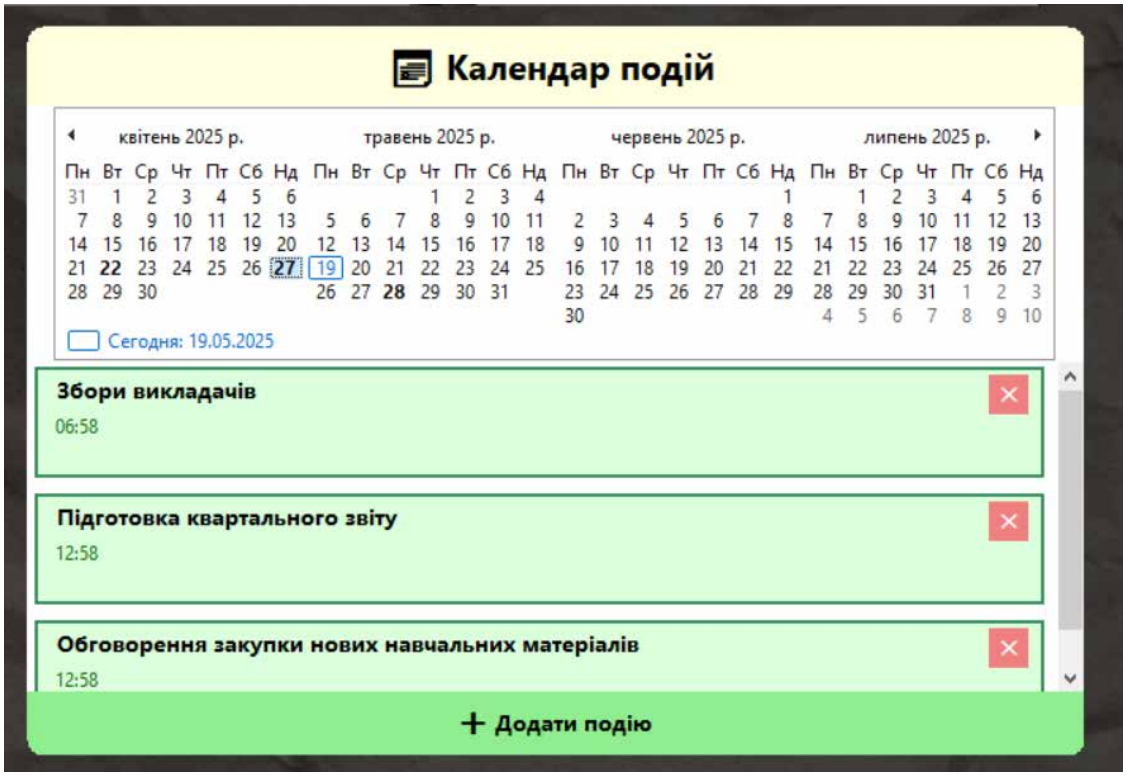


Рис. 4.16 Календар подій в системі

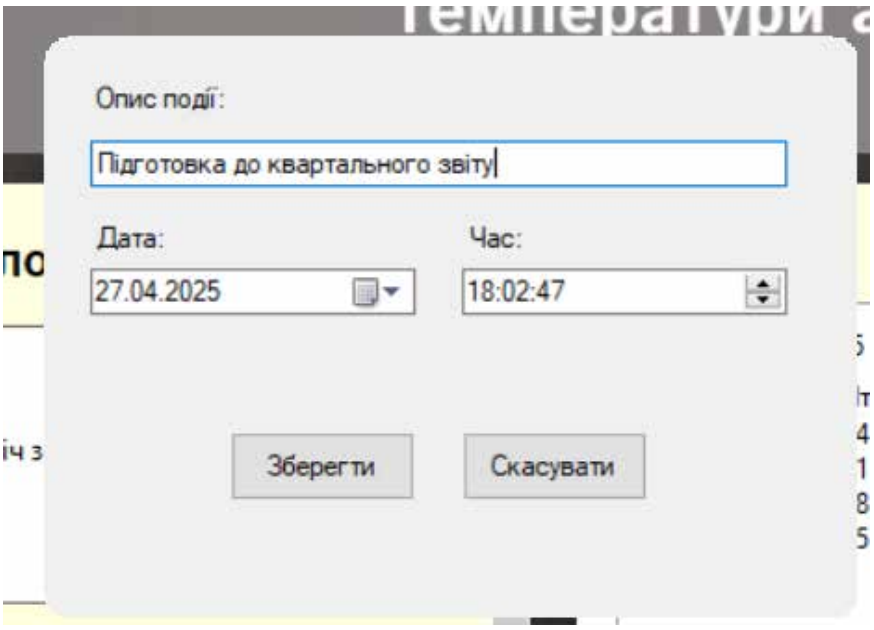


Рис. 4.17 Вікно додавання подій до календаря

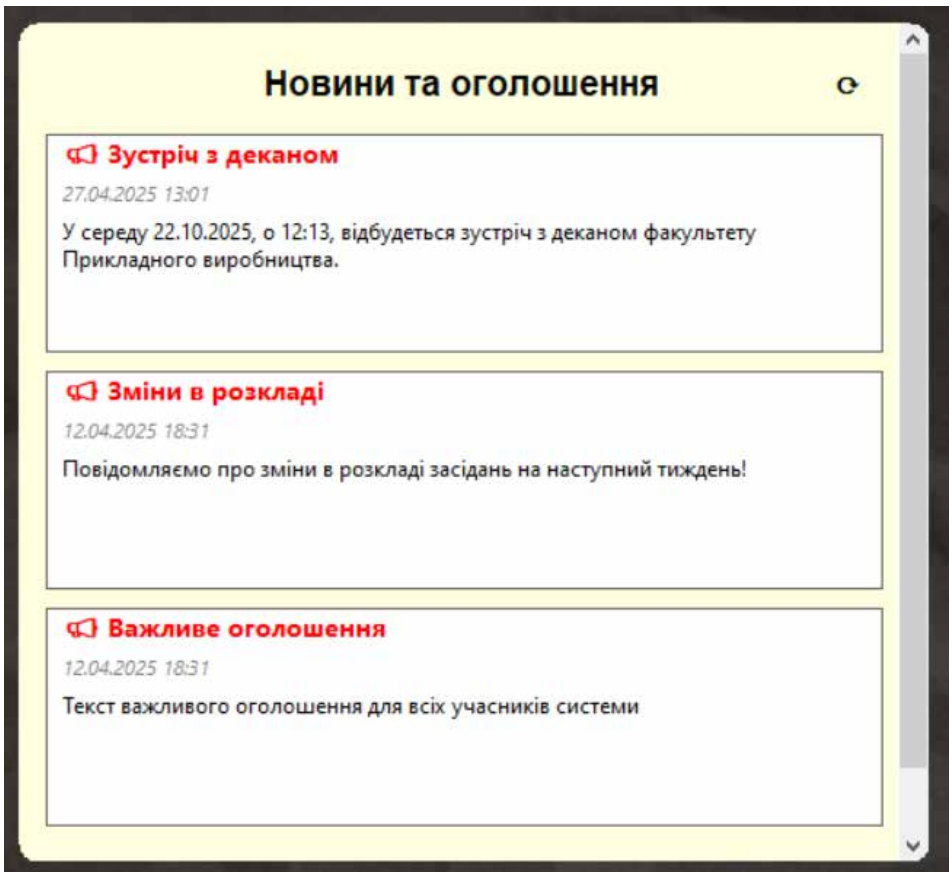


Рис. 4.18 Вікно з новинами та оголошеннями

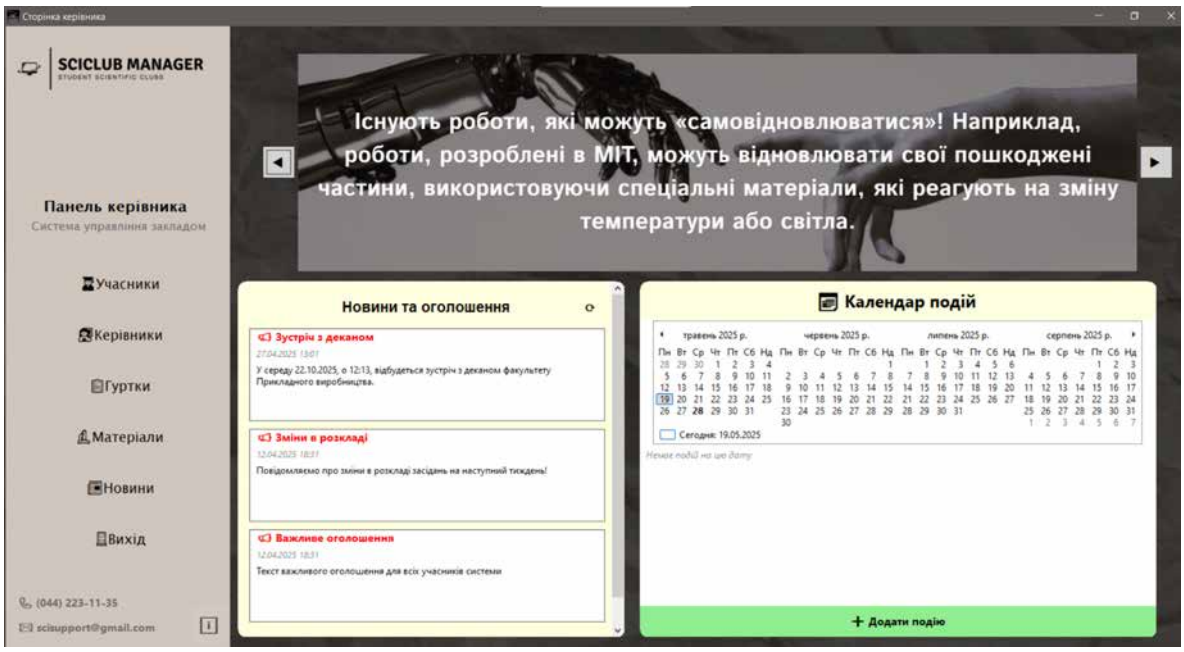


Рис. 4.19 Інтерфейс головного меню для ролі «Керівник»

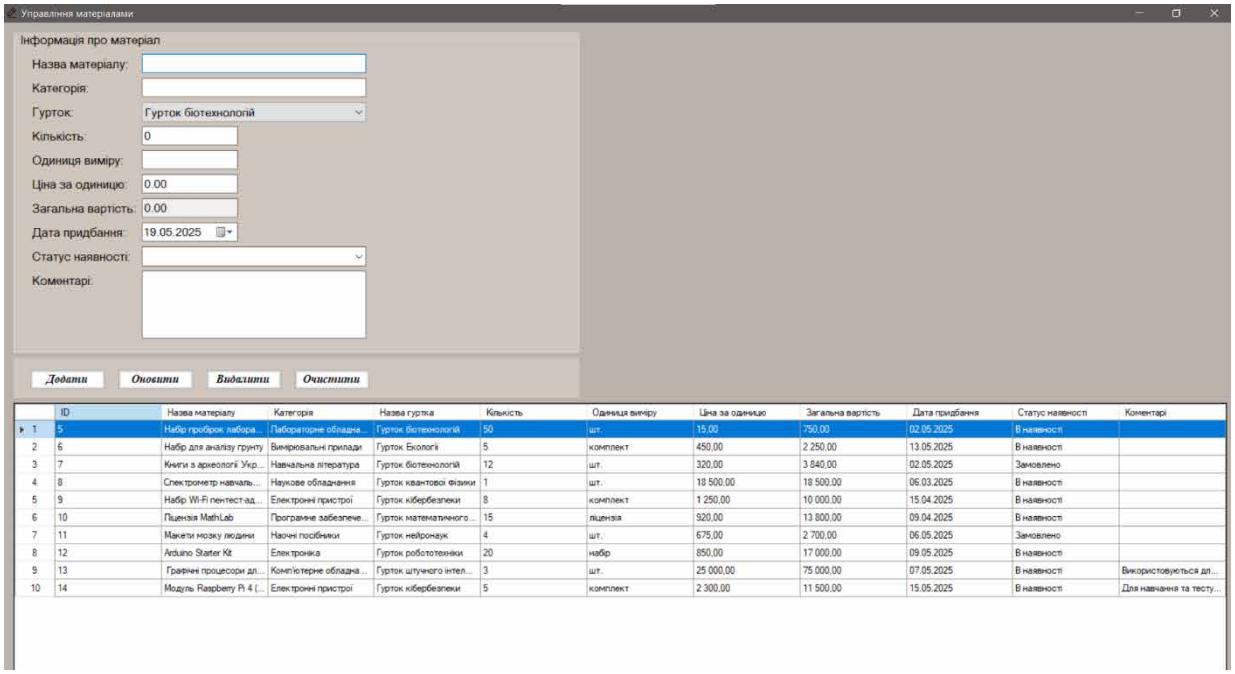


Рис. 4.20 Вікно управління матеріалами

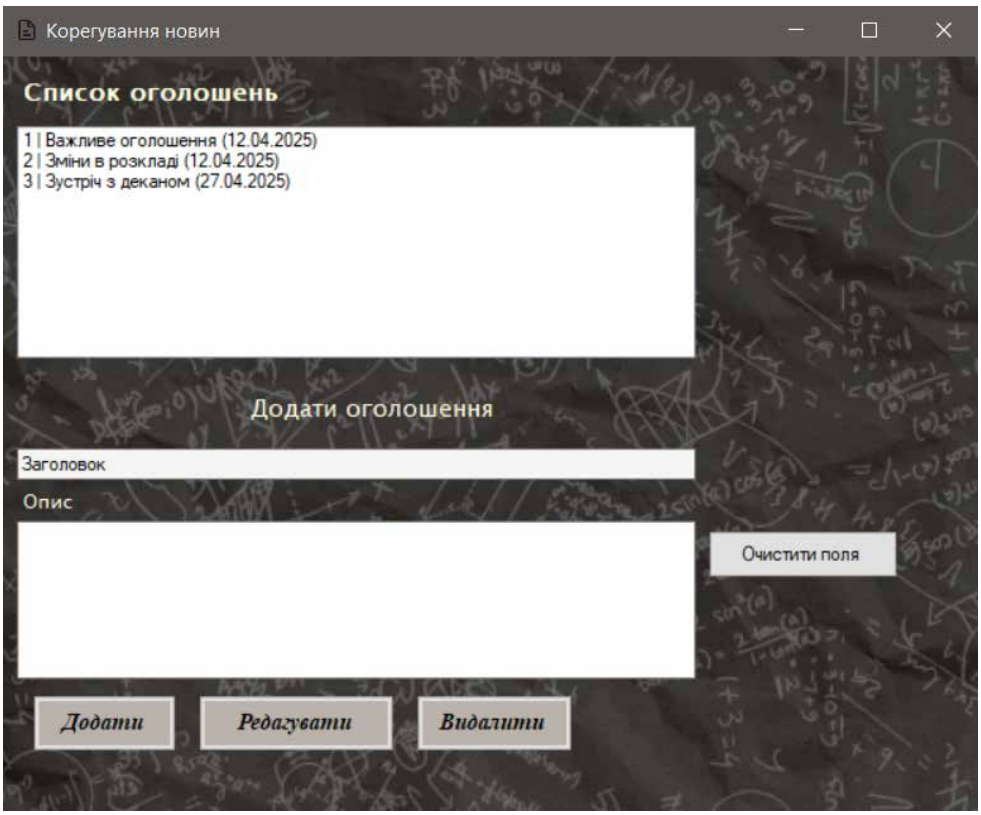


Рис. 4.21 Вікно «Корегування новин»

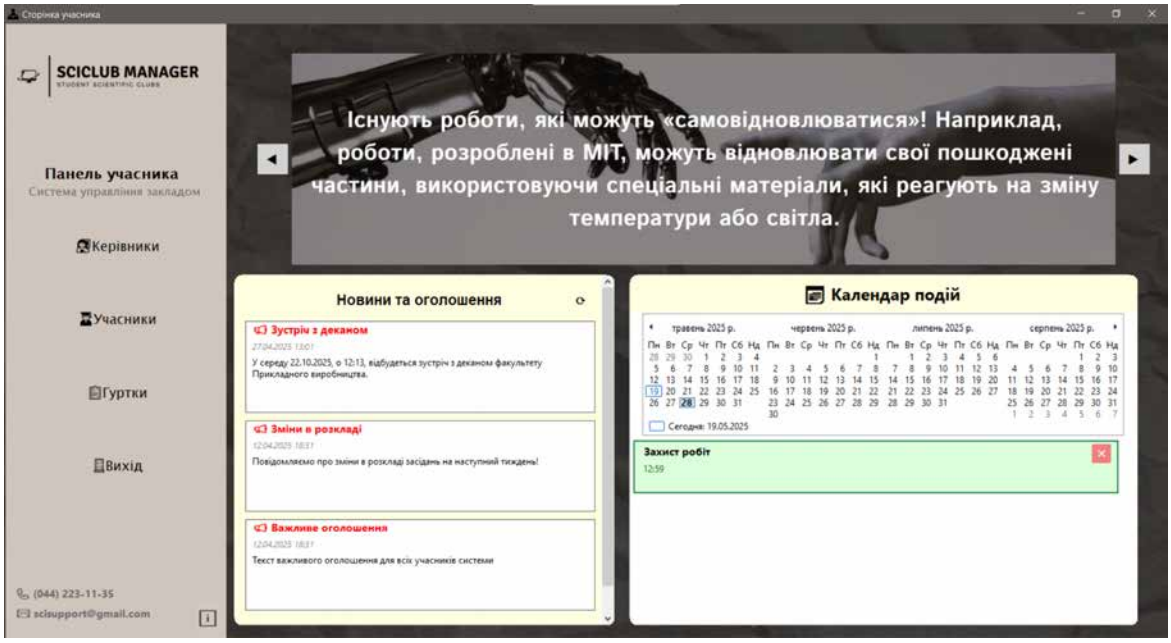


Рис. 4.22 Інтерфейс головного меню для ролі «Учасник»

Висновки до розділу 4

У четвертому розділі було висвітлено рекомендації щодо впровадження та експлуатації автоматизованої інформаційної системи управління студентськими науковими гуртками. Ретельне тестування програмного забезпечення дозволило виявити й усунути окремі недоліки, а також перевірити відповідність функціональних модулів технічному завданню. Особлива увага була приділена перевірці стабільності роботи, дотриманню логіки розмежування доступу, коректності взаємодії з базою даних і стійкості інтерфейсу до навантаження.

Функціональні можливості системи були перевірені з позицій трьох основних ролей користувачів — директора, керівника гуртка та учасника. Інтерфейс адаптується до ролі, відкриваючи лише дозволені розділи, що свідчить про належну реалізацію механізму авторизації та прав доступу. Було також забезпечено захист персональних даних, підтримку резервного копіювання та передбачено відновлення в разі збоїв.

Додатково описано особливості використання програми — від завантаження та авторизації до роботи з ключовими модулями, включаючи керування користувачами, створення подій, новин, формування звітів тощо. Окремі приклади роботи інтерфейсу супроводжуються ілюстраціями, що демонструють зручність користування та зрозумілу структуру програми.

У межах розділу також наведено апаратні та програмні вимоги до серверної й клієнтської частин системи. Запропонована конфігурація дозволяє забезпечити сумісність із сучасними платформами, підтримку необхідного рівня продуктивності, а також потенціал для масштабування в умовах зростання кількості користувачів або розширення функціоналу.

Таким чином, розроблена система повністю відповідає поставленим вимогам, успішно пройшла перевірку в умовах моделювання реального використання та готова до впровадження в освітнє середовище з метою оптимізації управління науковими гуртками, покращення внутрішньої комунікації та зручного адміністрування діяльності.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було розроблено прикладне програмне забезпечення, яке реалізує функціональність автоматизованої системи управління студентськими науковими гуртками. Робота охопила всі етапи повного циклу розробки — від аналізу предметної області та постановки завдання до створення інтерфейсу, реалізації логіки, побудови бази даних і тестування.

Система орієнтована на автоматизацію основних процесів, що відбуваються в межах роботи гуртків — облік учасників і керівників, ведення матеріалів, додавання подій, створення новин та формування звітності. Важливу увагу приділено диференціації прав доступу: кожна роль (директор, керівник

гуртка, учасник) має власний набір функцій, що гарантує безпечність і структурованість роботи в системі.

Для розробки програмного забезпечення було використано середовище **Microsoft Visual Studio** з мовою програмування **C#** та технологією **Windows Forms**, що дозволило швидко створити графічно зручний інтерфейс. У якості інструменту для зберігання та обробки даних було обрано **Microsoft SQL Server**, який забезпечив ефективну роботу з табличними структурами та підтримку складної логіки запитів.

Під час реалізації програмної частини було створено зручну базу даних, передбачено механізми авторизації, обробки подій, керування користувачами, а також реалізовано модуль створення та збереження звітів. Особливістю системи є простота у використанні, інтуїтивна навігація, а також можливість швидкого масштабування у випадку розширення функціоналу або кількості користувачів.

Важливою частиною проєкту стало тестування розробленого ПЗ, яке підтвердило коректність роботи всіх основних функцій, відповідність вимогам технічного завдання, стійкість до помилкових даних та дотримання логіки розмежування прав доступу.

Таким чином, розроблена інформаційна система здатна значно полегшити облік та організацію діяльності наукових гуртків у навчальних закладах, зменшити кількість ручної роботи, підвищити точність зберігання даних та забезпечити прозору звітність. У майбутньому система може бути доповнена мобільним інтерфейсом або веб-версією, що зробить її ще доступнішою для ширшого кола користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бочаров В. А., Рибак С. О. *Системи управління базами даних: навч. посіб.* — Київ: Ліра-К, 2021. — 304 с.
2. Еріксон Г. *Проєктування баз даних. ER-моделювання.* — Харків: Фоліо, 2019. — 216 с.
3. Фаулер М. *UML. Основи. Стандартний навчальний курс.* — Київ: Діалектика, 2020. — 272 с.
4. Хосейні Е. *Об'єктно-орієнтоване програмування з використанням C#.* — Львів: Новий Світ, 2022. — 384 с.

5. Глушков В. М. *Основи програмної інженерії: навч. посіб.* — Київ: НаУКМА, 2020. — 248 с.
6. Коберник О. М., Мішустін Д. Ю. *Інформаційні системи та технології: підручник*. — Харків: ХНУ, 2019. — 356 с.
7. Microsoft Docs. *SQL Server documentation* [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/en-us/sql/sql-server
8. Microsoft Docs. *C# Guide* [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/
9. Visual Studio. *Інтегроване середовище розробки* [Електронний ресурс]. — Режим доступу: https://visualstudio.microsoft.com
10. JetBrains. *Rider Documentation* [Електронний ресурс]. — Режим доступу: https://www.jetbrains.com/rider/documentation
11. CA ERwin Data Modeler. *User Guide*. — Broadcom Inc. [Електронний ресурс]. — Режим доступу: https://techdocs.broadcom.com

ДОДАТОК А

База даних

Сторінок 8

Київ – 2025

Код SQL-запитів для формування таблиць бази даних

```
USE Art_clubs
GO
-- Видалення таблиць у правильному порядку
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'News' AND type_desc = 'USER_TABLE')
 DROP TABLE News
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Materials' AND type_desc = 'USER_TABLE')
 DROP TABLE Materials
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Member_of_the_gourt' AND type_desc = 'USER_TABLE')
```

```
 DROP TABLE Member_of_the_gourt
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Material_gourt' AND type_desc = 'USER_TABLE')
 DROP TABLE Material_gourt
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Head_of_the_gourt' AND type_desc = 'USER_TABLE')
 DROP TABLE Head_of_the_gourt
IF EXISTS (SELECT name FROM sys.objects WHERE name = 'Gourt' AND type_desc = 'USER_TABLE')
 DROP TABLE Gourt
GO
-- Створення таблиць
CREATE TABLE Gourt (
    Gourt_code CHAR(10) NOT NULL PRIMARY KEY,
    Name_Gourt VARCHAR(50) NOT NULL UNIQUE,
    Description_Gourt VARCHAR(MAX) NOT NULL,
    Date_of_creation DATE,
    Number_of_participants CHAR(10) NOT NULL
);
CREATE TABLE Member_of_the_gourt (
    ID_member INT NOT NULL PRIMARY KEY,
    First_name_Last_name CHAR(255) NOT NULL UNIQUE,
    Phone_number CHAR(255) NOT NULL UNIQUE
```



```
    Date_of_birth DATE
);
CREATE TABLE Material_gourt (
    Material_code CHAR(10) NOT NULL PRIMARY KEY,
    Description_ VARCHAR(MAX) NOT NULL,
    Author VARCHAR(50) NOT NULL,
    Year_of_creation DATE
);
CREATE TABLE Head_of_the_gourt (
    ID_head INT NOT NULL PRIMARY KEY,
    First_name_Last_name VARCHAR(255) NOT NULL UNIQUE,
    Email VARCHAR(255) NOT NULL UNIQUE
```

```
);
-- Розширення таблиці Member_of_the_gourt
ALTER TABLE Member_of_the_gourt
ADD Course INT,
   Group_name CHAR(255),
   Specialty CHAR(255);
GO
-- Створення таблиці новин
IF NOT EXISTS (SELECT * FROM sys.tables WHERE name = 'News')
BEGIN
   CREATE TABLE News (
      ID_news INT PRIMARY KEY IDENTITY(1,1),
      Title NVARCHAR(200) NOT NULL,
      Content NTEXT NOT NULL,
      Date DATETIME DEFAULT GETDATE(),
      Author_ID INT,
      FOREIGN KEY (Author_ID) REFERENCES Head_of_the_gourt(ID_head)
   );
   INSERT INTO News (Title, Content, Date, Author_ID)
VALUES ('Важливе оголошення', 'Текст важливого оголошення для всіх учасників системи',
GETDATE(), 1);
   INSERT INTO News (Title, Content, Date, Author_ID)
VALUES ('Зміни в розкладі', 'Повідомляємо про зміни в розкладі засідань на наступний
тиждень', GETDATE(), 1);
END                                                                    Сторінка 2
GO
-- Таблиця матеріалів для обліку
CREATE TABLE Materials (
   ID INT PRIMARY KEY IDENTITY(1,1),
   Material_Name NVARCHAR(255) NOT NULL,
   Category NVARCHAR(100) NOT NULL,
   Gourt_code CHAR(10) NOT NULL,
   Gourt_Name NVARCHAR(255) NOT NULL,
   Quantity INT NOT NULL,
   Unit NVARCHAR(50) NOT NULL,
   Price_Per_Unit DECIMAL(10, 2) NOT NULL,
   Total_Cost DECIMAL(10, 2) NOT NULL,
   Purchase_Date DATE NOT NULL,
```

```
    Availability_Status NVARCHAR(50) NOT NULL,
    Comments NVARCHAR(500),
CONSTRAINT FK_Materials_Gourt FOREIGN KEY (Gourt_code) REFERENCES Gourt(Gourt_code)
);
```


Код SQL-запитів для створення ролей і надання прав в базі даних

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

```
END
IF NOT EXISTS (SELECT name FROM sys.server_principals WHERE name = 'HeadUsername')
BEGIN
   CREATE LOGIN [HeadUsername] WITH PASSWORD = 'HeadPassword';
END
ELSE
BEGIN
   PRINT 'Логін [HeadUsername] уже існує.';
END
IF NOT EXISTS (SELECT name FROM sys.server_principals WHERE name =
'MemberUsername')
BEGIN
   CREATE LOGIN [MemberUsername] WITH PASSWORD = 'MemberPassword';
END
ELSE
BEGIN
   PRINT 'Логін [MemberUsername] уже існує.';
END
-- Створення користувачів
IF NOT EXISTS (SELECT name FROM sys.database_principals WHERE name =
'DirectorUsername' AND type = 'U')
BEGIN
   CREATE USER [DirectorUsername] FOR LOGIN [DirectorUsername];
END
ELSE
BEGIN
   PRINT 'Користувач [DirectorUsername] уже існує.';
END
IF NOT EXISTS (SELECT name FROM sys.database_principals WHERE name = 'HeadUsername'
AND type = 'U')
BEGIN
   CREATE USER [HeadUsername] FOR LOGIN [HeadUsername];
END
ELSE
BEGIN
   PRINT 'Користувач [HeadUsername] уже існує.';
END
```

```
IF NOT EXISTS (SELECT name FROM sys.database_principals WHERE name =
'MemberUsername' AND type = 'U')
BEGIN
   CREATE USER [MemberUsername] FOR LOGIN [MemberUsername];
END
ELSE
BEGIN
   PRINT 'Користувач [MemberUsername] 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 = 'Head_gourt' AND
type = 'R')
BEGIN
   ALTER ROLE [Head_gourt] ADD MEMBER [HeadUsername];
END
ELSE
BEGIN
   PRINT 'Роль [Head_gourt] не існує.';
END
IF EXISTS (SELECT name FROM sys.database_principals WHERE name =
'Member_of_the_gourt' AND type = 'R')
BEGIN
   ALTER ROLE [Member_of_the_gourt] ADD MEMBER [MemberUsername];
END
ELSE
BEGIN
   PRINT 'Роль [Member_of_the_gourt] не існує.';
END
USE Art_clubs;
```

```
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, INSERT, UPDATE, DELETE ON dbo.Gourt TO [Director];
   GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Member_of_the_gourt TO [Director];
   GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Material_gourt TO [Director];
   GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Head_of_the_gourt TO [Director];
   GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Classes TO [Director];
   GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Visiting TO [Director];
END
ELSE
BEGIN
   -- Якщо роль вже існує, оновлюємо її дозволи
        GRANT SELECT ON dbo.Users TO [Director];
  GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Gourt TO [Director];
   GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Member_of_the_gourt TO [Director];
   GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Material_gourt TO [Director];
     GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Head_of_the_gourt TO [Director];
   GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Classes TO [Director];
   GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Visiting TO [Director];
END
IF NOT EXISTS (SELECT name FROM sys.database_principals WHERE name = 'Head_gourt' AND type
= 'R')                                                                    Сторінка 6
BEGIN
   CREATE ROLE [Head_gourt];
        GRANT SELECT ON dbo.Users TO [Head_gourt];
        GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Visiting TO [Head_gourt];
        GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Material_gourt TO [Head_gourt];
        GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Member_of_the_gourt TO [Head_gourt];
   GRANT SELECT ON dbo.Gourt TO [Head_gourt];
   GRANT SELECT ON dbo.Head_of_the_gourt TO [Head_gourt];
END
ELSE
BEGIN
   -- Якщо роль вже існує, оновлюємо її дозволи
```

```
  GRANT SELECT ON dbo.Users TO [Head_gourt];
        GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Visiting TO [Head_gourt];
        GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Material_gourt TO [Head_gourt];
        GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Member_of_the_gourt TO [Head_gourt];
   GRANT SELECT ON dbo.Gourt TO [Head_gourt];
   GRANT SELECT ON dbo.Head_of_the_gourt TO [Head_gourt];
END
IF NOT EXISTS (SELECT name FROM sys.database_principals WHERE name = 'Member_of_the_gourt'
AND type = 'R')
BEGIN
   CREATE ROLE [Member_of_the_gourt];
        GRANT SELECT ON dbo.Users TO [Member_of_the_gourt];
   GRANT SELECT ON dbo.Gourt TO [Member_of_the_gourt];
   GRANT SELECT ON dbo.Material_gourt TO [Member_of_the_gourt];
   GRANT SELECT ON dbo.Head_of_the_gourt TO [Member_of_the_gourt];
        GRANT SELECT ON dbo.Classes TO [Member_of_the_gourt];
END
ELSE
BEGIN
        GRANT SELECT ON dbo.Users TO [Member_of_the_gourt];
   GRANT SELECT ON dbo.Gourt TO [Member_of_the_gourt];
   GRANT SELECT ON dbo.Material_gourt TO [Member_of_the_gourt];
   GRANT SELECT ON dbo.Head_of_the_gourt TO [Member_of_the_gourt];
        GRANT SELECT ON dbo.Classes TO [Member_of_the_gourt];
END
GO
```


Код SQL-запитів для формування запитів, що слугують основою звітності

```
USE Art_clubs;
-- Уявлення з інформацією про гурти та їх учасників
CREATE VIEW GourtMembers AS
SELECT
   G.Name_Gourt,
   M.First_name_Last_name,
   M.Date_of_birth
FROM
   Gourt G
JOIN
   Member_of_the_gourt M ON G.Gourt_code = M.ID_member;
```

```
GO
-- Уявлення з інформацією про гурти та їх голів
CREATE VIEW GourtHead AS
SELECT
   G.Name_Gourt,
   H.First_name_Last_name,
   H.Email
FROM
   Gourt G
JOIN
   Head_of_the_gourt H ON G.Gourt_code = H.ID_head;
GO
-- Уявлення для розрахунку загальних витрат по гуртках
CREATE VIEW Gourt_Expenses AS
SELECT
   G.Name_Gourt,
   SUM(M.Total_Cost) AS Total_Expenses
FROM
   Materials M
JOIN
   Gourt G ON M.Gourt_code = G.Gourt_code
GROUP BY
   G.Name_Gourt;
GO
```


ДОДАТОК Б

Код програми

Сторінок 16

Київ – 2025

Menu.cs

```
public partial class Menu : Form
{
    private Button btnClose; //  кастомний хрестик
    public Menu()
    {
        InitializeComponent();
        pass.UseSystemPasswordChar = true;
        this.AcceptButton = button1; // аби при натисканні ентер входило
        InitializeCustomCloseButton(); //  додаємо хрестик
        this.FormBorderStyle = FormBorderStyle.None; //  забираємо рамку
```

```
}
private void InitializeCustomCloseButton()
{
   btnClose = new Button();
   btnClose.Text = "    ";
   btnClose.Font = new Font("Arial", 10, FontStyle.Bold);
   btnClose.Size = new Size(30, 30);
   btnClose.Location = new Point(this.ClientSize.Width - 35, 5);
   btnClose.FlatStyle = FlatStyle.Flat;
   btnClose.FlatAppearance.BorderSize = 0;
   btnClose.BackColor = Color.Transparent;
   btnClose.ForeColor = Color.White;
   btnClose.Cursor = Cursors.Hand;
   btnClose.Anchor = AnchorStyles.Top | AnchorStyles.Right;
   btnClose.Click += (s, e) => this.Close();
   this.Controls.Add(btnClose);
   btnClose.BringToFront();
}
protected override void OnResize(EventArgs e)
{
   base.OnResize(e);
   int radius = 30;
   GraphicsPath path = new GraphicsPath();
   path.AddArc(0, 0, radius, radius, 180, 90);
   path.AddArc(this.Width - radius, 0, radius, radius, 270, 90);
```



```
   path.AddArc(this.Width - radius, this.Height - radius, radius, radius, 0, 90);
   path.AddArc(0, this.Height - radius, radius, radius, 90, 90);
   path.CloseAllFigures();
   this.Region = new Region(path);
}
public void SplitLine(string line, char delimiter, ref string first_part, ref string last_part)
{
   int pos = line.IndexOf(delimiter);
   if (pos == -1)
   {
      first_part = "";
      last_part = "";
```

```
        return;
    }
    first_part = line.Substring(0, pos).Trim();
    last_part = line.Substring(pos + 1).Trim();
}
public string ComposeConnectionString(string StringFromWizard, string UserID, string Pass)
{
    string Provider = "", Password = "", Persist_Security_Info = "", User_ID = "",
        Initial_Catalog = "", Data_Source = "";
    string ConnStr, tmp = "", tmp1;
    SplitLine(StringFromWizard, ';', ref Provider, ref tmp);
    if (tmp.IndexOf("Password=") != -1)
    {
        SplitLine(tmp, ';', ref Password, ref tmp);
    }
    SplitLine(tmp, ';', ref Persist_Security_Info, ref tmp);
    SplitLine(tmp, ';', ref User_ID, ref tmp);
    SplitLine(tmp, ';', ref Initial_Catalog, ref Data_Source);
    User_ID = "User ID=" + UserID;
    Password = "Password=" + Pass;
    ConnStr = Provider + ";" + Password + ";" + Persist_Security_Info + ";"
        + User_ID + ";" + Initial_Catalog + ";" + Data_Source;
    return ConnStr;
}
        private void Form1_Load(object sender, EventArgs e) { }
```

Сторінка 2

```
private void button1_Click(object sender, EventArgs e)
{
    string CS = "", csnew;
    if (!Class1.GetConnectionString(ref CS))
    {
        MessageBox.Show("Не знайдений файл з параметрами зв’язку", "Помилка авторизації!",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
    csnew = ComposeConnectionString(CS, login.Text, pass.Text);
    SqlConnection connection = new SqlConnection(csnew);
    try
    {
```

```
            string f = "";
            connection.Open();
SqlCommand command = new SqlCommand($"SELECT Form FROM Users WHERE Username = '{login.Text}'", connection);
            f = command.ExecuteScalar().ToString();
            Type formType = Type.GetType($"BD.{f}");
            Form form = (Form)Activator.CreateInstance(formType, connection);
            form.ShowDialog();
        }
        catch (Exception ex)
        {
MessageBox.Show(ex + "Ім'я користувача або пароль введено невірно!", "Помилка авторизації!",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
    }
    private void label2_Click(object sender, EventArgs e) { }
    private void label1_Click(object sender, EventArgs e) { }
    private void textBox2_TextChanged(object sender, EventArgs e) { }
    private void checkBox1_CheckedChanged(object sender, EventArgs e) { }
    private void label3_Click(object sender, EventArgs e) { }
    private void label3_Click_1(object sender, EventArgs e) { }
    private void checkPass_CheckedChanged(object sender, EventArgs e)
    {
        pass.UseSystemPasswordChar = !checkPass.Checked;
    }
    protected override void WndProc(ref Message m)
    {
        const int WM_NCLBUTTONDBLCLK = 0x00A3;
        if (m.Msg == WM_NCLBUTTONDBLCLK)
            return;
        base.WndProc(ref m);
    }
  }
}
```

Сторінка 3

AddGourt.cs

```
  public partial class AddGourt : Form
```

```
{
    SqlConnection connection;
    public AddGourt(SqlConnection connection)
    {
        InitializeComponent();
       this.connection = connection;
    }
    private void button1_Click(object sender, EventArgs e)
    {
        try
        {
            string value1 = textBox1.Text;
            string value2 = textBox2.Text;
            string value3 = textBox3.Text;
            string value4 = textBox4.Text;
            string value5 = textBox5.Text;
            SqlCommand command = new SqlCommand(
"INSERT INTO Gourt (Gourt_code, Name_Gourt, Description_Gourt, Date_of_creation, Number_of_participants) " +
                "VALUES (@value1, @value2, @value3, @value4, @value5)", connection);
            command.Parameters.AddWithValue("@value1", value1);
            command.Parameters.AddWithValue("@value2", value2);
            command.Parameters.AddWithValue("@value3", value3);
            command.Parameters.AddWithValue("@value4", value4);
            command.Parameters.AddWithValue("@value5", value5);
            int rowsAffected = command.ExecuteNonQuery();
            if (rowsAffected > 0)
            {
MessageBox.Show("Гурток успішно додано!", "Успіх", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            else
            {
MessageBox.Show("Помилка при додаванні гуртка!", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            // Очищення полів
            textBox1.Text = "";
```

```
            textBox2.Text = "";
            textBox3.Text = "";
            textBox4.Text = "";
            textBox5.Text = "";
        }
        catch (Exception ex)
        {
MessageBox.Show("Помилка при додаванні гуртка: " + ex.Message, "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
}
```

DeleteMember.cs

```
  public partial class DeleteMember : Form
  {
    SqlConnection connection;
    public DeleteMember(SqlConnection connection)
    {
      InitializeComponent();
      this.connection = connection;
      FillComboBox();
    }
    private void FillComboBox()
    {
      Try
      {
        string query = "SELECT * FROM Member_of_the_gourt";
        SqlCommand command = new SqlCommand(query, connection);
SqlDataReader reader = command.ExecuteReader(); // виконуємо запит та отримуємо результат у вигляді об'єкту SqlDataReader
        while (reader.Read())
        {
int ID = reader.GetInt32(0); // отримуємо значення стовпця Company_ID з об'єкту SqlDataReader
          string Name = reader.GetString(1);
string Phone = reader.GetString(2);// отримуємо значення стовпця Company_Name з об'єкту SqlDataReader
          DateTime Date = reader.GetDateTime(3);
```

```
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
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
    private void button1_Click(object sender, EventArgs e)
    {
        try
        {
            if (comboBox1.SelectedItem == null)
            {
MessageBox.Show("Будь ласка, оберіть учасника для видалення.", "Попередження", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }
            string selectedString = comboBox1.SelectedItem.ToString();
            string[] substring = selectedString.Split(' ');
            int code = Convert.ToInt32(substring[0].Trim());
```



```
            string query = "DELETE FROM Member_of_the_gourt WHERE ID_member = @code";
            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@code", code);
            int rowsAffected = command.ExecuteNonQuery();
            if (rowsAffected > 0)
            {
MessageBox.Show("Учасника успішно видалено!", "Успіх", MessageBoxButtons.OK, MessageBoxIcon.Information);
                comboBox1.Items.Remove(comboBox1.SelectedItem); // оновлюємо комбобокс
            }
            else
            {
```

```
MessageBox.Show("Учасника не знайдено або не вдалося видалити.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
        catch (Exception ex)
        {
MessageBox.Show("Помилка при видаленні учасника: " + ex.Message, "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
}
```

ListViewMember.cs

```
    public partial class ListViewMember : Form
    {
        SqlConnection connection
        public ListViewMember(SqlConnection connection)
        {
            InitializeComponent();
            this.connection = connection;
            FillComboBox();
        }
        private void FillComboBox()
        {
            Try
            {
                // Запит для отримання всіх учасників
                string query = "SELECT * FROM Member_of_the_gourt";
                SqlCommand command = new SqlCommand(query, connection);
                SqlDataReader reader = command.ExecuteReader();
                // Перевірка, чи є дані
                if (!reader.HasRows)
                {
                    MessageBox.Show("Немає учасників для відображення.");
                    return;
                }
                while (reader.Read())
                {
                    int ID = reader.GetInt32(0); // Отримуємо ID
```

```
string Name = reader.IsDBNull(1) ? "Невідомо" : reader.GetString(1); // Перевірка на null
string Phone = reader.IsDBNull(2) ? "Невідомо" : reader.GetString(2); // Перевірка на null
DateTime Date = reader.IsDBNull(3) ? DateTime.MinValue : reader.GetDateTime(3); // Перевірка на null
            int Course = reader.IsDBNull(4) ? 0 : reader.GetInt32(4); // Перевірка на null
string GroupName = reader.IsDBNull(5) ? "Невідомо" : reader.GetString(5); // Перевірка на null
string Specialty = reader.IsDBNull(6) ? "Невідомо" : reader.GetString(6); // Перевірка на null
            // Формуємо рядок для ComboBox
            string comboBoxItem = $"{ID} {Name}";
            comboBox1.Items.Add(comboBoxItem);
        }
        reader.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Помилка при завантаженні даних: " + ex.Message);
    }
}
private void button1_Click_1(object sender, EventArgs e)
{
    try
    {
        // Перевірка на наявність вибраного елемента в ComboBox
        if (comboBox1.SelectedItem == null)
        {
            MessageBox.Show("Будь ласка, виберіть учасника з ComboBox.");
            return;  // Вихід з методу, якщо нічого не вибрано
        }
        string selectedString = comboBox1.SelectedItem.ToString();
        const char V = ' ';
        string[] substring = selectedString.Split(V)
        if (substring.Length == 0)
        {
            MessageBox.Show("Невірний формат вибраного елемента.");
```

Сторінка 8

```
    return;
}
string code = substring[0].Trim(); // Отримуємо ID
string query = "SELECT * FROM Member_of_the_gourt WHERE ID_member = @code";

SqlCommand command = new SqlCommand(query, connection);
command.Parameters.AddWithValue("@code", code);
SqlDataReader reader = command.ExecuteReader();
// Перевірка на наявність результатів
if (!reader.HasRows)
{
    MessageBox.Show("Учасник не знайдений.");
    return;
}
// Очищення DataGridView перед додаванням нових даних
dataGridView1.Rows.Clear();
dataGridView1.Columns.Clear();
// Додавання стовпців до DataGridView
dataGridView1.Columns.Add("ColumnID", "Номер в журналі");
dataGridView1.Columns.Add("ColumnName", "ПІБ");
```

Сторінка 9

```
dataGridView1.Columns.Add("ColumnPhone", "Номер телефону");
dataGridView1.Columns.Add("ColumnDate", "Дата народження");
dataGridView1.Columns.Add("ColumnCourse", "Курс");
dataGridView1.Columns.Add("ColumnGroup", "Група");
dataGridView1.Columns.Add("ColumnSpecialty", "Спеціальність");
while (reader.Read())
{
    int ID = reader.GetInt32(0);
    string Name = reader.GetString(1);
    string Phone = reader.GetString(2);
    DateTime Date = reader.GetDateTime(3);
    int Course = reader.GetInt32(4);
    string GroupName = reader.GetString(5);
    string Specialty = reader.GetString(6);
    // Додаємо рядок до DataGridView
```

```
dataGridView1.Rows.Add(ID, Name, Phone, Date.ToShortDateString(), Course, GroupName, Specialty);
            }
            reader.Close();
        }
        catch (Exception ex)
        {
            MessageBox.Show("Помилка: " + ex.Message);
        }
    }
```

ZvitHead.cs

```
  public partial class ZvitHead : Form
  {
    SqlConnection connection;
    public ZvitHead(SqlConnection connection)
    {
      InitializeComponent();
      this.connection = connection;
    }
    private void ZvitHead_Load(object sender, EventArgs e)
    {
      this.reportViewer1.RefreshReport();
        }
    private void button1_Click(object sender, EventArgs e)
    {
      string queru = "Select * From GourtHead";
      SqlDataAdapter adapter = new SqlDataAdapter(queru, connection);
      DataSet ds = new DataSet();
      adapter.Fill(ds);
      adapter.Dispose();
      gourtHeadBindingSource.DataSource = ds.Tables[0];
      reportViewer1.RefreshReport();
    }
    private void ZvitHead_Load_1(object sender, EventArgs e)
    {
// TODO: данная строка кода позволяет загрузить данные в таблицу "dataSet.GourtHead". При необходимости она может быть перемещена или удалена.
      this.gourtHeadTableAdapter.Fill(this.dataSet.GourtHead);
```

```
            this.reportViewer1.RefreshReport();
        }
    }
```

Announcement.cs

```
    public partial class Announcement : Form
    {
        SqlConnection connection;
        string userRole; // "Head", "Member", "Director"
Dictionary<int, int> displayToRealId = new Dictionary<int, int>(); // Для зберігання відповідності між відображуваними та реальними ID
        public Announcement(SqlConnection conn, string role)
        {
            InitializeComponent();
            connection = conn;
            userRole = role;
            this.btnClear = new System.Windows.Forms.Button();
            //
            // btnClear
            //
this.btnClear.Location = new System.Drawing.Point(450, 300); // змінюй координати за потребою
            this.btnClear.Name = "btnClear";
            this.btnClear.Size = new System.Drawing.Size(120, 30);
            this.btnClear.TabIndex = 5;
            this.btnClear.Text = "Очистити поля";
            this.btnClear.UseVisualStyleBackColor = true;
            this.btnClear.Click += new System.EventHandler(this.btnClear_Click);
            this.Controls.Add(this.btnClear);
        }
        private void News_Load(object sender, EventArgs e)
        {
            LoadNews();
            // Перевірка на ролі користувача
            if (userRole != "Head" && userRole != "Director")
            {
                btnAdd.Enabled = false; // Заборонити додавання новин
                btnEdit.Enabled = false; // Заборонити редагування новин
                btnDelete.Enabled = false; // Заборонити видалення новин
```

```
        txtTitle.ReadOnly = true; // Заборонити редагування заголовку
        txtContent.ReadOnly = true; // Заборонити редагування тексту
    }
    // Встановлення тексту "Заголовок" у поле при завантаженні
    txtTitle.Text = "Заголовок";
    // Підключення обробників подій для txtTitle
    txtTitle.Enter += new EventHandler(txtTitle_Enter);
    txtTitle.Leave += new EventHandler(txtTitle_Leave);
}
// Обробник події Enter для txtTitle
private void txtTitle_Enter(object sender, EventArgs e)
{
    // Якщо текст в полі "Заголовок", замінюємо на порожній рядок
    if (txtTitle.Text == "Заголовок")
    {
        txtTitle.Text = "";
    }
}
// Обробник події Leave для txtTitle
private void txtTitle_Leave(object sender, EventArgs e)
{
    // Якщо поле порожнє, відновлюємо "Заголовок"
    if (string.IsNullOrWhiteSpace(txtTitle.Text))
    {
        txtTitle.Text = "Заголовок";
    }
}
private void LoadNews()
{
    listBoxNews.Items.Clear();
    displayToRealId.Clear(); // Очищуємо словник при оновленні
    // Змінюємо ORDER BY з DESC на ASC, щоб показувати новини в порядку їх додавання
    string query = "SELECT ID_news, Title, Date FROM News ORDER BY ID_news ASC";
    SqlCommand command = new SqlCommand(query, connection);
    SqlDataReader reader = command.ExecuteReader();
    int displayId = 1; // Послідовний номер для відображення
    while (reader.Read())
    {
```

Сторінка 12

```
            int realId = Convert.ToInt32(reader["ID_news"]);
            displayToRealId[displayId] = realId; // Зберігаємо відповідність

string item = $"{displayId} | {reader["Title"]} ({Convert.ToDateTime(reader["Date"]).ToShortDateString()})";
            listBoxNews.Items.Add(item);
            displayId++;
        }
        reader.Close();
    }
    private void listBoxNews_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (listBoxNews.SelectedItem == null) return;
        string selected = listBoxNews.SelectedItem.ToString();
        int displayId = int.Parse(selected.Split('|')[0].Trim());
        int realId = displayToRealId[displayId]; // Отримуємо реальний ID з словника
        string query = "SELECT Title, Content FROM News WHERE ID_news = @id";
        SqlCommand command = new SqlCommand(query, connection);
        command.Parameters.AddWithValue("@id", realId);
        SqlDataReader reader = command.ExecuteReader();
        if (reader.Read())
        {
            txtTitle.Text = reader["Title"].ToString();
            txtContent.Text = reader["Content"].ToString();
        }
        reader.Close();
    }
    private void btnAdd_Click(object sender, EventArgs e)
    {
        string title = txtTitle.Text.Trim();
        string content = txtContent.Text.Trim();

if (string.IsNullOrWhiteSpace(title) || string.IsNullOrWhiteSpace(content) || title == "Заголовок")
            {
MessageBox.Show("Заповніть всі поля.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
```

```
        }
        // Запит на додавання новини в базу даних
string query = "INSERT INTO News (Title, Content, Author_ID) VALUES (@title, @content, @author)";
        SqlCommand command = new SqlCommand(query, connection);
        command.Parameters.AddWithValue("@title", title);
        command.Parameters.AddWithValue("@content", content);
        command.Parameters.AddWithValue("@author", 1); // Вставте тут ID вашого автора
        command.ExecuteNonQuery();
MessageBox.Show("Новину успішно додано!", "Успіх", MessageBoxButtons.OK, MessageBoxIcon.Information);
        LoadNews(); // Оновлюємо новини на формі
        // Очищення полів після додавання
        txtTitle.Text = "Заголовок";
        txtContent.Clear();
    }
    private void btnEdit_Click(object sender, EventArgs e)
    {
        if (listBoxNews.SelectedItem == null)
        {
```



```
MessageBox.Show("Оберіть новину для редагування.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }
        string selected = listBoxNews.SelectedItem.ToString();
        int displayId = int.Parse(selected.Split('|')[0].Trim());
        int realId = displayToRealId[displayId]; // Отримуємо реальний ID з словника
        string title = txtTitle.Text.Trim();
        string content = txtContent.Text.Trim();
if (string.IsNullOrWhiteSpace(title) || string.IsNullOrWhiteSpace(content) || title == "Заголовок")
        {
MessageBox.Show("Заповніть всі поля.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }
string query = "UPDATE News SET Title = @title, Content = @content WHERE ID_news = @id";
```

```
            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@title", title);
            command.Parameters.AddWithValue("@content", content);
            command.Parameters.AddWithValue("@id", realId);
            command.ExecuteNonQuery();
MessageBox.Show("Новину успішно оновлено!", "Успіх", MessageBoxButtons.OK,
MessageBoxIcon.Information);
            LoadNews();
        }
        private void btnDelete_Click(object sender, EventArgs e)
        {
            if (listBoxNews.SelectedItem == null)
            {
MessageBox.Show("Оберіть новину для видалення.", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }
            // Перевірка на роль "Head" або "Director"
            if (userRole != "Head" && userRole != "Director")
            {
```



```
MessageBox.Show("У вас немає прав на видалення новин.", "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }
            // Отримуємо ID новини зі списку
            string selected = listBoxNews.SelectedItem.ToString();
            int displayId = int.Parse(selected.Split('|')[0].Trim());
            int realId = displayToRealId[displayId]; // Отримуємо реальний ID з словника
            // Запит на видалення новини з бази даних
            string query = "DELETE FROM News WHERE ID_news = @id";
            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@id", realId);
            try
            {
                command.ExecuteNonQuery();
MessageBox.Show("Новину успішно видалено!", "Успіх", MessageBoxButtons.OK,
MessageBoxIcon.Information);
                LoadNews(); // Оновлюємо список новин
```

```
            }
            catch (Exception ex)
            {
MessageBox.Show("Помилка при видаленні новини: " + ex.Message, "Помилка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
        private void btnClear_Click(object sender, EventArgs e) // Очистка полів
        {
            txtTitle.Text = "Заголовок";
            txtContent.Clear();
        }
    }
}
```

Сторінка 16

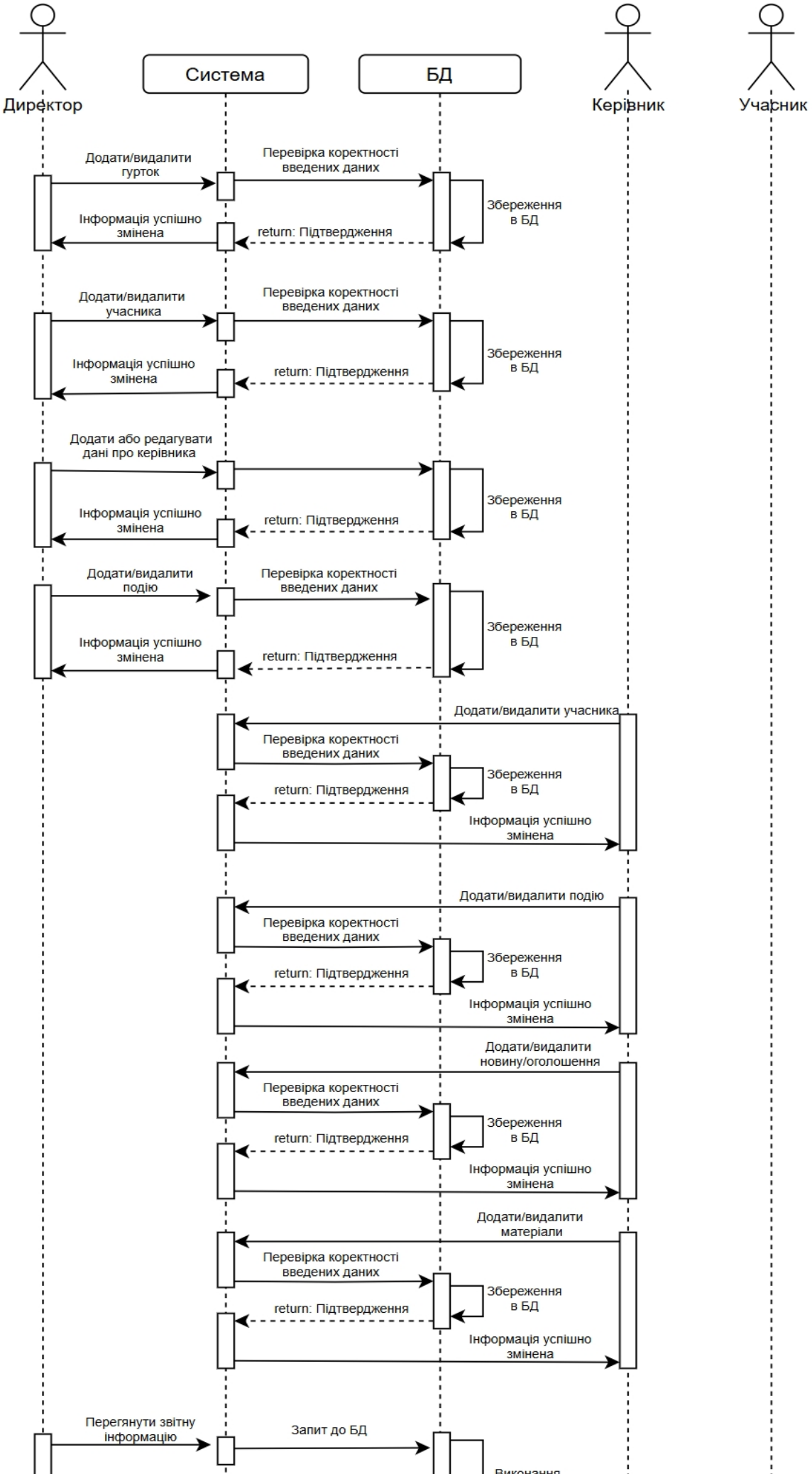
ДОДАТОК В

Діаграми

Сторінок 4

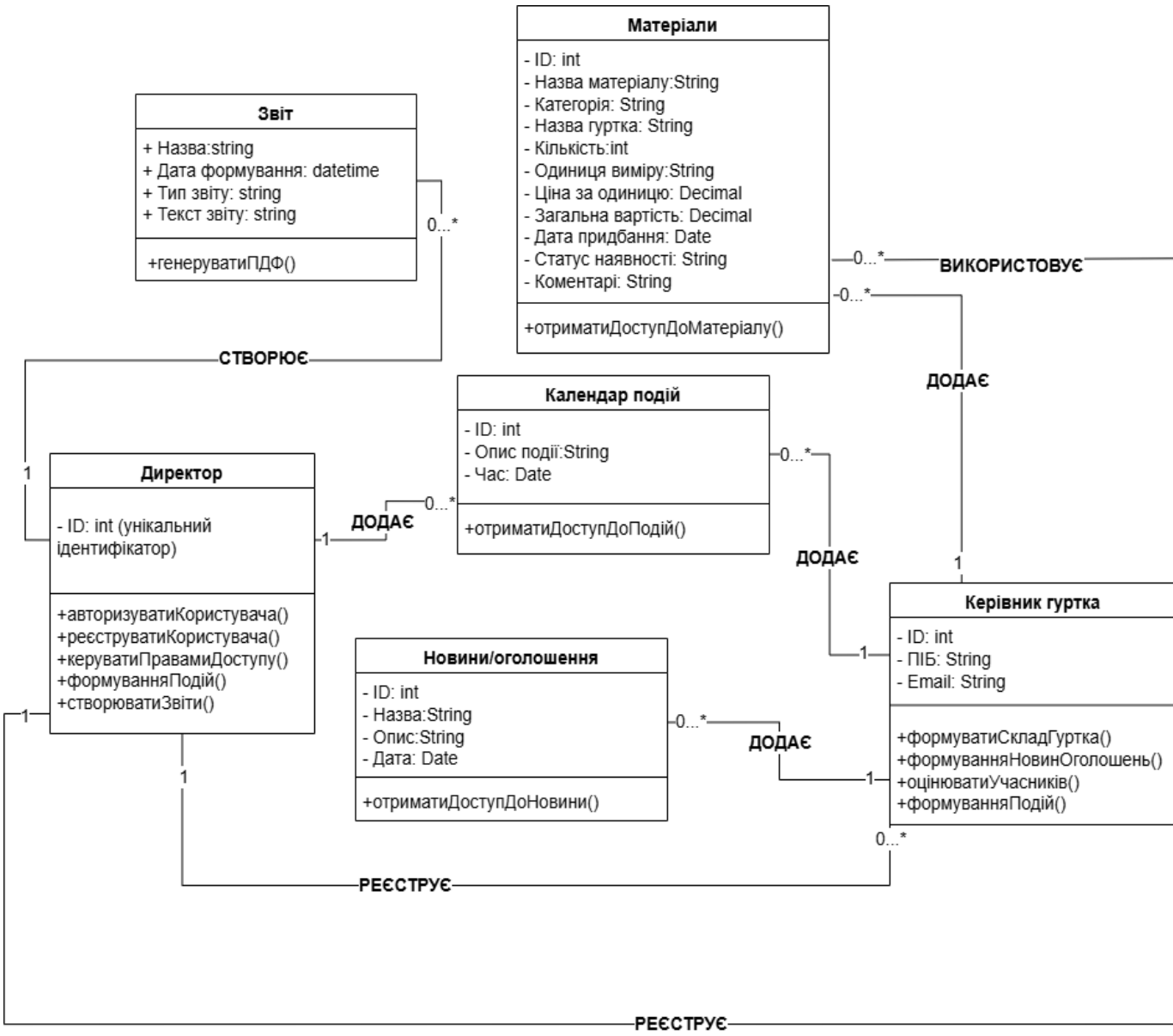
Київ – 2025

Діаграма послідовності

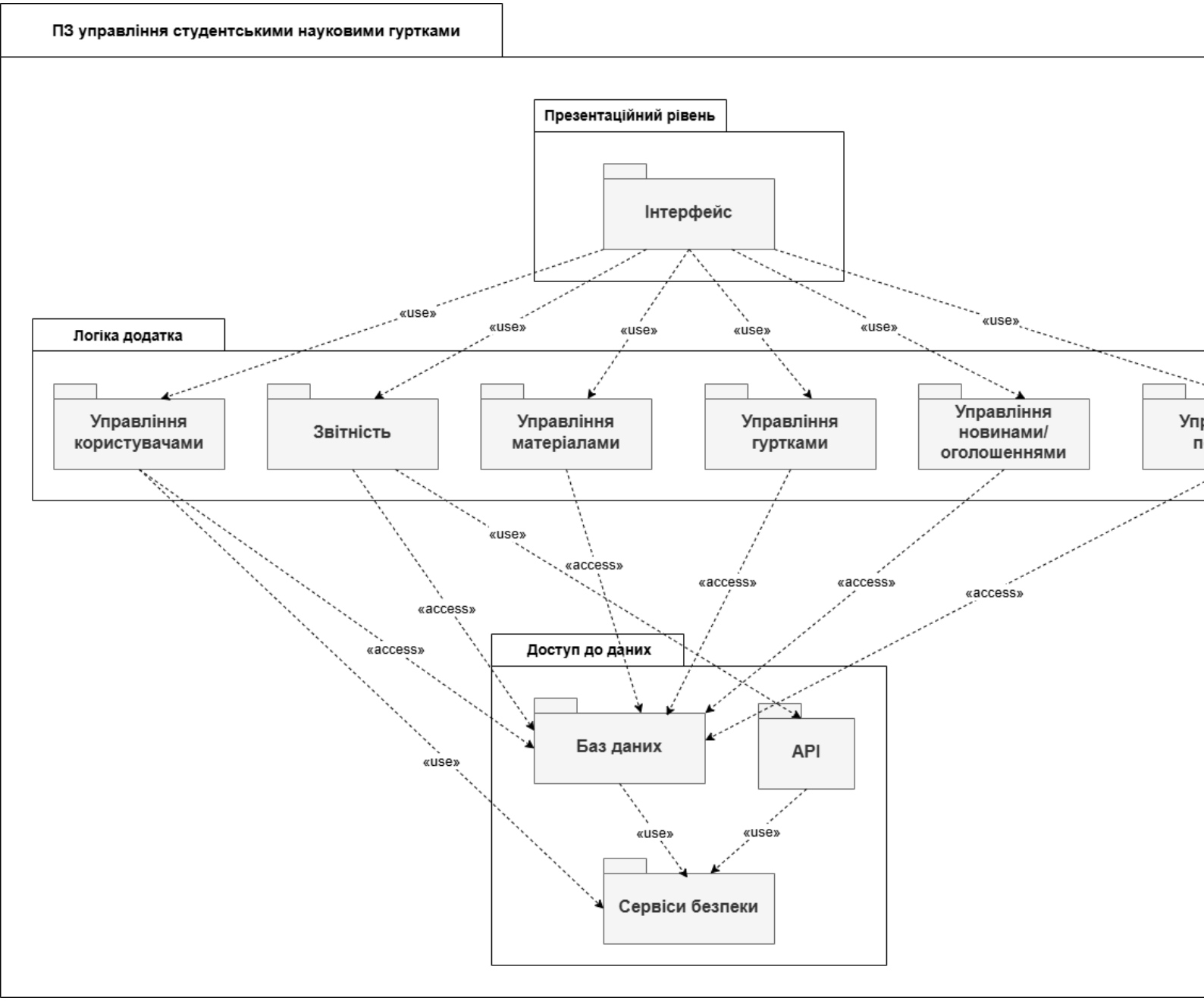

Директор
Система
БД
Керівник
Учасник
Додати/видалити гурток
Перевірка коректності введених даних
Збереження в БД
Інформація успішно змінена
return: Підтвердження
Додати/видалити учасника
Перевірка коректності введених даних
Збереження в БД
return: Підтвердження
Інформація успішно змінена
Додати або редагувати дані про керівника
Збереження в БД
Інформація успішно змінена
return: Підтвердження
Додати/видалити подію
Перевірка коректності введених даних
Збереження в БД
Інформація успішно змінена
return: Підтвердження
Додати/видалити учасника
Перевірка коректності введених даних
Збереження в БД
return: Підтвердження
Інформація успішно змінена
Додати/видалити подію
Перевірка коректності введених даних
Збереження в БД
return: Підтвердження
Інформація успішно змінена
Додати/видалити новину/оголошення
Перевірка коректності введених даних
Збереження в БД
return: Підтвердження
Інформація успішно змінена
Додати/видалити матеріали
Перевірка коректності введених даних
Збереження в БД
return: Підтвердження
Інформація успішно змінена
Перегянути звітну інформацію
Запит до БД

Сторінка 1

Діаграма класів


Матеріали
- ID: int
- Назва матеріалу:String
- Категорія: String
- Назва гуртка: String
- Кількість:int
- Одиниця виміру:String
- Ціна за одиницю: Decimal
- Загальна вартість: Decimal
- Дата придбання: Date
- Статус наявності: String
- Коментарі: String
+отриматиДоступДоМатеріалу()
Звіт
+ Назва:string
+ Дата формування: datetime
+ Тип звіту: string
+ Текст звіту: string
+генеруватиПДФ()
0...*
ВИКОРИСТОВУЄ
СТВОРЮЄ
ДОДАЄ
Календар подій
- ID: int
- Опис події:String
- Час: Date
+отриматиДоступДоПодій()
Директор
- ID: int (унікальний ідентифікатор)
+авторизуватиКористувача()
+реєструватиКористувача()
+керуватиПравамиДоступу()
+формуванняПодій()
+створюватиЗвіти()
Керівник гуртка
- ID: int
- ПІБ: String
- Email: String
+формуватиСкладГуртка()
+формуванняНовинОголошень()
+оцінюватиУчасників()
+формуванняПодій()
Новини/оголошення
- ID: int
- Назва:String
- Опис:String
- Дата: Date
+отриматиДоступДоНовини()
1
РЕЄСТРУЄ
РЕЄСТРУЄ

Діаграма пакетів


ПЗ управління студентськими науковими гуртками
Презентаційний рівень
Інтерфейс
Логіка додатка
Управління користувачами
Звітність
Управління матеріалами
Управління гуртками
Управління новинами/ оголошеннями
Доступ до даних
Баз даних
API
Сервіси безпеки
«use»
«access»

Сторінка 3

Діаграма компонентів