Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота

за освітнім ступенем «бакалавр»

на тему:

“Застосунок для відстеження академічного прогресу студентів”

Виконав:
здобувач IV курсу
групи КН-41
спеціальності 122 «Комп’ютерні науки»
Олександр Олександрович Бойко

Науковий керівник::
старший викладач Тетяна Анатоліївна Кирик

Рівне – 2025

ЗМІСТ

ВСТУП

У сучасному світі інформаційних технологій автоматизація процесів стає ключовим фактором для підвищення ефективності роботи у багатьох сферах, зокрема в освіті. Автоматизовані системи дозволяють значно полегшити управління навчальним процесом, забезпечити точність обліку успішності студентів, мінімізувати вплив людського фактору та підвищити загальну якість освітнього середовища. Саме тому розробка застосунку для автоматизованого обліку та аналізу успішності студентів є актуальною та перспективною темою дипломної роботи бакалавра.

Об’єктом дослідження є процеси організації та ведення обліку успішності студентів у навчальному закладі. Це включає всі аспекти, пов’язані з внесенням, зберіганням, обробкою та аналізом даних щодо оцінок студентів, участі в курсах, динаміки їх навчальних досягнень, а також взаємодію між викладачами, адміністрацією та самими студентами.

Предметом дослідження є розробка та впровадження програмного забезпечення для автоматизації обліку навчальних досягнень студентів. Особливу увагу приділено створенню застосунку, який дозволяє зручно вносити, редагувати та переглядати дані, здійснювати базовий аналіз навчальних результатів, а також забезпечує збереження й безпеку даних.

Мета дослідження полягає у створенні робочого прототипу програмного продукту, що забезпечить автоматизацію процесів обліку та аналізу успішності студентів, відповідатиме потребам викладачів та адміністрації навчального закладу, буде простим у використанні, розширюваним та безпечним.

Структура роботи. Основна частина кваліфікаційної роботи складається з чотирьох основних розділів.

Перший розділ присвячено теоретичним аспектам розробки інформаційних систем: тут наведено огляд сучасних рішень автоматизації освітнього процесу в Україні, а також описано принципи проєктування таких систем.

У другому розділі розглядається аналіз вимог і проєктування системи, моделювання бізнес-процесів, побудова архітектури, демонстрація інтерфейсу, а також оцінка впливу системи на навчальний процес.

Третій розділ охоплює технологічні аспекти розробки: вибір мови програмування, фреймворків, опис побудови бази даних, реалізацію клієнтської й серверної частин, заходи безпеки та результати тестування.

Четвертий розділ описує перспективи розвитку, включаючи плани розширення функціоналу, розширеної аналітики, впровадження чат-бота, хмарної синхронізації та адміністрування.

Для досягнення поставленої мети в ході роботи було реалізовано такі кроки:

1. **Проведено аналіз існуючих рішень** для автоматизації обліку успішності студентів, визначено їхні переваги та недоліки.
2. **Сформульовано вимоги до системи**, підготовлено технічне завдання, яке включає опис функціоналу, технічних обмежень та очікуваних результатів.
3. **Розроблено архітектуру застосунку**, що забезпечує модульність, масштабованість та зручність підтримки.
4. **Спроектовано базу даних**, яка дозволяє ефективно зберігати дані про студентів, курси та оцінки.
5. **Реалізовано користувацький інтерфейс** для роботи з даними, що відповідає принципам зручності та доступності.
6. **Інтегровано функціонал для виконання базових аналітичних запитів** (середні оцінки, список оцінок за курсом, пошук за студентом).
7. **Проведено тестування застосунку**, включаючи перевірку основного функціоналу та роботи з даними.

8. **Визначено обмеження поточної версії та сформульовано перспективи розвитку**, включаючи можливості для впровадження розширеної аналітики, чат-бота та автоматичної генерації звітів.

У роботі використано комплексний підхід, що об’єднує такі методи дослідження: **теоретичні** (аналіз, синтез, порівняння, систематизація — для вивчення наукових джерел, аналізу літератури, існуючих програмних рішень та вибору найбільш оптимальних концепцій для проєкту), **емпіричні** (спостереження, інтерв’ю з користувачами — для збору вимог, визначення потреб кінцевих користувачів), **моделювання** (створення схем архітектури, моделей даних, бізнес-процесів — для візуалізації й побудови логіки роботи системи), **експериментальні методи** (розробка прототипу, **тестування** — для перевірки працездатності розробленого програмного продукту). Кожен з методів відігравав важливу роль у досягненні поставленої мети: аналітичні методи допомогли зібрати та систематизувати знання про предметну область, моделювання дозволило чітко уявити архітектуру, а реалізація та тестування забезпечили створення та перевірку прототипу системи.

У наступних розділах буде детально розглянуто теоретичні основи розробки інформаційних систем для освіти, аналіз існуючих рішень, а також власний підхід до проектування та реалізації розроблюваного застосунку.

РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ ІНФОРМАЦІЙНИХ СИСТЕМ

1.1 Огляд сучасних рішень автоматизації освітнього процесу в Україні

Станом на сьогодні в Україні активно впроваджуються різні програмні засоби для автоматизації навчального процесу. Серед найбільш поширених рішень можна виокремити кілька ключових систем.

Перша категорія — електронні журнали та щоденники. Прикладами таких рішень є система «Нові знання» та «Електронний журнал». «Нові знання» дозволяє вчителям вести облік оцінок, відвідуваності, домашніх завдань, а також забезпечує доступ до інформації для батьків і учнів. Її перевагою є наявність веб-інтерфейсу та можливість формування звітів [1].

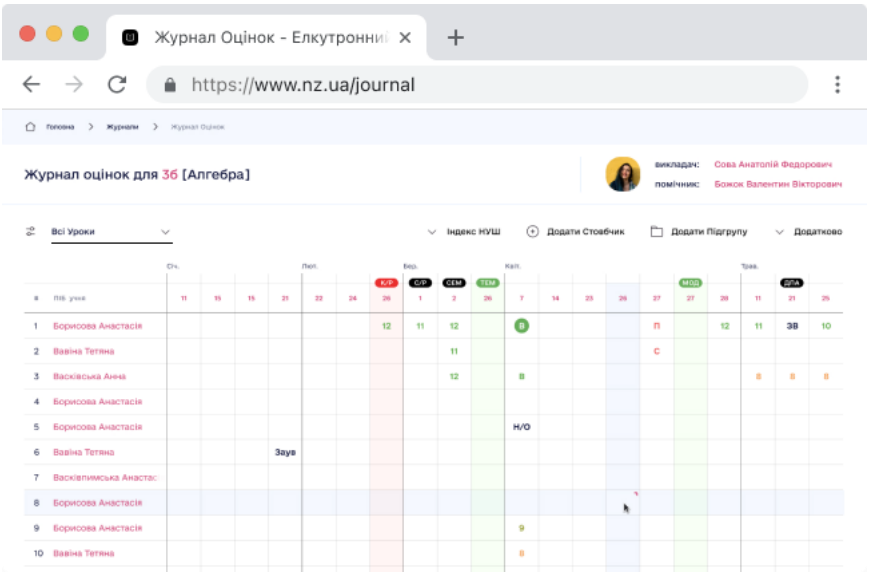

Рисунок 1.1. Журнал оцінок системи «Нові знання»

«Електронний журнал» — це ще одна система, яка підтримує введення оцінок, облік тем занять, формування розкладу та надання статистичних звітів для адміністрації. Обидві системи орієнтовані переважно на школи, їхні переваги —

прозорість навчального процесу та інтеграція з внутрішніми управлінськими процесами [2].

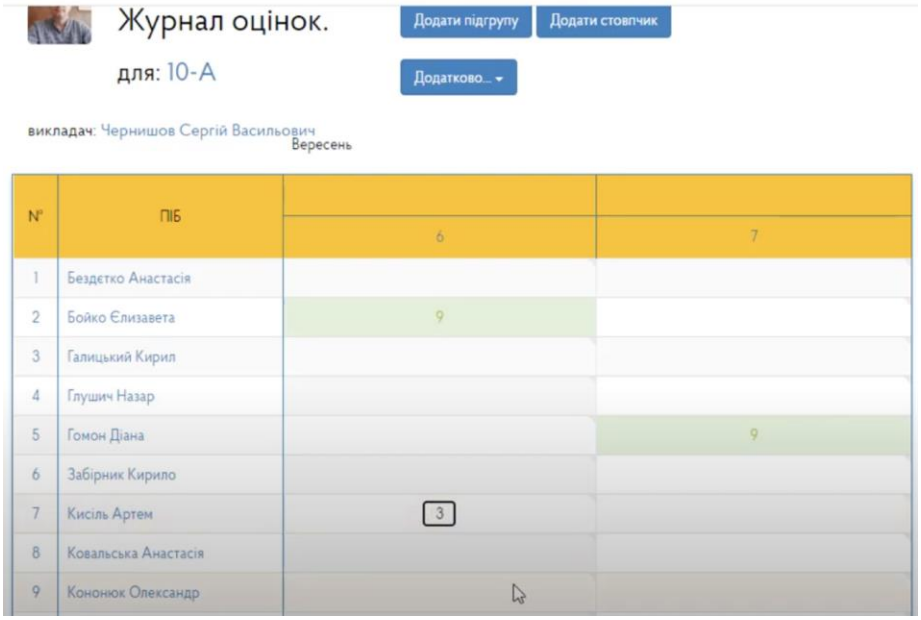

Рисунок 1.2. Журнал оцінок системи «Електронний журнал»

Друга категорія — системи управління навчанням (LMS). До цієї групи належать Moodle, Canvas, Edmodo. **Moodle** є однією з найпопулярніших платформ у світі й широко використовується в українських університетах. Вона дозволяє створювати курси, завантажувати матеріали, організовувати тести, проводити оцінювання, а також забезпечує зворотний зв’язок між студентами та викладачами [3].

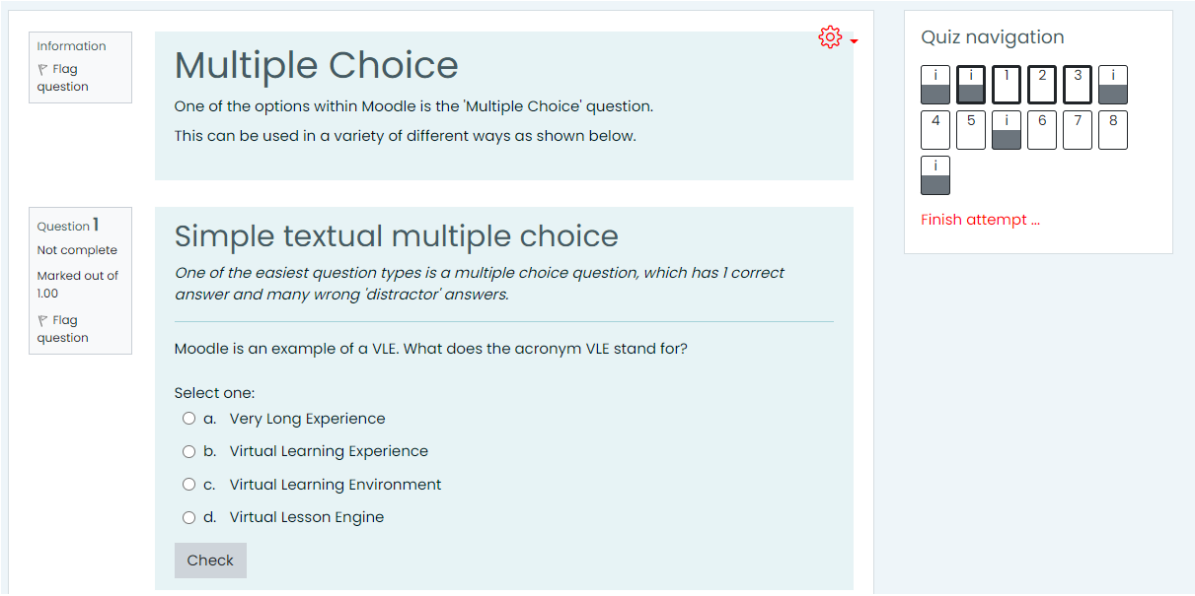

Рисунок 1.3. Приклад тесту на платформі «**Moodle**»

Canvas — ще одна потужна LMS, яка орієнтується на простий інтерфейс і зручність користування, але в Україні використовується рідше [4].

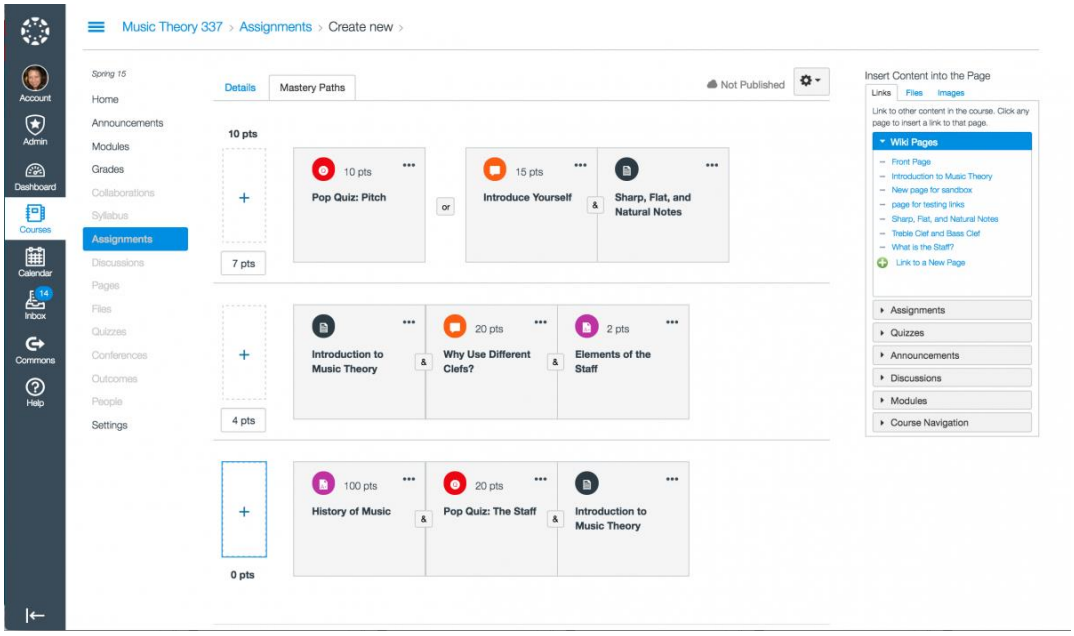

Рисунок 1.4. Приклад курсу на платформі «**Canvas**»

Edmodo, своєю чергою, фокусується на створенні спільнот між учнями, вчителями та батьками, пропонуючи простий інтерфейс для спілкування та управління завданнями [5].

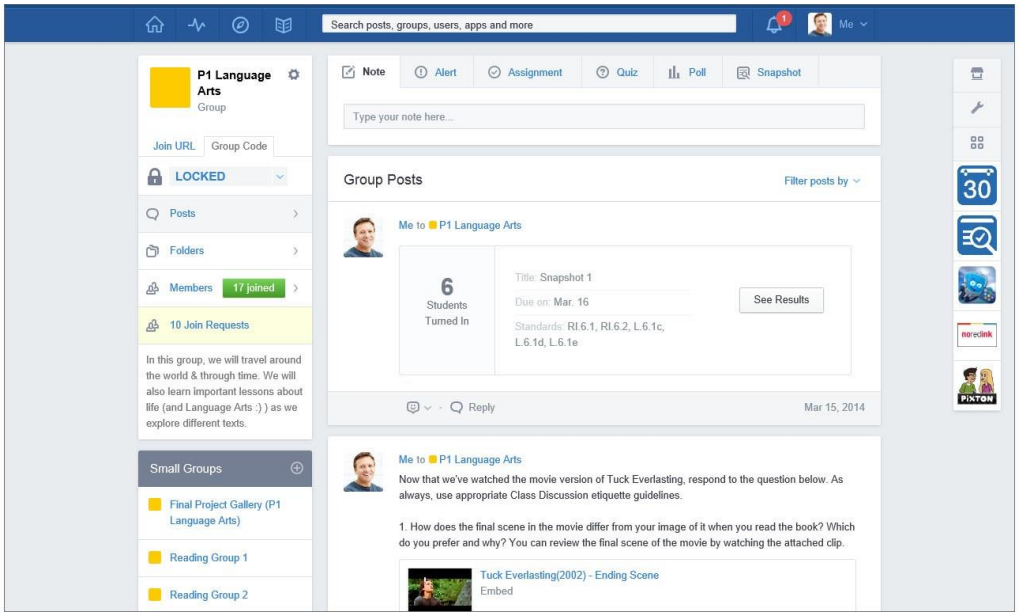

Рисунок 1.5. Приклад групи на платформі «**Edmodo**»

Третя категорія — платформи для організації дистанційного навчання. Google Classroom — це популярний безкоштовний продукт від Google, який дозволяє створювати віртуальні класи, розподіляти завдання, виставляти оцінки та забезпечувати комунікацію між учнями та викладачами. Серед переваг платформи — глибока інтеграція з іншими сервісами Google (Docs, Sheets, Drive), можливість працювати з будь-якого пристрою, простота використання. Проте Google Classroom більше орієнтований на онлайн-навчання, ніж на повноцінне адміністрування шкільного чи університетського документообігу [6].

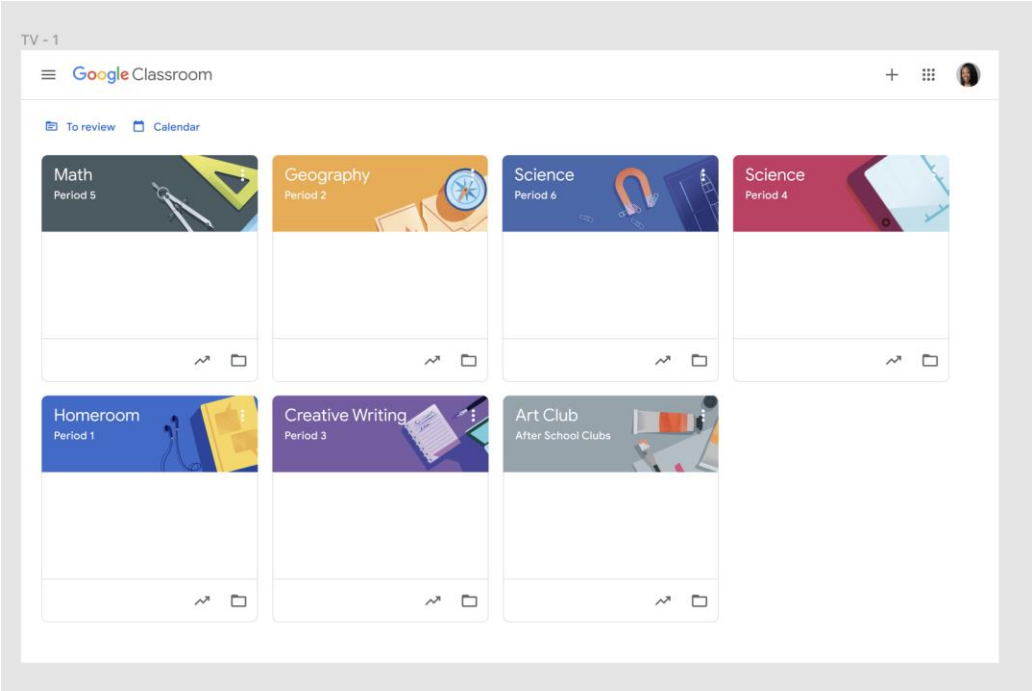

Рисунок 1.6. Список класів на платформі «**Classroom**»

Четверта категорія — державні системи електронного документообігу. Найбільш відомою є ЄДЕБО (Єдина державна електронна база з питань освіти), яка використовується Міністерством освіти та науки України, адміністраціями навчальних закладів і університетами для обробки даних про студентів, формування звітів, адміністрування вступних кампаній, перевірки документів та атестатів. Система виконує важливу роль на макрорівні, але не призначена для локального ведення обліку на рівні окремого викладача або кафедри [7].

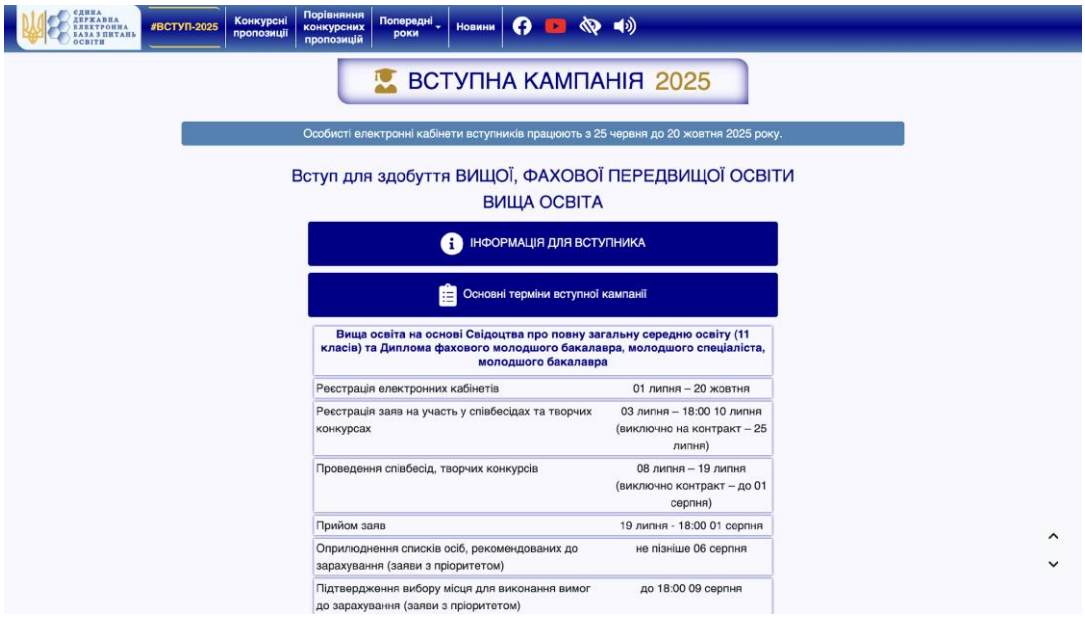

Рисунок 1.7. Вступна кампанія на порталі ЄДЕБО

Порівнюючи зазначені системи, можна визначити їх спільні характеристики:

- централізоване зберігання даних;
- наявність багаторівневого доступу залежно від ролі користувача (студент, викладач, адміністратор);
- інтеграція з зовнішніми сервісами та платформами;
- підтримка базових аналітичних функцій;
- вимоги до серверної інфраструктури або підключення до хмарних сервісів.

Незважаючи на багатий функціонал, більшість існуючих систем мають обмеження: вони часто надлишково складні для малих організацій, вимагають коштів на впровадження, ліцензії, адміністрування, а також навчання персоналу. У випадку з малими навчальними центрами, гуртками або навіть окремими викладачами виникає потреба в спрощених, легких рішеннях, що можуть працювати локально, без підключення до масштабних систем.

Саме цю нішу покликаний заповнити розроблений у межах дипломної роботи застосунок, який дозволяє вести облік студентів, курсів та оцінок у простому форматі,

забезпечуючи базовий аналіз даних та можливість гнучкої адаптації під конкретні потреби.

У наступному підрозділі розглядатимуться загальні принципи, що лежать в основі проектування інформаційних систем подібного типу, а також те, як вони були враховані у нашій розробці.

1.1 Принципи проектування інформаційних систем

Проектування інформаційних систем — це багаторівневий процес, який охоплює концептуальне, логічне та фізичне моделювання, а також враховує цілу низку принципів, що забезпечують ефективність, надійність, масштабованість та довговічність створюваного продукту. Від правильного застосування цих принципів залежить успішність впровадження системи, її здатність відповідати змінним потребам користувачів та стабільно працювати в різних умовах експлуатації.

Принцип модульності. Модульна структура передбачає поділ системи на окремі частини (модулі), кожен з яких виконує свою чітко визначену функцію. Такий підхід дозволяє одночасно працювати над різними компонентами командам розробників, спрощує тестування, оновлення та підтримку системи. У нашому проекті виокремлено модулі для управління студентами, управління курсами, обліку оцінок, взаємодії з базою даних та адміністрування користувачів. Завдяки цьому можна окремо оновлювати або вдосконалювати, наприклад, модуль оцінювання без ризику збоїв в інших частинах системи.

Принцип гнучкості. Гнучкість означає, що система повинна легко адаптуватися до змін, наприклад, додавання нових функцій або зміни бізнес-логіки. Для цього застосовуються архітектурні рішення, що дозволяють змінювати окремі компоненти без впливу на загальну роботу продукту. Під час проектування нашої системи враховувалася можливість розширення — наприклад, майбутнього

підключення модуля аналітики або формування автоматичних звітів без перепроектування всієї структури.

Принцип масштабованості. Масштабованість системи — це її здатність підтримувати стабільну продуктивність при зростанні обсягу даних, кількості користувачів або навантаження. У розробці нашого застосунку особлива увага приділялася оптимізації запитів до бази даних, запобіганню блокуванню ресурсів та можливості використання системи як у локальному середовищі, так і з перспективою перенесення на серверне або хмарне рішення.

Принцип надійності. Надійність означає, що система повинна продовжувати роботу навіть у разі часткових відмов компонентів. Це включає реалізацію механізмів перевірки правильності введених даних, відновлення після збоїв, створення резервних копій бази даних та логування критичних операцій для виявлення проблемних місць. Для нашого продукту передбачено функції резервного копіювання, що дозволяє відновлювати дані у разі аварійних ситуацій.

Принцип безпеки. Безпека є ключовим аспектом сучасних інформаційних систем, особливо коли йдеться про персональні дані студентів та інформацію про їхню успішність. Забезпечення конфіденційності, цілісності та доступності даних досягається через багаторівневу автентифікацію, шифрування чутливих даних, обмеження доступу відповідно до ролей користувачів, а також регулярне оновлення систем безпеки.

Принцип інтуїтивної зрозумілості інтерфейсу. Зручність користування системою безпосередньо впливає на її прийнятність для кінцевих користувачів. Інтерфейс повинен бути логічно структурованим, зрозумілим та адаптованим до основних сценаріїв використання. В нашій системі розроблено прості меню, зрозумілі кнопки, підказки для користувача, а також механізми повідомлень про помилки або підтвердження дій.

Принцип повторного використання компонентів. Використання готових бібліотек, фреймворків і перевірених архітектурних шаблонів дозволяє скоротити час

розробки, підвищити надійність системи та уникнути помилок. У нашій роботі застосовуються такі інструменти, як .NET MAUI для кросплатформенного інтерфейсу та Entity Framework Core для взаємодії з базою даних, що дозволяє забезпечити узгодженість і повторюваність рішень.

Принцип забезпечення продуктивності. Система має обробляти запити користувачів швидко та ефективно. Це передбачає грамотне проектування алгоритмів, оптимізацію роботи з базами даних, кешування часто використовуваних даних та мінімізацію часу виконання найважливіших операцій. Під час проектування нашої системи було проведено аналіз можливих вузьких місць і впроваджено заходи для підвищення швидкодії.

Принцип супровідності. Супровідність означає, що систему можна легко підтримувати та оновлювати упродовж усього життєвого циклу. Це досягається завдяки чистій архітектурі, грамотній документації, коментарям у коді та використанню єдиних стандартів програмування.

Усі зазначені принципи були враховані при проектуванні та реалізації системи автоматизованого обліку успішності студентів, що дозволило створити надійну основу для подальшого розвитку продукту. У наступному розділі буде представлено аналіз функціональних вимог до системи та описано ключові архітектурні рішення, прийняті під час проектування.

РОЗДІЛ 2. АНАЛІЗ І ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Аналіз функціональних вимог до системи

Аналіз функціональних вимог є одним з ключових етапів проектування будь-якої інформаційної системи, оскільки саме на цьому етапі формується розуміння того, що саме повинна робити система, які завдання вона має вирішувати, хто є її основними користувачами та які сценарії взаємодії мають бути передбачені.

У межах розробки застосунку для автоматизованого обліку та аналізу успішності студентів було визначено кілька основних груп користувачів, кожна з яких має свої вимоги та очікування від системи:

1. **Викладачі.** Для них система повинна надавати можливості:
   - вводити та змінювати відмітки присутності;
   - вводити, змінювати та переглядати оцінки;
   - переглядати підсумкові результати за курсом або студентом;
   - отримувати прості текстові звіти про успішність.
2. **Студенти.** На цьому етапі розробки передбачено лише базовий функціонал:
   - перегляд особистих оцінок;
   - перегляд інформації про курси, на які вони записані;
   - отримання інформації від викладачів.
3. **Адміністрація (планується в майбутньому).** Хоча у поточній версії застосунку адміністративний модуль не реалізовано, під час аналізу вимог було враховано потребу в наступних функціях для подальшого розвитку:
   - доступ до загальної статистики успішності по всіх курсах;
   - управління правами доступу користувачів;
   - наявність інструментів резервного копіювання та відновлення даних;
   - перегляд історії змін у системі (логування дій користувачів).

Крім цього, були визначені загальні функціональні вимоги:

- **Облік студентів.** Система повинна зберігати дані про студентів, включаючи їхні ПІБ, контактну інформацію, групу та список записаних курсів.
- **Облік курсів.** Має бути передбачено можливість створення, редагування та видалення курсів, додавання опису курсу та розподілу студентів по групах.
- **Облік оцінок.** Важливою є можливість введення оцінок для студентів по кожному курсу, а також зміни або видалення некоректно внесених оцінок.
- **Формування підсумкової інформації.** Для викладачів мають бути доступні підсумкові дані: середні бали по студенту, середні бали по курсу, рейтинг студентів тощо.
- **Базова аналітика.** Передбачено прості аналітичні запити, наприклад виявлення студентів з найнижчими або найвищими оцінками, розрахунок середніх оцінок за період, підрахунок кількості виставлених оцінок.
- **Безпека даних.** Обов’язковим є захист даних від несанкціонованого доступу, що передбачає реалізацію автентифікації користувачів, обмеження доступу відповідно до ролей, а також захищене зберігання конфіденційної інформації.
- **Зручність інтерфейсу.** Інтерфейс системи має бути зрозумілим і доступним для користувачів без спеціальної технічної підготовки.
- **Надійність роботи.** Система повинна працювати стабільно, передбачати механізми збереження даних у разі збою та мати можливість резервного копіювання.

Під час аналізу вимог окремо враховувалися обмеження, пов’язані з обсягами даних, кількістю користувачів, можливістю використання системи як на окремому комп’ютері, так і в локальній мережі.

Загалом функціональні вимоги були сформовані на основі проведених інтерв’ю з викладачами, аналізу існуючих рішень, а також тестового моделювання робочих сценаріїв, що дозволило створити основу для подальшого архітектурного проектування системи.

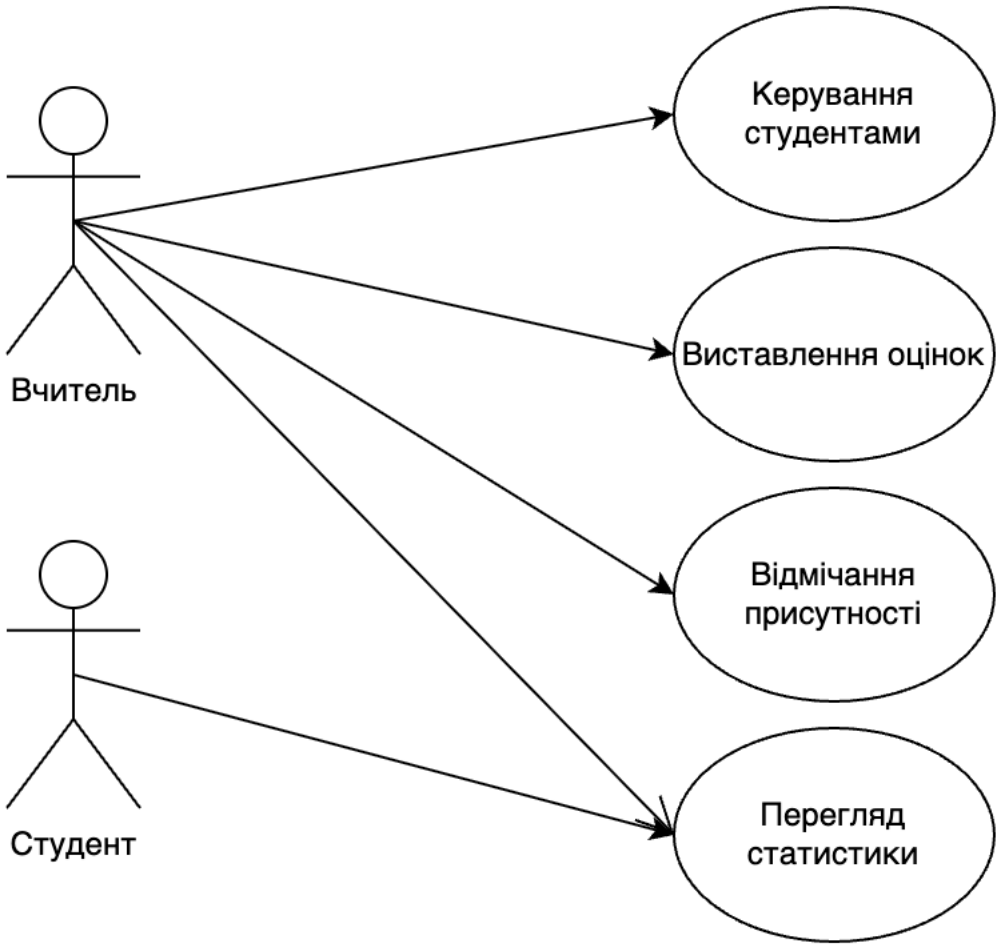

Рисунок 2. Use-case діаграма застосунку

2.2 Моделювання бізнес-процесів

Моделювання бізнес-процесів — це ключовий етап розробки інформаційної системи, який дозволяє зрозуміти, як будуть взаємодіяти різні користувачі, які дані проходитимуть через систему та як виглядатимуть основні сценарії використання. Для нашого застосунку, що автоматизує облік успішності студентів, моделювання допомогло побудувати логічну структуру взаємодії між студентами, викладачами, а також передбачити можливість розширення системи під адміністративні потреби.

Основні учасники бізнес-процесів:

1. **Викладач:**
    - Відповідає за створення курсів, внесення студентів у курси, виставлення оцінок, перегляд підсумкових результатів.
    - Має змогу аналізувати успішність групи студентів, виявляти студентів з низькими або високими результатами, формувати прості текстові звіти.
2. **Студент:**
    - Має доступ до особистого кабінету для перегляду своїх оцінок, підсумків по кожному курсу та отримання повідомлень від викладачів.
    - Не має права змінювати свої дані або редагувати інформацію в системі.
3. **Адміністратор (передбачений у майбутньому):**
    - Буде здійснювати контроль за системою, створювати облікові записи викладачів, надавати або обмежувати права доступу, забезпечувати резервне копіювання та відновлення даних.

Основні бізнес-процеси системи:

- **Створення та налаштування курсу:** Викладач формує новий курс, додає до нього назву, опис, інформацію про графік, а також запрошує студентів.
- **Додавання оцінок:** Викладач після проведених занять або тестувань вносить результати студентів у систему. Система автоматично розраховує середній бал, рейтингові показники та оновлює дані в особистих кабінетах студентів.
- **Перегляд результатів:** Студенти мають змогу увійти в систему, переглянути свої оцінки, загальні підсумки по кожному курсу, а також дізнатися, які завдання ще залишаються невиконаними.
- **Аналіз успішності:** Викладач отримує можливість швидко переглянути зведення по групі: хто має найвищі або найнижчі показники, які теми викликали найбільше труднощів.

- **Адміністрування системи (майбутній модуль):** Передбачається, що адміністратор зможе керувати налаштуваннями системи, проводити аудит змін, забезпечувати регулярне резервування даних, вирішувати проблеми з обліковими записами.

Візуалізація бізнес-процесів (загальна логіка):

1. Створення курсу → Додавання студентів → Введення оцінок → Перегляд результатів.
2. Викладач → Формування запитів → Аналіз даних → Прийняття рішень (наприклад, про необхідність додаткових занять).
3. Студент → Отримання інформації → Зворотний зв'язок через викладача (питання, уточнення).

Моделювання бізнес-процесів дозволило чітко визначити, які етапи потребують автоматизації, як організувати логіку доступу для різних ролей користувачів та які точки розширення потрібно передбачити в архітектурі. Це стало основою для формування технічного завдання та подальшого проектування архітектури застосунку.

2.3 Проектування архітектури системи

Проектування архітектури системи є центральним етапом розробки, адже саме на цьому рівні визначаються структурні компоненти, їх взаємодія, розподіл відповідальностей, потоки даних і способи масштабування. Для нашого застосунку архітектура проектувалася з урахуванням простоти, надійності та можливості подальшого розширення.

Обраний архітектурний підхід:

Система побудована за архітектурним патерном Model-View-ViewModel (MVVM), що дозволяє розділити представлення інтерфейсу користувача (View), логіку управління даними (ViewModel) і безпосередньо модель даних (Model). Це забезпечує:

- легкість супроводу й оновлення;
- зручність тестування окремих компонентів;
- масштабованість, що дозволяє додавати нові модулі без перепроектування всієї системи.

Ключові компоненти архітектури:

1. **View (Інтерфейс користувача):** реалізований за допомогою .NET MAUI, забезпечує кросплатформенний інтерфейс, доступний як на десктопі, так і на мобільних пристроях. Інтерфейс включає форми для реєстрації студентів, викладачів, сторінки для введення оцінок, перегляду результатів тощо.
2. **ViewModel (Логіка керування):** є проміжною ланкою між інтерфейсом та даними. Забезпечує обробку подій користувача, виконує валідацію введених даних, управляє станом інтерфейсу. ViewModel використовує сервіси для роботи з базою даних та аналітичними запитами.
3. **Model (Дані):** описує структуру даних, що використовуються в системі — студенти, курси, оцінки, ролі користувачів. Дані моделі є основою для взаємодії з базою даних через ORM-бібліотеку Entity Framework Core.
4. **Сховище даних:** система використовує SQLite як локальну базу даних. Це дає змогу зберігати всі ключові дані без необхідності підключення до мережі, забезпечує швидкість доступу й простоту резервного копіювання.
5. **Сервіси:** окремий шар логіки, який включає операції з базою даних, роботу з файлами (наприклад, імпорт/експорт даних), обробку аналітичних запитів. Сервіси можуть бути легко замінені або розширені без впливу на інші компоненти системи.

Логіка взаємодії компонентів:

- Користувач взаємодіє з інтерфейсом (View), наприклад, натискає кнопку «Додати оцінку».
- ViewModel перехоплює цю дію, перевіряє правильність даних, формує запит до відповідного сервісу.
- Сервіс передає дані в Model, яка через Entity Framework Core взаємодіє з базою даних.
- Після успішного збереження інформації ViewModel оновлює інтерфейс, щоб користувач бачив актуальні дані.

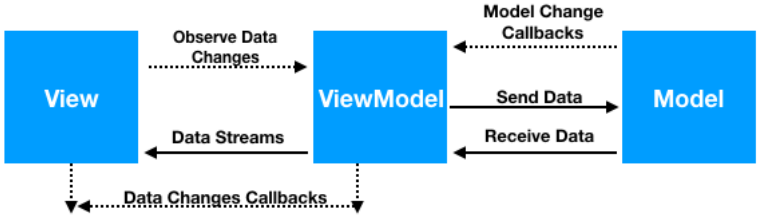

Рисунок 2.1. Діаграма архітектури MVVM

Передбачення на майбутнє: Архітектура передбачає можливість:

- підключення ролі адміністратора для управління правами доступу;
- додавання модуля аналітики з графічними звітами;
- інтеграції з хмарним сховищем для синхронізації даних між пристроями;
- реалізації чат-бота для автоматичної відповіді на прості запити користувачів.

Таким чином, побудова системи на основі патерну MVVM у поєднанні з розділенням логіки на окремі шари дозволила закласти основу для розширюваної, підтримуваної та гнучкої системи, що відповідає сучасним стандартам розробки.

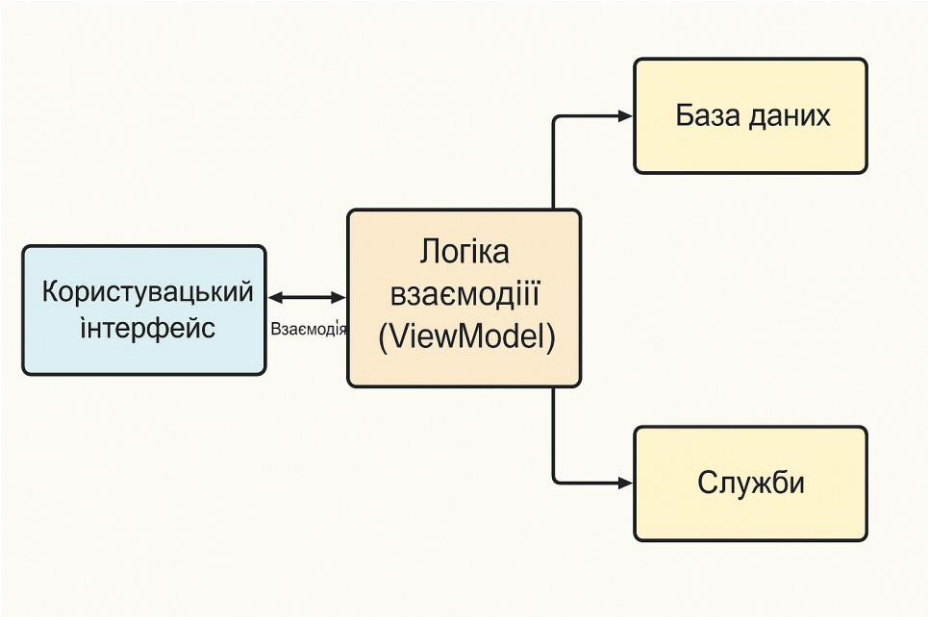

Рисунок 2.2. Схема архітектури застосунку

2.4 Демонстрація інтерфейсу застосунку

Інтерфейс користувача є критично важливою частиною будь-якого програмного продукту, оскільки саме він визначає зручність, зрозумілість та швидкість роботи користувача із системою. У розробленому застосунку для автоматизованого обліку успішності студентів інтерфейс створено відповідно до сучасних вимог щодо UX/UI-дизайну, з акцентом на простоту та мінімалізм.

Основні екрани та функціональність:

1. **Головна сторінка (Dashboard):**
    - Вітальний екран для викладача або студента залежно від ролі.
    - Відображення загальної інформації про кількість курсів, кількість студентів у групах (для викладача), або короткий огляд результатів (для студента).

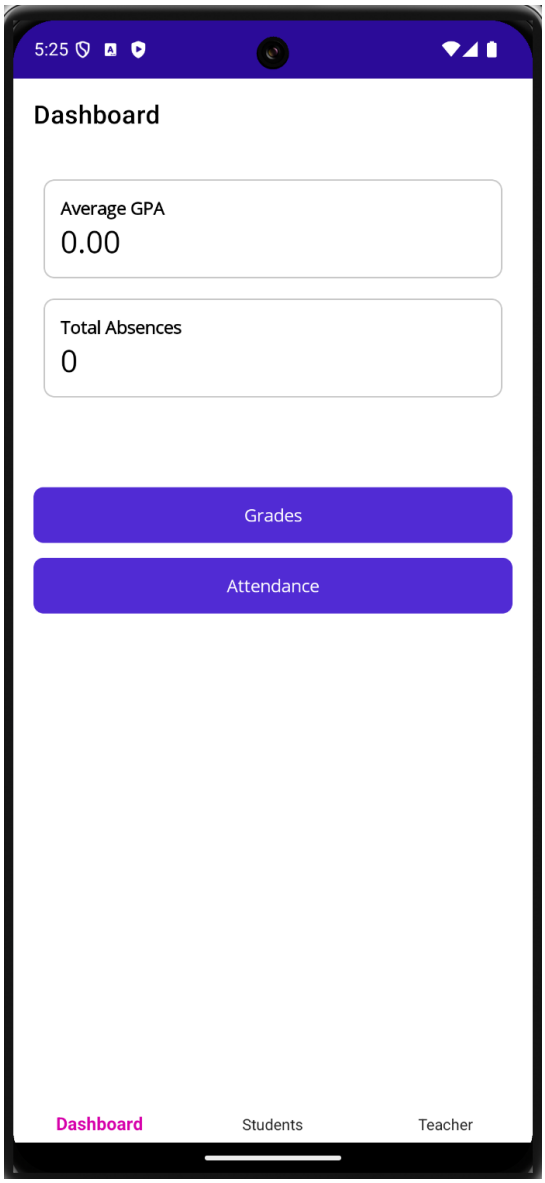

Рисунок 2.3. Сторінка Dashboard

2. **Сторінка керування курсами:**
    - Викладачі можуть переглядати список своїх курсів.
    - Форми для внесення опису курсу, тривалості, розкладу занять.

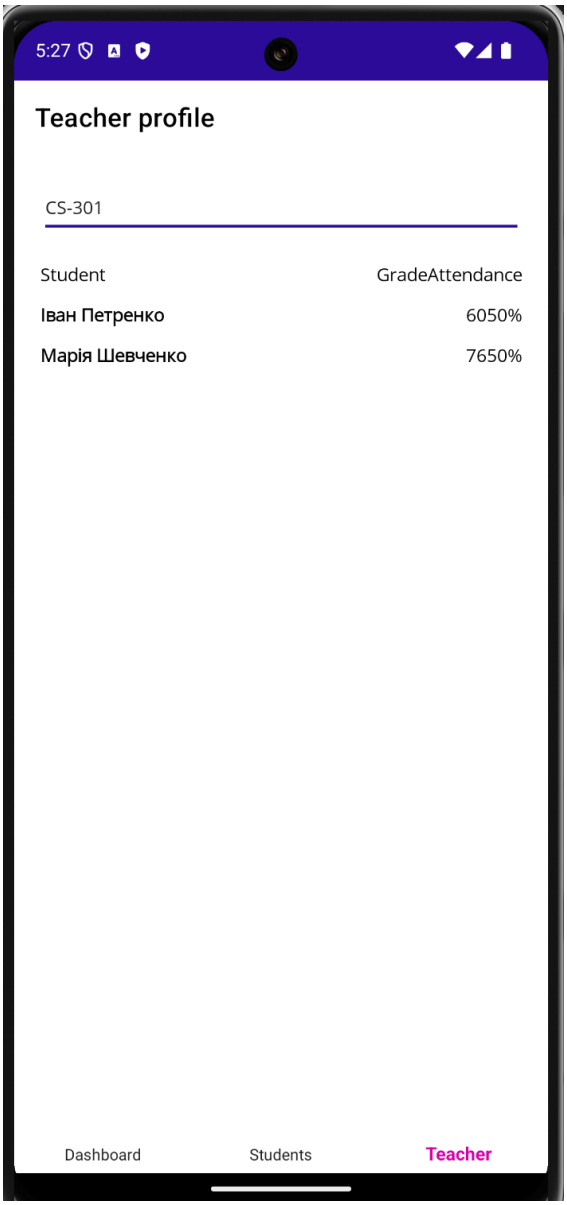

Рисунок 2.4. Сторінка Teacher profile

3. **Сторінка управління студентами:**
    - Можливість переглядати список учасників.
    - Відображення основної контактної інформації студентів.

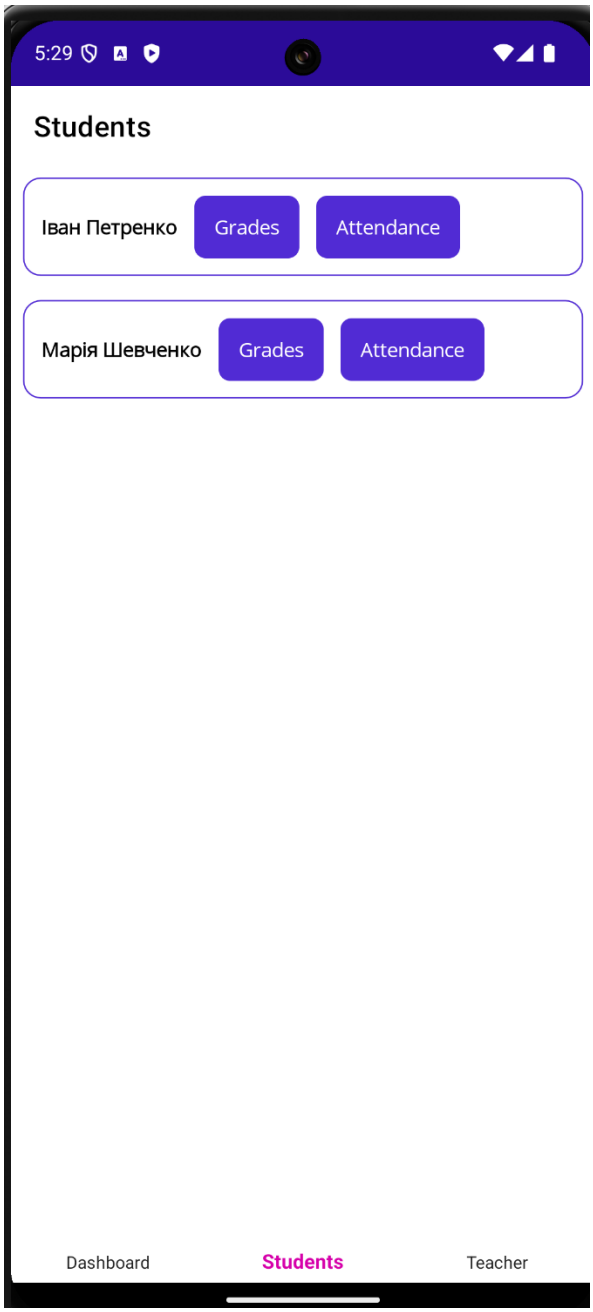

Рисунок 2.5. Сторінка Students

4. **Сторінка внесення оцінок:**
    - Викладачі мають змогу виставляти оцінки студентам по кожному заняттю чи модулю.
    - Форма введення даних з автоматичною валідацією (наприклад, перевірка діапазону балів).

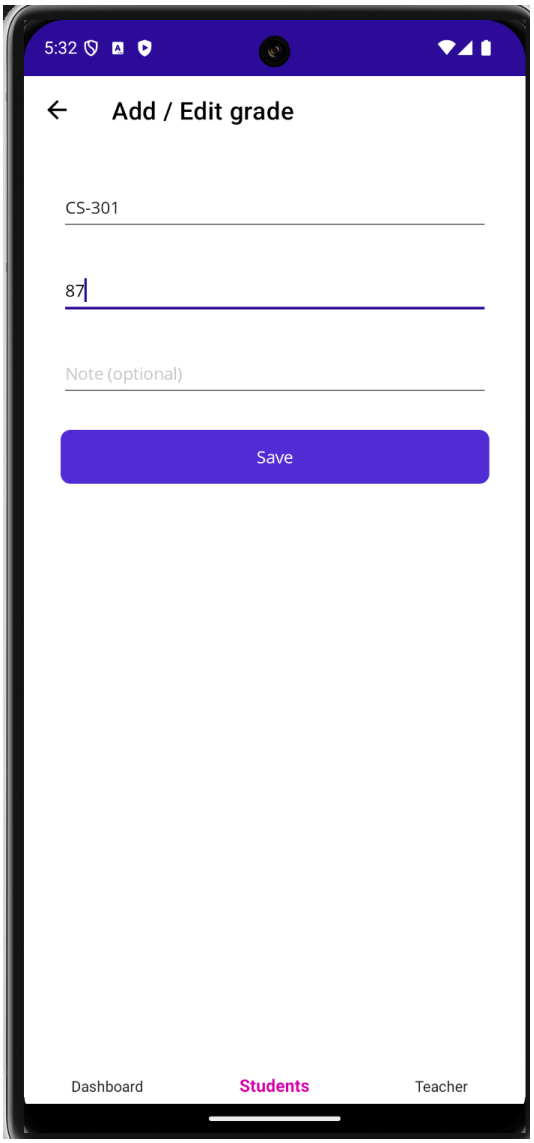

Рисунок 2.6. Сторінка Add/Edit grade

5. **Сторінка перегляду результатів (для студентів):**
    - Студенти бачать свої оцінки по всіх курсах, а також середній бал по кожному курсу.
    - Є простий візуальний індикатор: наприклад, зелений колір для високих результатів, жовтий — для середніх, червоний — для низьких.

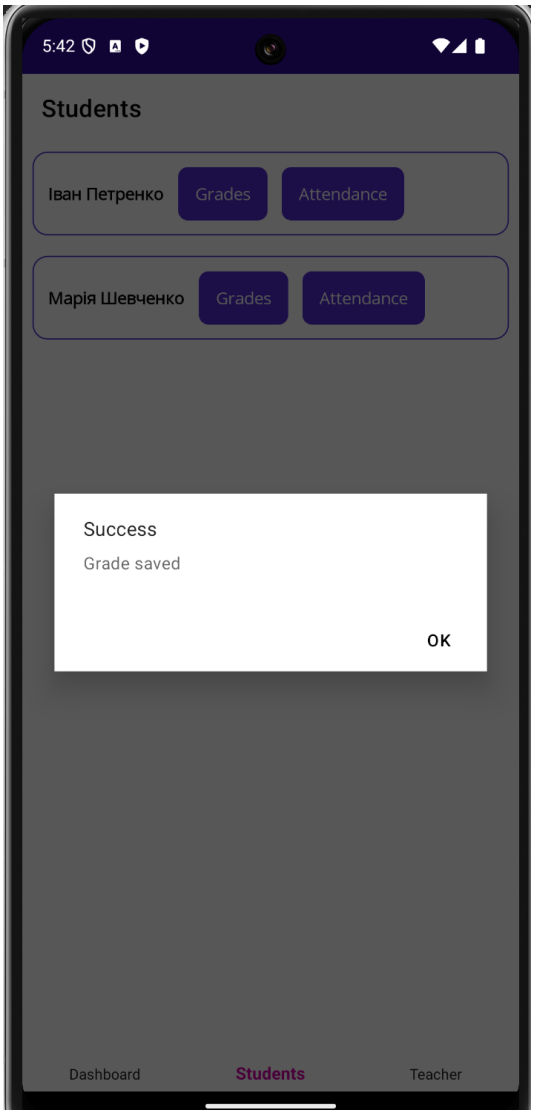

Рисунок 2.7. Повідомлення про успішно поставлену оцінку

Приклади реалізації:

- Елементи інтерфейсу побудовані на основі компонентів .NET MAUI, що дозволяє забезпечити однаковий вигляд на різних платформах (Windows, Android, iOS).
- Застосовано шаблони (templates) для списків, форм, повідомлень, що спрощує користувацький досвід.

Особливості дизайну:

- Застосовано принципи мінімалізму: прості кольорові схеми, мінімум відволікаючих деталей, акцент на ключових елементах (наприклад, кнопках, повідомленнях).
- Реалізовано адаптивність: інтерфейс коректно масштабується під розміри екрану різних пристроїв.
- Використано іконки для швидкого сприйняття дій (наприклад, значки редагування, видалення, додавання).

Важливі моменти:

- На етапі розробки проводилися консультації з викладачами для врахування їхніх побажань щодо вигляду та розташування елементів інтерфейсу.
- У майбутньому планується впровадження темного режиму, а також розширення налаштувань персоналізації інтерфейсу.

Таким чином, інтерфейс застосунку став важливим інструментом, який забезпечує ефективну роботу користувачів, мінімізує навчання при першому використанні та дозволяє швидко виконувати основні завдання без зайвих ускладнень.

2.5 Огляд технологічних інструментів та платформ

Розробка застосунку для автоматизованого обліку успішності студентів потребувала ретельного вибору технологічного стеку, який би відповідав вимогам проекту щодо надійності, продуктивності, кросплатформенності та можливості подальшого розширення. У цьому розділі розглянуто основні інструменти та платформи, використані під час реалізації.

.NET MAUI (Multi-platform App UI): .NET MAUI є сучасним кросплатформенним фреймворком, який дозволяє створювати застосунки для Windows, macOS, Android та iOS з єдиною базою коду. Завдяки цьому розробка значно пришвидшується, а витрати на підтримку продукту зменшуються. Серед його переваг:

- єдина мова програмування (C#) для фронтенду та бекенду;

- інтеграція з Visual Studio та іншими інструментами Microsoft;
- підтримка розширень та сторонніх бібліотек.

Мова програмування C#: C# є основною мовою розробки застосунку, що забезпечує надійність, високу продуктивність, об’єктно-орієнтоване програмування та хорошу інтеграцію з екосистемою .NET. Мова дозволяє використовувати багатий набір вбудованих бібліотек, спрощує роботу з базами даних, обробку подій та реалізацію багатопотокових завдань.

Entity Framework Core: Це ORM (Object-Relational Mapping) бібліотека, яка значно спрощує роботу з базами даних, дозволяючи розробникам використовувати об’єктно-орієнтований підхід замість написання сирих SQL-запитів. Основні переваги:

- автоматична генерація запитів;
- підтримка міграцій бази даних;
- гнучкість при роботі з різними СУБД (у нашому випадку SQLite).

SQLite: Для локального зберігання даних використовується SQLite — легка вбудована реляційна база даних, яка ідеально підходить для мобільних та десктопних застосунків. Вона не потребує окремого серверного середовища, забезпечує швидкий доступ до даних, дозволяє легко створювати резервні копії та переносити базу між пристроями.

Інструменти розробки:

- **Visual Studio 2022:** Основне середовище розробки, що надає зручні засоби для написання коду, налагодження, тестування та розгортання застосунку.
- **Git:** Система контролю версій, яка дозволила ефективно управляти змінами у проекті, працювати над кількома гілками одночасно та відстежувати історію розробки.
- **NuGet:** Менеджер пакетів для .NET, через який були підключені сторонні бібліотеки для додаткового функціоналу (наприклад, для валідації даних або роботи з інтерфейсами).

Додаткові бібліотеки та ресурси: Під час розробки застосунку використовувалися бібліотеки для реалізації анімацій, покращення зовнішнього вигляду інтерфейсу, спрощення роботи з формами та повідомленнями. Також застосовувались ресурси офіційної документації Microsoft, open-source приклади та навчальні матеріали для пришвидшення процесу розробки.

Вибір зазначених технологій був обґрунтований потребами проекту: забезпечити стабільну роботу застосунку на різних платформах, мінімізувати витрати на розробку, отримати можливість швидкого масштабування та розширення функціоналу в майбутньому.

2.6 Оцінка впливу системи на навчальний процес

Впровадження автоматизованої системи обліку та аналізу успішності студентів має значний потенціал для покращення організації навчального процесу. Розроблений застосунок дозволяє не лише зменшити адміністративне навантаження на викладачів, але й підвищити загальну ефективність навчання завдяки більшій прозорості даних, швидкому доступу до оцінок і можливості своєчасного реагування на проблеми.

Вплив на викладачів:

- **Зменшення рутинних завдань:** Викладачі можуть швидко вносити оцінки, формувати списки студентів, оновлювати інформацію про курси, не витрачаючи час на заповнення паперових журналів.
- **Підвищення точності:** Завдяки автоматичній обробці даних мінімізується ризик помилок, пов’язаних з ручним введенням, а також полегшується відстеження змін та корекцій.
- **Аналітика:** Викладачі отримують змогу швидко аналізувати рівень успішності групи, бачити слабкі місця у засвоєнні матеріалу та приймати обґрунтовані рішення щодо необхідності додаткових занять або змін у викладанні.

Вплив на студентів:

- **Прозорість:** Студенти отримують доступ до своїх оцінок у зручному форматі, що дозволяє їм самостійно відстежувати прогрес і своєчасно реагувати на невдачі.
- **Мотивація:** Постійний доступ до результатів допомагає студентам краще розуміти свої сильні та слабкі сторони, підвищує мотивацію до навчання та сприяє відповідальності за власні результати.

Вплив на навчальний заклад:

- **Покращення якості обліку:** Навчальні заклади отримують сучасний інструмент для управління даними, що спрощує звітність і дозволяє легше контролювати навчальний процес.
- **Основи для розвитку:** Створення такої системи формує платформу для подальшого впровадження розширеного функціоналу — наприклад, модулів для аналітики, чат-ботів, генерації звітів тощо.

Практична цінність системи:

- Застосунок відповідає реальним потребам викладачів та студентів, оскільки був розроблений з урахуванням консультацій з кінцевими користувачами.
- Система дозволяє зменшити витрати часу на адміністративні завдання, підвищує загальну організованість навчального процесу та створює умови для якіснішого аналізу результатів.

Таким чином, впровадження розробленої системи має позитивний вплив на всі рівні освітнього процесу — від окремих студентів до адміністрації навчального закладу. У наступному розділі буде розглянуто технологічні аспекти реалізації системи, включаючи деталі вибору платформ, мови програмування, структури бази даних та безпеки.

РОЗДІЛ 3. ТЕХНОЛОГІЧНІ АСПЕКТИ РОЗРОБКИ СИСТЕМИ

3.1 Вибір технологій

Вибір правильних технологій є фундаментальним етапом при розробці будь-якого програмного продукту, оскільки саме від нього залежить продуктивність, масштабованість, легкість супроводу, адаптивність до змінних вимог і безпека системи. У випадку нашої системи автоматизованого обліку успішності студентів ми обирали технології з огляду на такі основні критерії: кросплатформеність, стабільність, швидкість розробки, доступність документації та прикладів, наявність активної спільноти та перспективи розвитку в майбутньому.

3.1.1 Основна мова програмування: C#

Мова програмування C# є одним із найпотужніших інструментів для розробки сучасних додатків, особливо у зв'язці з платформою .NET. Її основними перевагами є об’єктно-орієнтований підхід, зручний синтаксис, висока продуктивність, широке використання у промисловості та багата екосистема бібліотек. Важливим є також те, що C# активно розвивається, має регулярні оновлення та підтримується Microsoft, що гарантує довготривалу стабільність проектів на його основі.

У нашій системі мова C# дозволила ефективно реалізувати бізнес-логіку, роботу з базою даних, організацію взаємодії між компонентами, а також забезпечити високий рівень безпеки. Завдяки багатій стандартній бібліотеці розробка типових завдань, таких як робота з файлами, мережевими запитами чи обробкою даних, значно спрощується.

3.1.2 Фреймворк: .NET MAUI

.NET MAUI (Multi-platform App UI) є новим кроком у розвитку кросплатформенних рішень Microsoft, що прийшов на зміну Xamarin.Forms. Цей фреймворк дозволяє писати єдиний код, який працюватиме на Windows, Android, iOS і macOS, що значно скорочує час розробки та дозволяє підтримувати всі платформи одночасно [8].

Серед технічних переваг .NET MAUI варто виділити:

- **Єдина кодова база.** Можливість використовувати один проект для кількох платформ без дублювання коду.
- **Потужний інструментарій.** Інтеграція з Visual Studio забезпечує зручне середовище для розробки, налагодження та тестування.
- **Гнучкий UI.** Підтримка кастомізації інтерфейсу під специфіку кожної платформи.
- **Висока продуктивність.** Завдяки оновленим механізмам рендерингу додатки працюють швидше, ніж попередні рішення на Xamarin.

У нашому проекті .NET MAUI став центральною платформою для створення клієнтської частини, яка є одночасно адаптивною, сучасною та продуктивною.

3.1.3 Фреймворк: ASP.NET Core

Для проектування серверної частини системи було обрано ASP.NET Core — відкритий, кросплатформенний фреймворк для створення веб-застосунків, REST API та мікросервісів. Його переваги включають високу продуктивність, модульну архітектуру, підтримку сучасних стандартів безпеки, гнучкість налаштувань та масштабованість. ASP.NET Core забезпечує легку інтеграцію з базами даних, зовнішніми сервісами та клієнтськими застосунками. Хоча у першій версії нашого

продукту серверна частина ще не впроваджена, архітектура вже спроєктована таким чином, щоб у майбутньому підтримувати розширення до повноцінного клієнт-серверного рішення з використанням ASP.NET Core [9].

3.1.4 Інструменти та технології

Під час розробки застосовувалися такі ключові інструменти та технології:

- **Visual Studio Code.** Основне середовище розробки, що забезпечує повний цикл робіт: написання коду, налагодження, тестування, профілювання та розгортання [10].
- **Git.** Система контролю версій, яка дозволила ефективно управляти історією змін, працювати над кількома гілками проекту, вести документацію змін та забезпечувати безпеку коду [11].
- **NuGet.** Менеджер пакетів для .NET, через який підключалися додаткові бібліотеки (наприклад, для роботи з валідацією форм, побудови графічного інтерфейсу, логування тощо) [12].
- **Entity Framework Core.** ORM-бібліотека, яка спрощує взаємодію з базою даних, дозволяючи працювати з даними у вигляді об’єктів замість написання сирих SQL-запитів. Це значно підвищує швидкість розробки, зменшує кількість помилок і забезпечує зручність при зміні структури даних [13].
- **SQLite.** База даних для локального зберігання інформації, яка не потребує окремого серверного середовища й забезпечує високу продуктивність у випадках, коли система використовується локально [14].
- **JWT (JSON Web Token).** JWT дозволяє передавати підтвердження прав доступу між клієнтом і сервером у вигляді захищеного токена. У нашому проекті цей механізм використовувався для того, щоб забезпечити розмежування прав користувачів, таких як студенти та викладачі, захищати маршрути API й залишати відкритими лише ті, що призначені для загального

доступу. Він дозволив зменшити навантаження на сервер, оскільки після автентифікації інформація про користувача передається в токені без потреби постійної перевірки в базі даних. Завдяки використанню підпису токенів секретним ключем гарантується, що дані є захищеними та не можуть бути підробленими, що підвищує загальний рівень безпеки всієї системи [15].

Додатково: використовувалися інструменти для перевірки якості коду, профілювання продуктивності, перевірки безпеки, а також спеціалізовані плагіни для роботи зі специфічними завданнями (наприклад, формування повідомлень користувачам, робота з формами вводу тощо).

Таким чином, вибір технологій був зумовлений бажанням створити масштабоване, продуктивне, зручне в супроводі рішення, яке відповідає сучасним вимогам та є готовим до майбутнього розширення функціоналу. У наступному пункті буде розглянуто деталі розробки бази даних, яка є фундаментом для зберігання всієї навчальної інформації.

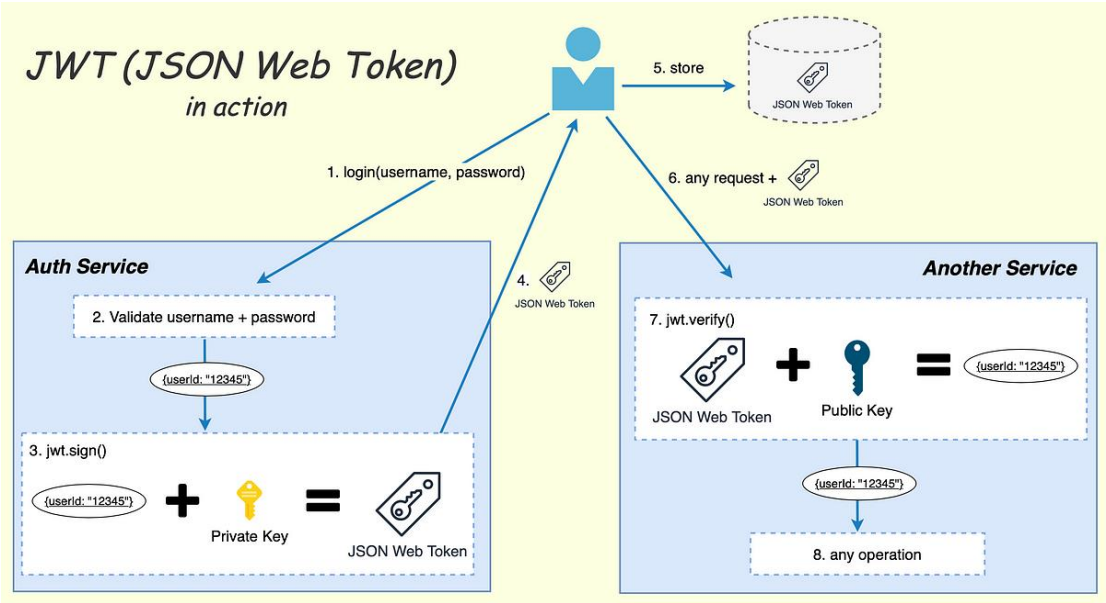

Рисунок 3. Схема роботи JWT

3.2 Розробка бази даних

3.2.1 Вибір системи управління базами даних (СУБД)

У нашому проєкті «Система обліку успішності студентів» було обрано SQLite як основну систему управління базами даних. Це рішення обумовлене простотою впровадження, можливістю роботи без окремого серверного середовища та високою швидкістю обробки запитів у локальних застосунках. SQLite дозволяє зберігати всі дані у вигляді одного файлу на пристрої, що ідеально підходить для проєктів невеликого масштабу, таких як мобільні або десктопні освітні застосунки. Крім того, використання SQLite забезпечує хорошу сумісність із .NET MAUI та Entity Framework Core, що дозволяє розробникам швидко інтегрувати її у застосунок.

3.2.2 Структура бази даних

Архітектура бази даних складається з таких основних сутностей:

- **Users (Користувачі):** зберігає інформацію про облікові записи користувачів, включаючи унікальний ідентифікатор, ім’я користувача, хешований пароль та роль (викладач або студент, у майбутньому — адміністратор).
- **Students (Студенти):** зберігає дані про студентів, такі як ПІБ, група, контактна інформація та зв’язок з обліковим записом користувача.
- **Courses (Курси):** включає інформацію про навчальні курси, їхню назву, опис, призначеного викладача.
- **Grades (Оцінки):** містить записи про оцінки студентів за конкретними курсами, зберігаючи значення оцінки, дату виставлення, а також посилання на студентів і курси.

Зв’язки між таблицями реалізовані на основі реляційної моделі з використанням зовнішніх ключів. Це дозволяє гарантувати цілісність даних, наприклад,

неможливість видалення курсу, якщо існують залежні оцінки, або забезпечення того, що кожна оцінка пов’язана лише з існуючим студентом і курсом.

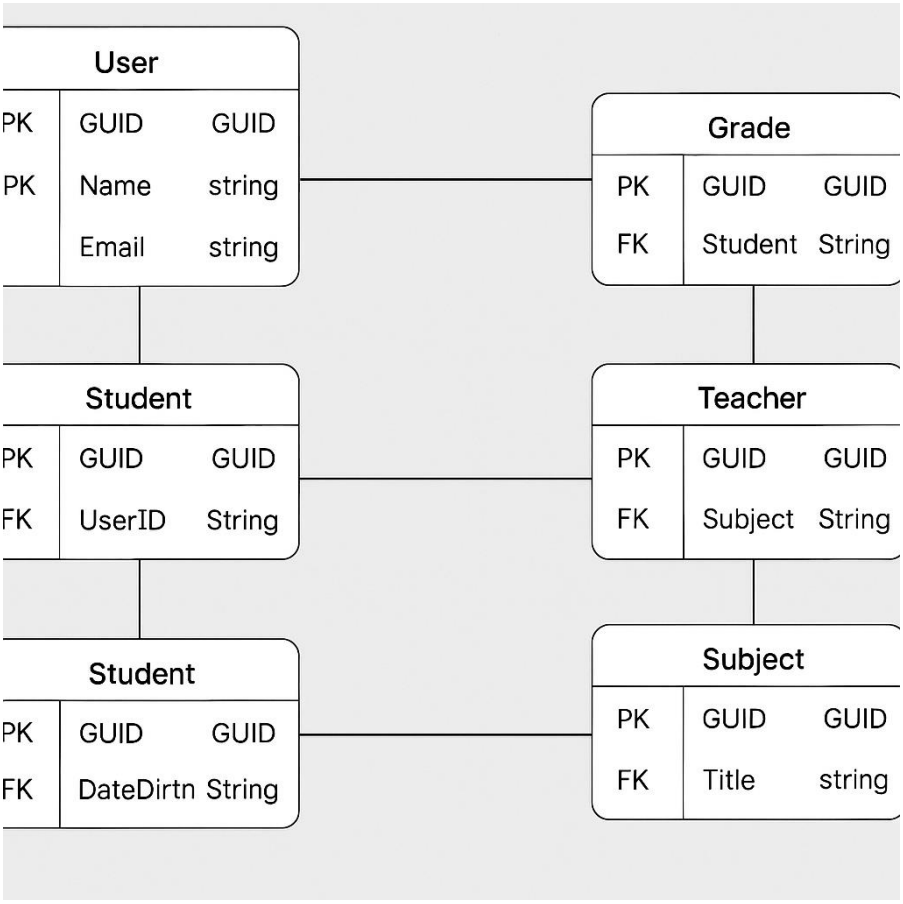

Рисунок 3.1. Схема зв'язків бази даних додатку

3.2.3 Безпека

Забезпечення безпеки даних у базі є одним із ключових завдань. У нашій системі реалізовано кілька рівнів захисту. По-перше, паролі користувачів зберігаються у вигляді хешів із використанням сучасних криптографічних алгоритмів, що запобігає можливості їх розшифрування навіть у разі отримання доступу до бази. По-друге, система контролю доступу будується на основі ролей, що визначає, які дії можуть виконувати різні користувачі — наприклад, студенти мають доступ лише до своїх даних, а викладачі — до курсів і студентів, за які вони відповідають.

Додатково передбачена інтеграція з JWT (JSON Web Token), що дозволяє безпечно передавати дані автентифікації між клієнтом і сервером та захищати API.

3.2.4 Резервне копіювання та відновлення

У рамках системи передбачено ручне резервне копіювання шляхом дублювання файлу бази даних SQLite. Це забезпечує просту можливість збереження копії даних на випадок збоїв або втрати інформації. У планах розвитку системи — впровадження автоматизованих механізмів резервного копіювання з інтеграцією хмарних сховищ, що дозволить налаштувати регулярне копіювання та швидке відновлення даних у разі аварійної ситуації. Також розглядається варіант використання інкрементного копіювання для зменшення обсягу даних, що зберігаються, і підвищення швидкості процесу.

3.2.5 Перспективи масштабування

Хоча SQLite є чудовим рішенням для локальних застосунків, у перспективі, коли система почне обслуговувати більше користувачів або потребуватиме синхронізації даних між різними пристроями, планується міграція на серверні СУБД, такі як PostgreSQL або SQL Server. Завдяки використанню Entity Framework Core, така міграція буде відносно легкою, оскільки бізнес-логіка вже абстрагована від конкретної СУБД, а запити сформульовані у вигляді LINQ-запитів.

Таким чином, розробка та архітектура бази даних у нашій системі забезпечують баланс між простотою, ефективністю, безпекою та готовністю до майбутнього розвитку та масштабування.

3.3 Реалізація клієнтської частини

Клієнтська частина системи реалізована за допомогою .NET MAUI. Вона складається з модульної структури, де кожен функціональний блок розділено на окремі компоненти. В інтерфейсі користувача передбачено спеціалізовані сторінки для ролей студентів та викладачів, а також передбачено адаптивність під різні розміри екранів.

Особлива увага приділялася тому, щоб дизайн інтерфейсу був інтуїтивно зрозумілим, із логічною навігацією між основними екранами: сторінкою входу, дашбордом, розділами курсів, сторінками перегляду оцінок та особистого кабінету. Розробка клієнтської частини передбачала гнучке управління станом, реактивність інтерфейсу та швидку реакцію на зміни даних.

Завдяки чіткому розділенню на компоненти й використанню сучасних патернів, таких як MVVM, реалізація клієнтської частини забезпечує легкість підтримки, масштабованість та можливість додавання нових функцій у майбутньому.

3.3.1 Структура компонентів

Клієнтська частина організована модульно: кожен компонент виконує окрему роль, що дозволяє розділити відповідальність і полегшити супровід. Основні компоненти групуються за функціоналом: екрани аутентифікації (вхід, реєстрація), панелі керування курсами, перегляд та редагування оцінок, сторінки студентського кабінету. У структурі проекту виділено окремі директорії для View, ViewModel та Model, що відповідає патерну MVVM. ViewModel компоненти займаються управлінням станом, обробкою подій користувача та взаємодією з сервісами, а View відповідають за відображення інформації. Така структура забезпечує модульність, що дозволяє швидко розширювати систему та впроваджувати нові функції, не порушуючи існуючу логіку.

3.3.2 Маршрутизація та навігація

Маршрутизація у клієнтській частині реалізована за допомогою вбудованих механізмів .NET MAUI, що забезпечує плавну навігацію між різними сторінками застосунку. Всі маршрути визначені у централізованому конфігураційному файлі, де кожен екран (сторінка) має свій унікальний маршрут. Це дозволяє легко перенаправляти користувача між сторінками, наприклад, після входу, після додавання оцінки чи після редагування курсу. Система підтримує як просту навігацію (наприклад, перехід на головну сторінку), так і передачу параметрів між екранами (наприклад, відкриття деталей конкретного курсу або перегляд оцінок конкретного студента).

3.3.3 Управління станом

Управління станом у клієнтській частині є важливою частиною архітектури, оскільки від нього залежить, наскільки швидко та правильно система реагує на зміни даних та взаємодію користувача. У нашій системі використовується підхід, заснований на патерні MVVM (Model-View-ViewModel), де ViewModel відповідає за зберігання стану сторінок, логіку взаємодії, а також за обробку подій. Кожна сторінка має свій ViewModel, що управляє її поведінкою: наприклад, оновлення списку курсів, збереження введених даних чи відображення повідомлень про помилки. Для зменшення дублювання логіки використовуються спільні сервіси, що надають доступ до даних (наприклад, списків студентів чи оцінок) усім ViewModel-компонентам. Завдяки такому підходу забезпечується чітке розділення логіки, підтримка реактивності інтерфейсу та простота розширення системи у майбутньому.

3.3.4 Взаємодія з бекендом

Взаємодія клієнтської частини з бекендом у нашій системі реалізована через спеціалізовані сервіси, які інкапсулюють логіку відправлення запитів до серверної частини та обробки відповідей. Для захищених запитів використовується механізм JWT (JSON Web Token), що дозволяє передавати маркери автентифікації разом із запитами та гарантувати, що доступ до захищених ресурсів отримають лише авторизовані користувачі. Основні операції взаємодії включають: отримання списку курсів, додавання нових оцінок, оновлення інформації про студентів, отримання зведеної статистики. Завдяки чітко виділеному шару сервісів клієнтська частина залишається ізольованою від деталей реалізації API, що спрощує її обслуговування та дає можливість у майбутньому масштабувати або змінювати серверну частину без масштабних змін на стороні клієнта.

3.4 Реалізація серверної частини

Серверна частина нашої системи реалізована на основі ASP.NET Core — потужного кросплатформенного фреймворку, який дозволяє створювати високопродуктивні вебсервіси, REST API та мікросервіси. Архітектура серверної частини побудована за модульним принципом: окремі контролери відповідають за обробку HTTP-запитів (наприклад, запити на отримання оцінок, додавання курсів, автентифікацію користувачів), сервіси інкапсулюють бізнес-логіку, а репозиторії працюють із базою даних через Entity Framework Core. Структура проекту включає також класи для налаштування та ініціалізації бази даних, обробки помилок, логування та конфігурації API. Завдяки використанню ASP.NET Core ми досягли високої масштабованості, безпеки та можливості інтеграції з іншими сервісами, що є важливим для сучасних інформаційних систем.

3.4.1 Огляд архітектури сервера

Серверна частина побудована за модульною архітектурою з чітким розділенням на контролери, сервіси, репозиторії та моделі. Контролери відповідають за обробку HTTP-запитів, сервіси — за бізнес-логіку (наприклад, розрахунок підсумкових оцінок, валідацію введених даних), а репозиторії забезпечують доступ до бази даних. Усі ці компоненти взаємодіють через чітко визначені інтерфейси, що забезпечує легкість тестування та розширення. Крім того, у проекті використовується механізм middleware для обробки помилок, логування запитів та забезпечення безпеки.

3.4.2 Конфігурація та запуск

Конфігурація серверної частини здійснюється через файли конфігурації appsettings.json та appsettings.Development.json, де задаються параметри підключення до бази даних, налаштування логування, секретні ключі для JWT та інші параметри середовища. Запуск серверного застосунку починається з класу Program.cs, який створює веб-хост, налаштовує залежності, реєструє сервіси в контейнері інверсії управління та визначає маршрутизацію запитів.

3.4.3 Безпека та аутентифікація

Важливою частиною серверної частини є впровадження безпечної системи аутентифікації та авторизації. Для цього використовується стандарт JWT, що дозволяє після входу користувача видавати йому токен, який використовується в подальших запитах. Сервер перевіряє дійсність токена, а також визначає права доступу на основі ролі користувача (студент, викладач). Паролі зберігаються у хешованому вигляді з використанням стійких криптографічних алгоритмів, що забезпечує високий рівень захисту.

3.4.4 Інтеграція з базою даних

Інтеграція з базою даних реалізована за допомогою Entity Framework Core — сучасного ORM-рішення, яке дозволяє взаємодіяти з даними у вигляді об’єктів, автоматично генеруючи SQL-запити. Використання міграцій дозволяє легко оновлювати схему бази даних у процесі розвитку проекту, а використання LINQ-запитів забезпечує гнучкість та виразність при формуванні запитів.

3.4.5 Тестування та моніторинг

Для забезпечення надійності серверної частини проводиться юніт-тестування основних компонентів, зокрема сервісів та репозиторіїв. Крім того, система логування дозволяє відстежувати ключові події та помилки, що виникають під час роботи. У майбутньому планується впровадження автоматизованих інтеграційних тестів, а також використання систем моніторингу продуктивності для оцінки стану серверного оточення та швидкого реагування на проблеми.

3.5 Безпека даних

Захист даних є одним із найважливіших аспектів розробки системи, що працює з особистою інформацією студентів та викладачів. У нашому проекті безпека реалізована на декількох рівнях. На стороні сервера впроваджено механізм аутентифікації та авторизації за допомогою JWT (JSON Web Tokens), що дозволяє гарантувати, що тільки авторизовані користувачі мають доступ до захищених ресурсів. Крім того, всі паролі зберігаються у базі даних у хешованому вигляді, що виключає можливість їх прямого зчитування навіть у разі компрометації бази.

На рівні клієнтської частини реалізовано перевірку прав доступу, щоб інтерфейс динамічно адаптувався під роль користувача, не даючи доступу до функціоналу, який

не передбачено для його ролі. Додатково застосовується захищене шифрування HTTPS для всіх запитів між клієнтом і сервером, що запобігає перехопленню даних третіми сторонами.

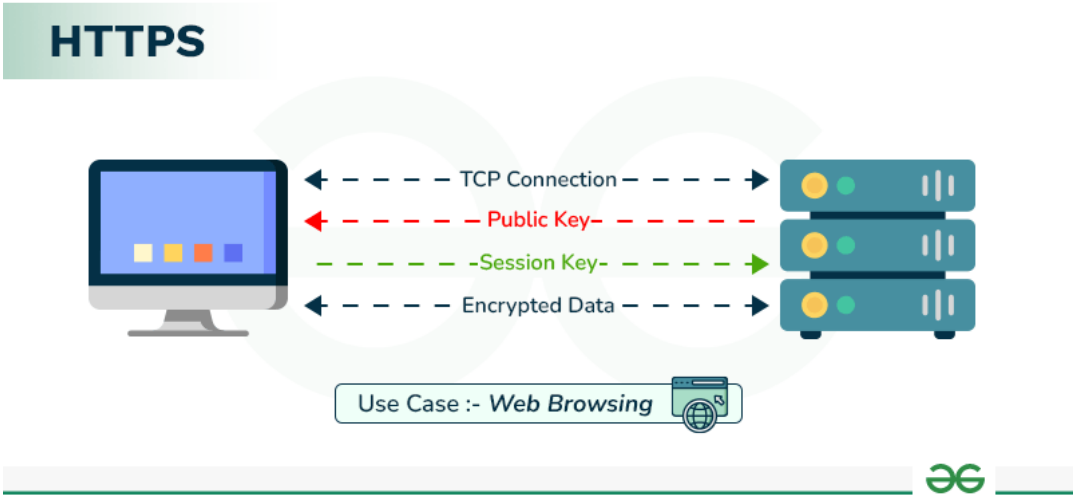

Рисунок 3.2. схема роботи HTTPS протоколу

Важливим елементом є також захист від типових веб-атак: SQL-ін’єкцій, XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery). Для цього використовуються стандартні засоби ASP.NET Core, включаючи автоматичну валідацію введених даних, захищені cookie та налаштування CORS.

У майбутньому передбачається розширення системи моніторингу, впровадження журналювання всіх критичних дій користувачів та автоматичних сповіщень адміністраторів про підозрілі активності. Завдяки такому комплексному підходу наша система забезпечує високий рівень захисту персональних даних і відповідає сучасним вимогам до безпеки інформаційних продуктів.

3.6 Тестування системи

Тестування є невід’ємною частиною життєвого циклу розробки програмного забезпечення, адже воно дозволяє перевірити працездатність усіх компонентів системи, виявити помилки на ранніх стадіях та забезпечити високу якість кінцевого продукту. У нашій системі тестування проводилося на декількох рівнях, охоплюючи як окремі модулі, так і інтеграцію компонентів.

Модульне тестування. Модульне тестування (unit testing) охоплює окремі сервіси, репозиторії та компоненти ViewModel. Було створено тести для перевірки правильності обробки даних, правильного формування запитів до бази даних, роботи логіки авторизації та валідації даних. Для написання тестів використовувалися бібліотеки MSTest та xUnit, що дозволяють автоматично запускати тести й перевіряти результати.

Інтеграційне тестування. Інтеграційні тести забезпечують перевірку взаємодії між різними частинами системи: клієнтською частиною, сервером та базою даних. Зокрема, тестувалися сценарії створення користувачів, додавання оцінок, оновлення інформації про курси, а також отримання звітів. Для проведення інтеграційного тестування застосовувалися середовища тестової бази даних і окремі налаштування серверної частини.

Тестування інтерфейсу користувача. Було проведено ручне тестування клієнтського інтерфейсу з метою перевірки зручності використання, коректності відображення даних, перевірки логіки навігації між сторінками. Окремо тестувалися реакція інтерфейсу на помилки користувача (наприклад, неправильний логін або некоректні дані у формах), а також відповідність дизайну початковим макетам.

Навантáжувальне тестування (планується). У майбутньому передбачається проведення навантáжувального тестування для оцінки продуктивності системи при одночасній роботі великої кількості користувачів. Це дозволить визначити, як система поводиться під високим навантаженням, де можуть виникати вузькі місця та які ресурси потребують оптимізації.

Таким чином, завдяки комплексному підходу до тестування вдалося досягти високої стабільності та надійності системи на момент завершення основного етапу розробки, а також підготувати її до розширення та масштабування в майбутньому.

РОЗДІЛ 4. ПЕРСПЕКТИВИ РОЗВИТКУ

Хоча розроблений застосунок уже надає широкий набір корисних функцій, існує низка напрямів, які дозволять підвищити його ефективність, масштабованість та зручність використання.

Одним із основних завдань є впровадження розширеної аналітики — побудова графіків, діаграм, автоматичних звітів, які допоможуть викладачам швидко оцінювати динаміку навчального процесу, виявляти проблемні місця та приймати обґрунтовані управлінські рішення.

Ще одним важливим етапом розвитку є реалізація ролі адміністратора, який зможе управляти правами доступу, створювати нові облікові записи, контролювати дії користувачів і забезпечувати централізоване адміністрування системи. Це дозволить підвищити рівень безпеки та управління системою.

Також планується перехід від локальної архітектури до хмарного зберігання даних, що дасть змогу синхронізувати інформацію між кількома пристроями, підтримувати одночасну роботу великої кількості користувачів та забезпечувати доступність даних з будь-якої точки.

Додатково передбачається впровадження чат-бота для текстової або голосової взаємодії користувачів із системою. Це спростить доступ до базових функцій, наприклад, перегляду оцінок чи розкладу занять, а також підвищить інтерактивність сервісу.

Нарешті, важливою задачею є проведення навантажувального тестування системи з метою оцінки її стабільності під великим навантаженням, виявлення вузьких місць у продуктивності й оптимізації ресурсів.

Завдяки впровадженню цих перспективних напрямів система зможе вийти на новий рівень якості, розширити сферу застосування та стати універсальним інструментом для організації навчального процесу в різних освітніх закладах.

ВИСНОВКИ

Проведена робота дозволила комплексно вирішити завдання створення системи автоматизованого обліку успішності студентів, що враховує ключові потреби навчального процесу. У ході виконання проекту було проведено ретельний аналіз предметної області, вивчено сучасні підходи до організації освітніх інформаційних систем, визначено сильні й слабкі сторони існуючих рішень. На основі цього аналізу було сформовано вимоги до майбутнього застосунку.

Проектна частина включала побудову архітектури, вибір оптимальних технологій, проектування бази даних, розробку клієнтської та серверної частин, а також впровадження механізмів безпеки, що відповідають сучасним стандартам. Реалізація включала повноцінний життєвий цикл: від концепції до створення прототипу, тестування та підготовки документації.

Основними результатами проекту стали:

- створення застосунку, що дозволяє викладачам управляти курсами, студентами та оцінками;
- впровадження безпечної автентифікації й авторизації на основі JWT;
- побудова гнучкої модульної архітектури, яка дозволяє швидко розширювати функціонал;
- підготовка клієнтського інтерфейсу, адаптованого до потреб студентів і викладачів;
- проведення комплексного тестування для перевірки працездатності системи.

Робота дозволила на практиці закріпити теоретичні знання зі сфери розробки інформаційних систем, поглибити навички роботи з сучасними фреймворками та технологіями (зокрема, .NET MAUI, ASP.NET Core, Entity Framework Core, SQLite), а також зрозуміти важливість ретельного планування, аналізу й тестування при створенні масштабованих програмних продуктів.

Система, створена в рамках цього проекту, готова до використання в навчальних закладах, де необхідно організувати облік успішності студентів, а її модульна структура дозволяє адаптувати продукт під конкретні потреби організації.

Досягнення мети, поставленої у вступі, повністю підтверджено. Була створена система, яка відповідає всім визначеним вимогам: забезпечує зручність використання, безпеку, масштабованість та модульність, дозволяючи автоматизувати облік оцінок і роботу з навчальними даними. Функціонал системи успішно протестовано, а результати тестування показали, що продукт готовий до практичного використання.

Практичне значення роботи полягає в тому, що впровадження розробленого програмного продукту дає змогу значно спростити рутинні процеси для викладачів (виставлення оцінок, ведення обліку присутності, перегляд успішності), а також надає студентам прозорий і швидкий доступ до інформації про власні результати. Система може використовуватися як окремо (локально), так і як основа для розширення на рівень навчального закладу. Її універсальність дозволяє впроваджувати її в різних освітніх установах, адаптуючи під конкретні запити та розширюючи функціонал (наприклад, додаючи модуль адміністративного управління або систему звітності). Розробка також має цінність як демонстрація практичного застосування сучасних технологій .NET MAUI, ASP.NET Core, Entity Framework Core та механізмів безпеки (JWT) у реальному проєкті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Електронні щоденники та журнали для закладів загальної середньої освіти Electronic Journal [Електронний ресурс]. – Режим доступу: https://e-journal.iea.gov.ua/#system (дата звернення: 25.05.2025)
2. Електронний щоденник Google Classroom [Електронний ресурс]. – Режим доступу: https://classroom.google.com/ (дата звернення: 25.05.2025)
3. Система обліку студентів та абітурієнтів ЄДЕБО [Електронний ресурс]. – Режим доступу: https://vstup.edbo.gov.ua/ (дата звернення: 25.05.2025)
4. Moodle Learning Management System [Електронний ресурс]. – Режим доступу: https://moodle.org/ (дата звернення: 25.05.2025)
5. Canvas LMS [Електронний ресурс]. – Режим доступу: https://www.instructure.com/canvas/ (дата звернення: 25.05.2025)
6. Edmodo Learning Platform [Електронний ресурс]. – Режим доступу: https://new.edmodo.com/ (дата звернення: 25.05.2025)
7. Freeman A. Pro ASP.NET Core MVC 2. Apress, 2017.
8. Smith J. P. Entity Framework Core in Action. Manning Publications, 2018.
9. SQLite Official Documentation [Електронний ресурс]. – Режим доступу: https://www.sqlite.org/docs.html (дата звернення: 25.05.2025)
10. .NET MAUI Documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/maui/ (дата звернення: 25.05.2025)
11. JWT.io: JSON Web Tokens Introduction [Електронний ресурс]. – Режим доступу: https://jwt.io/ (дата звернення: 25.05.2025)
12. Microsoft ASP.NET Core Documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/ (дата звернення: 25.05.2025)
13. Microsoft Entity Framework Core Documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 25.05.2025)

14. Хабр: П’ять простих кроків для розуміння JSON Web Tokens (JWT) [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/340146/’ (дата звернення: 25.05.2025)
15. Microsoft Visual Studio Code Documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/vscode/ (дата звернення: 25.05.2025)
16. Git Documentation [Електронний ресурс]. – Режим доступу: https://git-scm.com/doc (дата звернення: 25.05.2025)
17. Microsoft Azure Documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/azure/ (дата звернення: 25.05.2025)
18. REST API Design Handbook [Електронний ресурс]. – Режим доступу: https://restfulapi.net/ (дата звернення: 25.05.2025)
19. Design Patterns: Elements of Reusable Object-Oriented Software. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Addison-Wesley, 1994.
20. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Robert C. Martin. Prentice Hall, 2017.
21. Шилдт Г. C# 10 і .NET 6. Повне керівництво. Київ: Діалектика, 2022.
22. Material Design Guidelines [Електронний ресурс]. – Режим доступу: https://m3.material.io/ (дата звернення: 25.05.2025)
23. OWASP Foundation. Secure Coding Practices [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-secure-coding-practices/ (дата звернення: 25.05.2025)
24. Документація NuGet Package Manager [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/nuget/ (дата звернення: 25.05.2025)

ДОДАТОК А. ТЕХНІЧНЕ ЗАВДАННЯ

Загальний опис проєкту

- Назва проєкту: «Система обліку успішності студентів».
- Мета проєкту: створення інформаційної системи для автоматизації процесів обліку оцінок, управління курсами та студентами, а також забезпечення контролю за навчальним процесом.
- Короткий опис: система призначена для полегшення роботи викладачів завдяки автоматизації введення, зберігання й аналізу оцінок, надання студентам зручного інструмента для перегляду результатів навчання, а також створення основи для подальшого впровадження адміністративного модуля.

Вимоги до систем

- Функціональні вимоги:
    - Реєстрація та авторизація користувачів (викладачів, студентів, адміністрації).
    - Управління курсами та групами.
    - Введення, редагування та перегляд оцінок.
    - Перегляд підсумкової інформації про успішність студентів.
    - Захищена взаємодія між клієнтською та серверною частинами.
- Нефункціональні вимоги:
    - Високий рівень безпеки даних (хешування паролів, використання JWT).
    - Інтуїтивно зрозумілий та кросплатформенний інтерфейс.
    - Модульна архітектура, що підтримує масштабування.

Технологічний стек

- Мова програмування: C#.

- Система управління базами даних: SQLite.
- Фреймворки та бібліотеки: .NET MAUI, ASP.NET Core, Entity Framework Core.
- Інструменти та сервіси: JWT Token для автентифікації, Visual Studio для розробки, Git для контролю версій.

Архітектура системи

Система побудована за принципами клієнт-серверної архітектури. Клієнтська частина (інтерфейс) реалізована на .NET MAUI, що забезпечує кросплатформенність, серверна частина — на ASP.NET Core із REST API, який обробляє запити, виконує бізнес-логіку та взаємодіє з базою даних через Entity Framework Core. Для безпеки використовується механізм JWT, який дозволяє захищати API-запити та контролювати доступ за ролями.

Тестування та впровадження

- Стратегії тестування: модульне тестування окремих компонентів, інтеграційне тестування взаємодії між модулями, ручне тестування користувацького інтерфейсу, а також навантажувальне тестування для оцінки продуктивності.
- План впровадження: початкове пілотне впровадження в одному навчальному закладі, збір зворотного зв’язку від користувачів (викладачів і студентів), усунення виявлених недоліків, підготовка документації для адміністраторів і розробників, поступове масштабування системи та постійна підтримка.

ДОДАТОК Б. КОД ПРОГРАМИ

До цього додатку включено ключові фрагменти коду системи: реалізація основних контролерів, сервісів, моделей, налаштування безпеки (JWT), конфігурація Entity Framework Core для взаємодії з базою даних, а також приклади ViewModel і компонентів інтерфейсу на клієнтській стороні. Код структуровано за модулями для зручності ознайомлення.

Контролери (Controllers)

```
[ApiController]
[Route("api/[controller]")]
public class CoursesController : ControllerBase
{
    private readonly AppDbContext _db;
    public CoursesController(AppDbContext db) => _db = db;

    [HttpGet("{courseId:guid}/students-stats")]
    public async Task<ActionResult<IEnumerable<StudentCourseStatDto>>>
GetStudentStats(Guid courseId)
    {
        var enrollments = await _db.Enrollments
                              .Include(e => e.Student)
                              .Where(e => e.CourseId == courseId)
                              .ToListAsync();

        var studentIds = enrollments.Select(e => e.StudentId).ToList();
```

```
        var grades = await _db.Grades
                        .Where(g => g.EnrollmentCourseId == courseId &&
                                    studentIds.Contains(g.EnrollmentStudentId))
                        .GroupBy(g => g.EnrollmentStudentId)
                        .Select(g => new { StudentId = g.Key, Grade = g.Average(x =>
x.Value) })
                        .ToDictionaryAsync(g => g.StudentId, g => (double)g.Grade);

        var attendance = await _db.Attendances
                            .Where(a => a.CourseId == courseId &&
                                        studentIds.Contains(a.StudentId))
                            .GroupBy(a => a.StudentId)
                            .Select(g => new
                            {
                               StudentId = g.Key,
                               Pct = g.Any() ? g.Count(a => !a.IsExcused) / (double)g.Count()
: 0d
                            })
                            .ToDictionaryAsync(a => a.StudentId, a => a.Pct);

        var result = enrollments
           .OrderBy(e => e.Student.FullName)
           .Select(e => new StudentCourseStatDto(
              e.StudentId,
              e.Student.FullName,
              grades.TryGetValue(e.StudentId, out var g) ? g : 0d,
              attendance.TryGetValue(e.StudentId, out var p) ? p : 0d))
           .ToList();
```

```
        return Ok(result);
    }
}
```

Сервіси (Services)

```
public class JwtService : IJwtService
{
    private readonly string _secret;
    public JwtService(IConfiguration config) => _secret = config["Jwt:Key"];

    public string GenerateToken(string userId, string role)
    {
        var claims = new[]
        {
            new Claim(ClaimTypes.NameIdentifier, userId),
            new Claim(ClaimTypes.Role, role)
        };

        var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_secret));
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

        var token = new JwtSecurityToken(
            claims: claims,
            expires: DateTime.Now.AddHours(1),
            signingCredentials: creds);
```

```
        return new JwtSecurityTokenHandler().WriteToken(token);
    }
}
```

Моделі (Models)

```
public class Enrollment
{
    public Guid Id { get; set; }
    public Guid StudentId { get; set; }
    public Guid CourseId { get; set; }
    public Student Student { get; set; }
    public Course Course { get; set; }
}

public class Grade
{
    public Guid Id { get; set; }
    public Guid EnrollmentStudentId { get; set; }
    public Guid EnrollmentCourseId { get; set; }
    public decimal Value { get; set; }
}
```

Конфігурація Entity Framework Core

```
public class AppDbContext : DbContext
{
    public DbSet<Student> Students { get; set; }
```

```
    public DbSet<Course> Courses { get; set; }
    public DbSet<Enrollment> Enrollments { get; set; }
    public DbSet<Grade> Grades { get; set; }
    public DbSet<Attendance> Attendances { get; set; }

    public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
{ }
}

builder.Services.AddDbContext<AppDbContext>(o =>
    o.UseSqlite("Data Source=tracker.db3")
     .EnableSensitiveDataLogging()
     .LogTo(Console.WriteLine, LogLevel.Information));
```

ViewModel (Client Side)

```
public partial class TeacherProfileViewModel : BaseViewModel
{
    private readonly HttpClient _api;

    [ObservableProperty] ObservableCollection<CourseDto> courses = new();
    [ObservableProperty] ObservableCollection<StudentCourseStatDto> students =
new();
    [ObservableProperty] CourseDto? selectedCourse;

    public TeacherProfileViewModel(HttpClient api) => _api = api;
```

```
    [RelayCommand]
    public async Task InitAsync()
    {
        var courseList = await
_api.GetFromJsonAsync<List<CourseDto>>("api/courses");
        if (courseList != null) Courses = new
ObservableCollection<CourseDto>(courseList);
    }

    [RelayCommand]
    public async Task LoadStudentsAsync()
    {
        if (SelectedCourse == null) return;
        var stats = await _api.GetFromJsonAsync<List<StudentCourseStatDto>>(
            $"api/courses/{SelectedCourse.Id}/students-stats");
        if (stats != null) Students = new
ObservableCollection<StudentCourseStatDto>(stats);
    }
}
```

Ініціалізація MauiProgram.cs

```
public static MauiApp CreateMauiApp()
{
    var builder = MauiApp.CreateBuilder();
    builder
        .UseMauiApp<App>()
        .ConfigureFonts(fonts =>
```

```
        {
            fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
            fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
        });

    builder.Services.AddSingleton<HttpClient>();
    builder.Services.AddSingleton<TeacherProfileViewModel>();
    builder.Services.AddSingleton<TeacherProfilePage>();

    return builder.Build();
}
```

Схема авторизації (JWT-токенів)

1. Користувач надсилає логін і пароль на endpoint /api/auth/login.
2. Сервер перевіряє дані, і якщо вони правильні, генерує JWT-токен.
3. JWT-токен передається клієнту у відповіді.
4. Клієнт зберігає токен (наприклад, у localStorage) і додає його в Authorization-заголовок для кожного запиту.
5. Сервер перевіряє токен при кожному запиті, щоб підтвердити авторизацію користувача.

ДОДАТОК В. СЦЕНАРІЇ РОБОТИ В ДОДАТКУ

У даному додатку опишемо сценарії роботи нашого додатку.

Сценарій №1: Додавання оцінки студенту в профілі вчителя.

Крок 1: Для початку, входимо до профілю вчителя під username: «teacher@demo» і паролем: «Pass123!»

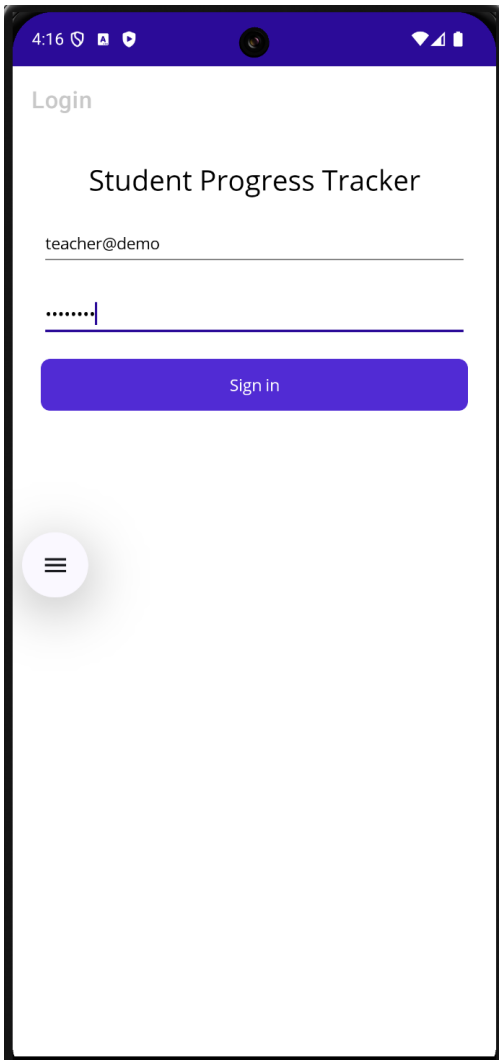

Рисунок 4. Сторінка Login

Крок 2: Після успішної авторизації, натискаємо на кнопку «Students» на панелі навігації.

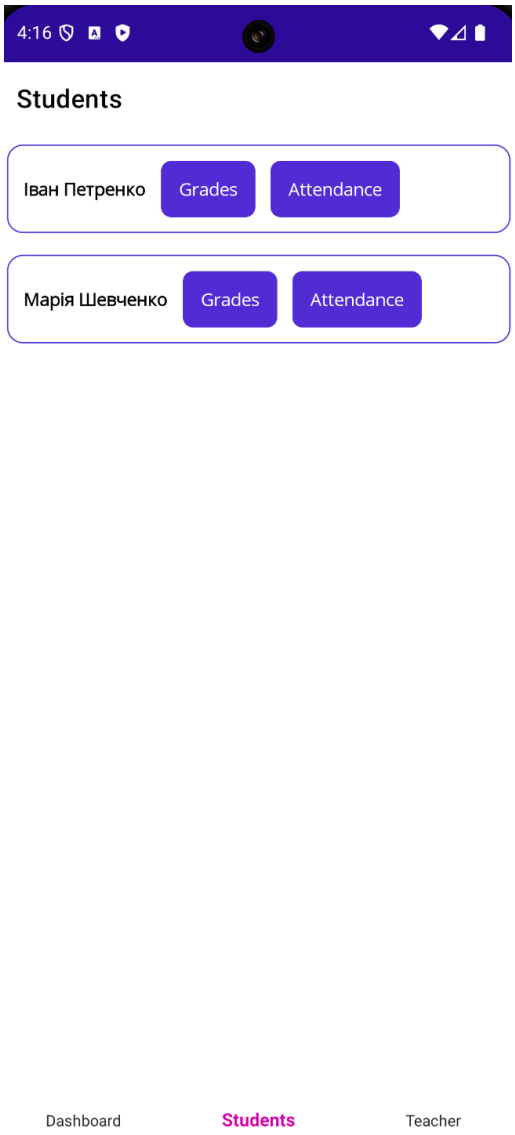

Рисунок 4.1. Сторінка Students

Крок 3: Натискаємо на кнопку «Grades» у студента Іван Петрук, відкриється сторінка «Grades».

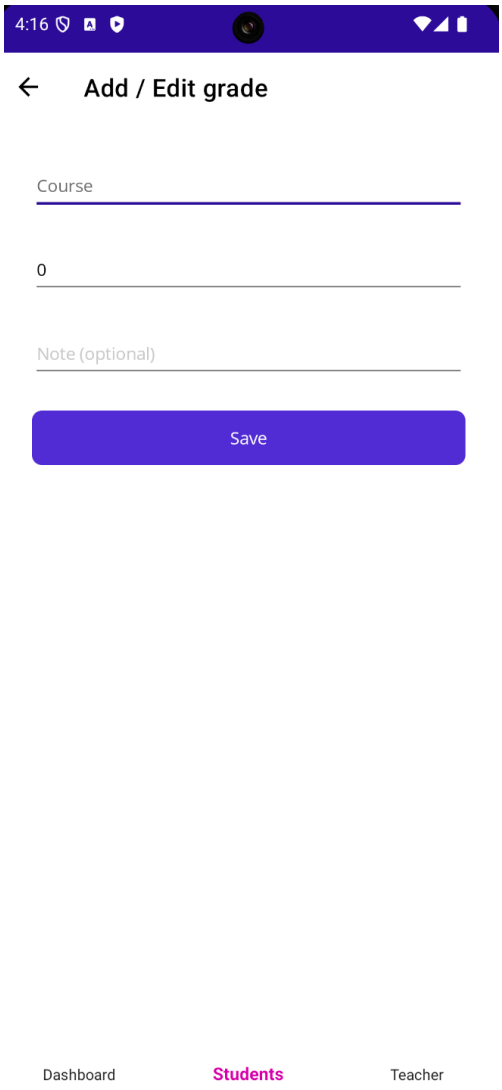

Рисунок 4.2. Сторінка Grades

Крок 4: Обираємо курс студента у picker-і, на скріні видно лише 3 курс, через те, що студент зареєстрований лише на ньому.

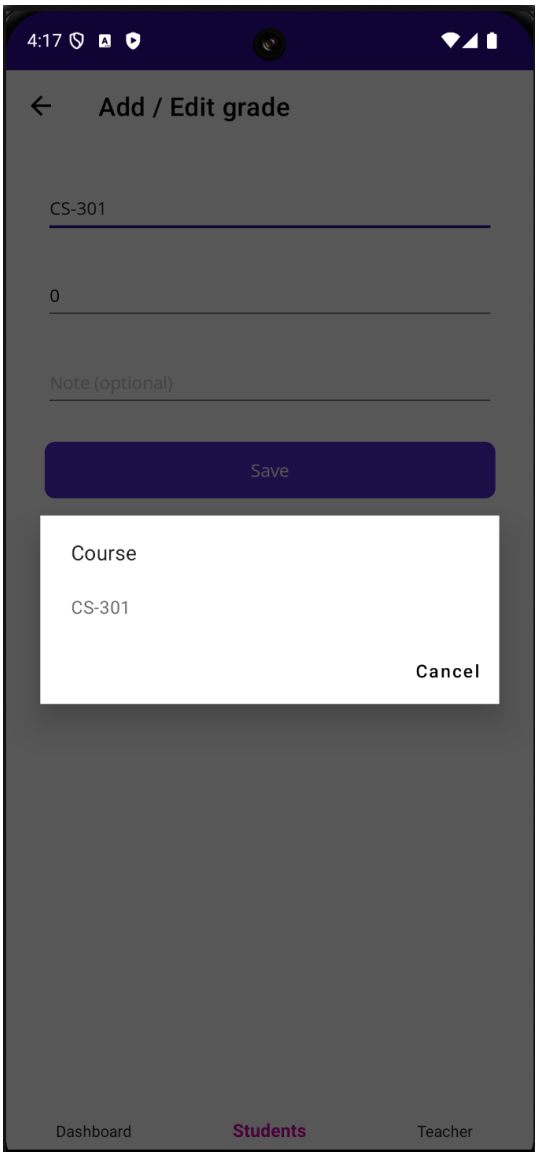

Рисунок 4.3. Course picker

Крок 5: Вписуємо нашу оцінку і натискаємо кнопку «Save» і після того бачимо alert, який говорить нам про успішне додавання оцінки.

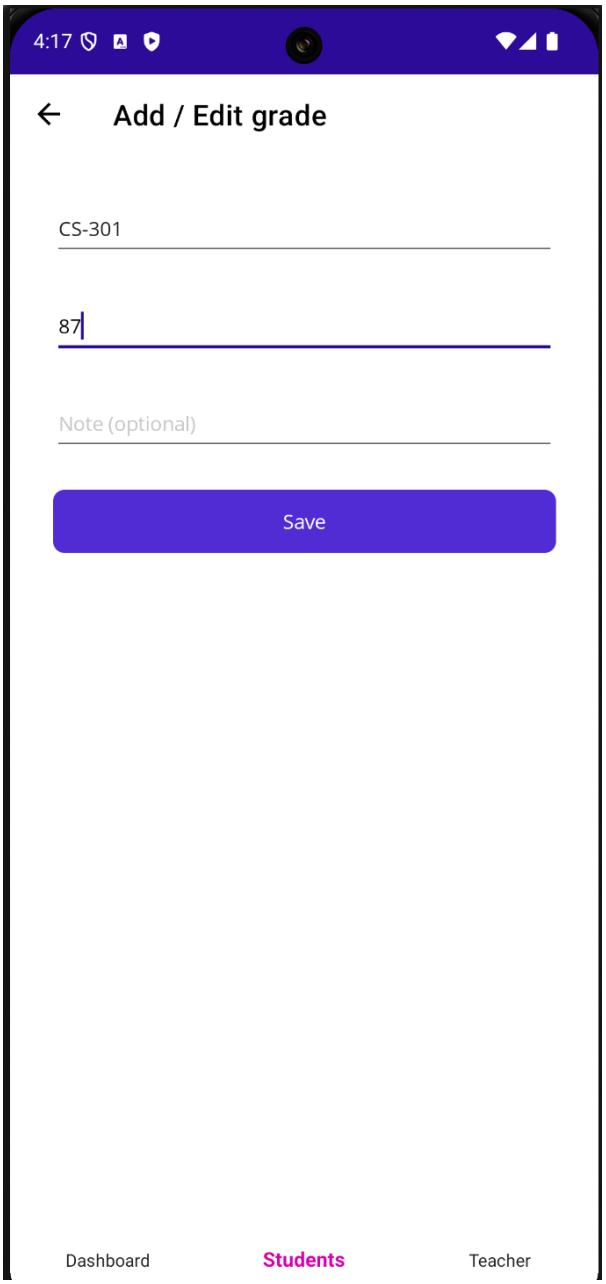

Рисунок 4.4. Виставлення оцінки

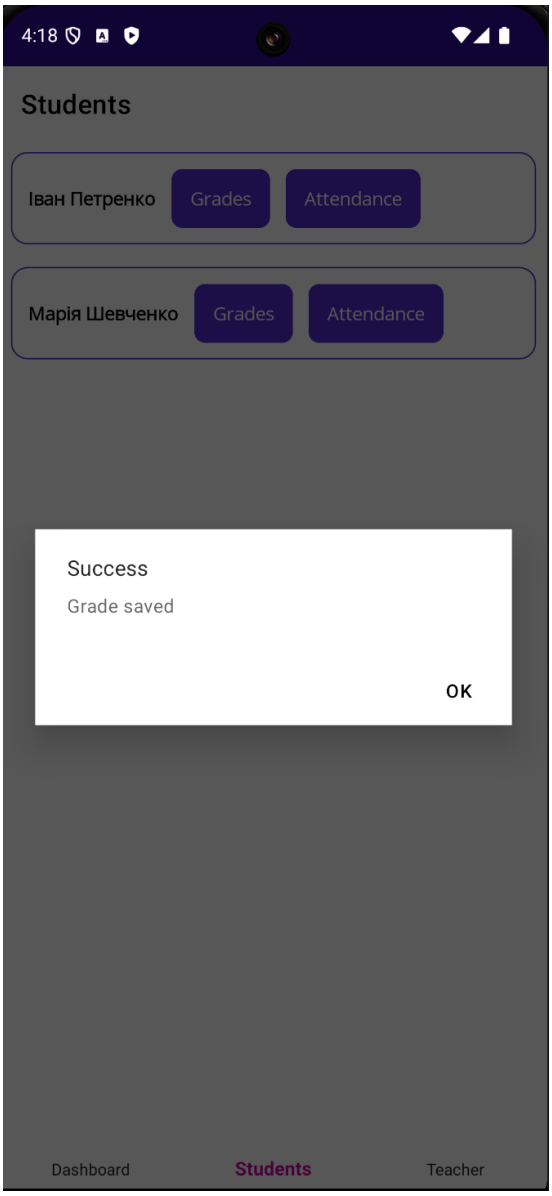

Рисунок 4.5. Alert успішності виствлення оцінки

Сценарій №2: Відмічання присутності студента.

Виконуємо **Крок 1** та **Крок 2** з попереднього сценарію. Після цього потрапляємо на сторінку «Attendance». На ній є можливість обрати перемикачом чи був присутній студент в встановлену дату чи ні. Для прикладу обираємо «присутній» і натискаємо кнопку «Save».

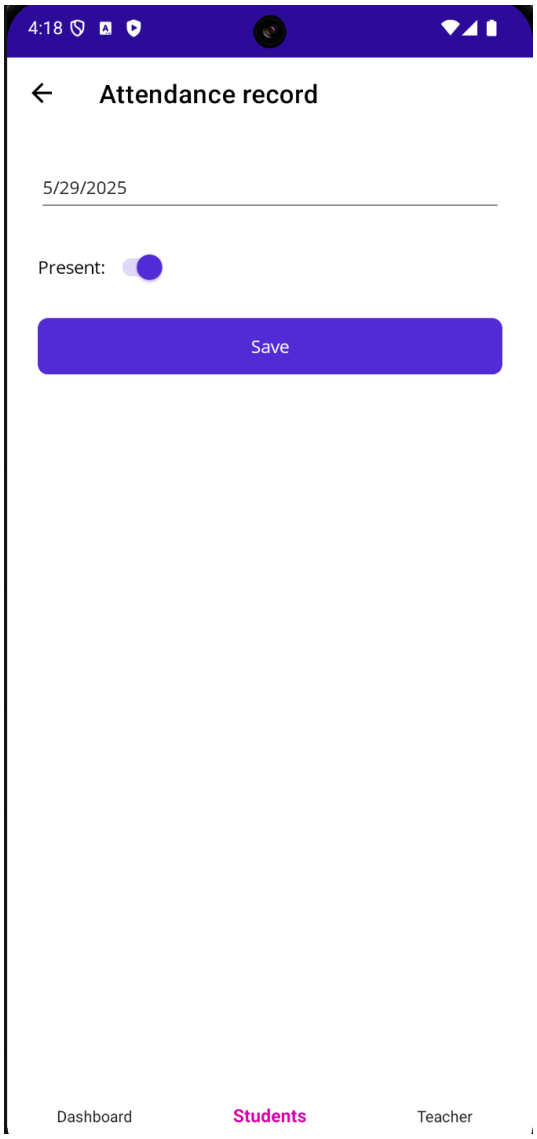

Рисунок 4.6. Сторінка «Attendance»

Сценарій 3: Перегляд таблиці студентів, відсортованими по курсах

Виконуємо **Крок 1** з попереднього сценарію.

Крок 2: Натискаємо кнопку «Teacher» на панелі навігації, відкриється сторінка «Teacher profile», на якій після вибору курсу, буде відображатись таблиця із студентами, їх оцінками і відвідуваністю.

Рисунок 4.7. Сторінка «Teacher profile»

Крок 3: Обираємо у picker-і «CS-301» і бачимо таблицю.

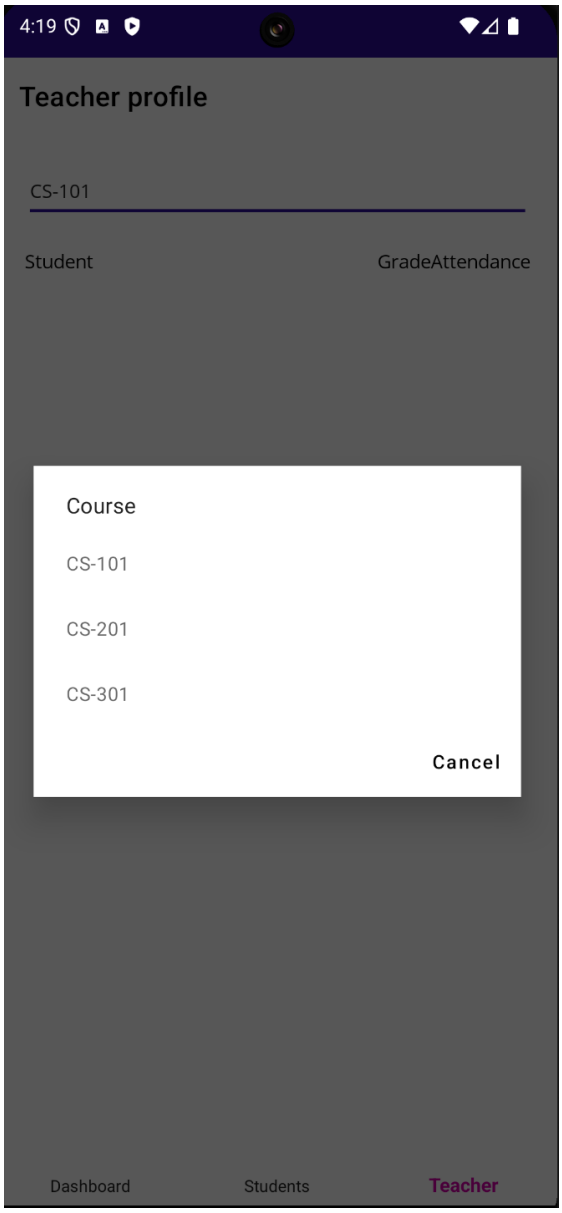

Рисунок 4.8. Course picker

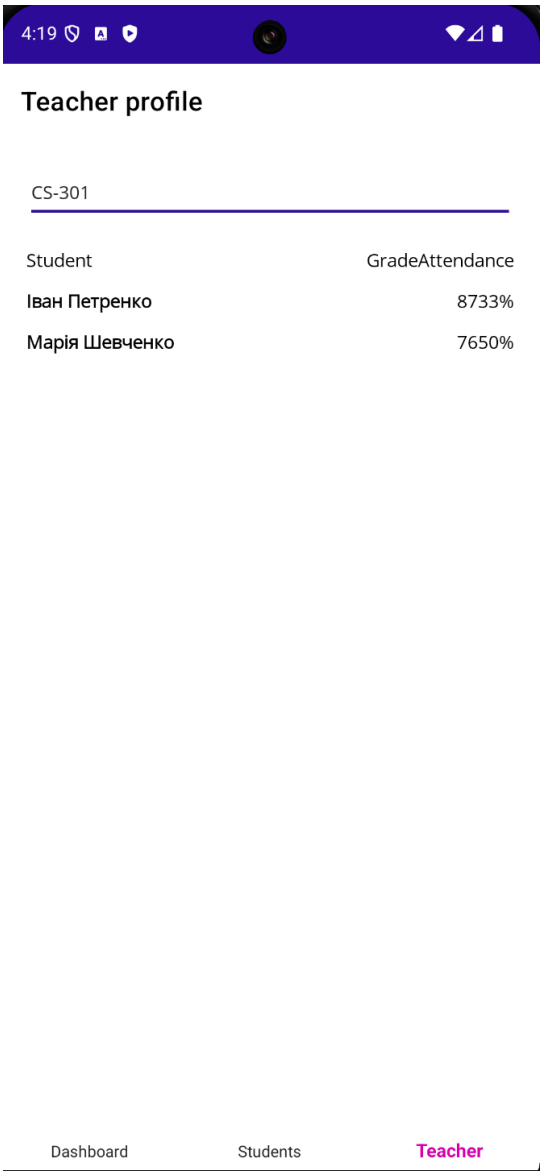

Рисунок 4.9. Таблиця для CS-301