МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

МОЛОДЧЕНКО ДМИТРО ВІКТОРОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р. техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

РОЗРОБКА КРОСПЛАТФОРМОВОГО ЗАСТОСУНКУ ДЛЯ ОРГАНІЗАЦІЇ НАВЧАЛЬНОЇ ДІЯЛЬНОСТІ НА ПРИКЛАДІ ЗДОБУВАЧІВ ВИЩОЇ ОСВІТИ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Надія ПОТАПОВА, доцент кафедри
інформаційних технологій,
к. е. н., доцент
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2026

АНОТАЦІЯ

Молодченко Д. В. Розробка кросплатформового застосунку для організації навчальної діяльності на прикладі здобувачів вищої освіти. Спеціальність 122 «Комп'ютерні науки». Освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2026.

У кваліфікаційній (бакалаврській) роботі представлено розробку застосунку для організації навчальної діяльності орієнтованого на здобувачів із можливістю групової комунікації, індивідуальним та спільним простором зберігання файлів, подій та ін. Розглянуто поширені проблеми здобувачів та шлях їх вирішення розробленою програмою. Розглянуто засіб взаємодії застосунку з базами даних, архітектуру кросплатформового застосунку, веб-сайту та API.

Ключові слова: C#, .NET MAUI, MS SQL Server, Entity Framework Core, Web API, розклад, файлова система, організація інформації.

85 ст., 10 рис., 41 джерело.

ABSTRACT

Molodchenko D. V. Development of a Cross-Platform Application for Organizing Educational Activities, Following the example of University Students. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2026.

The qualification paper examines development of application for organizing educational activities aimed at higher education students, with support for group communication and individual and shared file storage spaces, events and etc. The paper considers common problems faced by students and the ways these problems are solved by the developed software. It also examines the means of interaction between the application and databases, as well as the architecture of the cross-platform application, website, and API.

Keywords: C#, .NET MAUI, MS SQL Server, Entity Framework Core, Web API, schedule, file system, information organization.

ЗМІСТ

ВСТУП..... 4
РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ ВИКОРИСТАННЯ ІНСТРУМЕНТІВ ТА ПІДХОДІВ ПРИ РОЗРОБЦІ КРОСПЛАТФОРМОВОГО ЗАСТОСУНКУ..... 6
1.1. Кросплатформна розробка програмного забезпечення..... 6
1.2. Blazor Web App для вебверсії застосунку 11
1.3. Автентифікація, авторизація та контроль доступу 16
РОЗДІЛ 2. ПРОЄКТУВАННЯ ЗАСТОСУНКУ ОРГАНІЗАЦІЇ НАВЧАЛЬНОЇ ДІЯЛЬНОСТІ..... 22
2.1. Загальна концепція розробки кросплатформного застосунку..... 22
2.2. Проєктування ролей і прав доступу 28
2.3. Проєктування основних модулів застосунку..... 31
2.4. Проєктування групової взаємодії користувачів та організація даних .. 39
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ОРГАНІЗАЦІЇ НАВЧАЛЬНОЇ ДІЯЛЬНОСТІ..... 51
3.1. Структура програмного рішення 51
3.2. Реалізація автентифікації, авторизації та ролей користувачів..... 58
3.3. Реалізація файлового сховища..... 66
3.4. Реалізація вебверсії на Blazor Web App..... 72
3.5. Тестування та демонстрація роботи системи..... 76
ВИСНОВКИ..... 79
СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ 81
ДОДАТОК А..... 85

ВСТУП

Актуальність теми дослідження. У сучасних умовах студенти та викладачі покладаються на цифрові інструменти для організації навчального процесу. Розклад занять, навчальні матеріали, завдання з їх дедлайнами, групові повідомлення, особисті нотатки та файли – все це необхідні для навчання речі. Деякі застосунки можуть запропонувати одразу декілька з цих функціоналів, але число потрібних сервісів, які студенти мають використовувати, для задоволення потреби в перечисленому функціоналі залишається великим. Хоча використання великої кількості сервісів є поширеною практикою, перемикання між ними шкодить концентрації студентів під час виконання завдань. Не всі студенти вдаються до організації їх робочого місця і тому часто стикаються з ситуацією втрати інформації. Таке трапляється навіть з тими хто тримає їх робочий простір у порядку.

Актуальність теми бакалаврської роботи полягає у створенні єдиного програмного середовища для організації навчальної діяльності студента та академічної групи. Розділене зберігання інформації додає непотрібної складності до пошуку необхідної інформації: матеріали для виконання завдань, контроль їх дедлайнів, відстеження змін у розкладі та важливої інформації на рівні групи.

Мета роботи: розробка кросплатформового клієнт-серверного застосунку для організації навчального процесу студентів та академічних груп.

Завдання дослідження:

1. Дослідити теоретичні аспекти використання інструментів розробки кросплатформових застосунків;

2. Сформулювати вимоги до кросплатформної клієнт-серверної системи, визначити основні функціональні можливості;

3. Спроєктувати архітектуру та структуру даних системи, описати взаємодію між клієнтськими застосунками, серверним API, базою даних і

файловим сховищем;

4. Реалізувати основні програмні модулі системи, включаючи серверну частину, механізми автентифікації та авторизації, модулі розкладу;

5. Провести тестування розробленої системи, перевірити коректність роботи основних функцій, взаємодію клієнтів із сервером.

Об'єкт дослідження: процеси розробки кросплатформового застосунку для організації навчальної діяльності студентів та академічних груп.

Предмет дослідження: методи, підходи та програмні засоби створення кросплатформної системи з підтримкою групової взаємодії, файлового сховища та синхронізації.

Структура кваліфікаційної роботи: бакалаврська робота складається зі вступу, трьох розділів з висновками до кожного, списку використаних джерел та додатків.

Практичне значення результатів роботи полягає у налагодженні зв'язку між різними видами навчальної інформації. У випадку коли дані (конкретний предмет, завдання або події, а нотатки можуть бути пов'язані з підготовкою до лабораторної роботи чи практичного завдання) зберігаються окремо, користувач витрачає час для пошуку пов'язаних з завданням елементів та збору їх до купи. Дану проблематику вирішує інструмент, який зможе об'єднати розклад, матеріали предметів, завдання, нотатки, файли і події в програму, та містить функціонал для комунікації групи. Ця система дозволить впорядкувати навчальну інформацію, покращити роботу з файлами та забезпечити зручну взаємодію академічної групи.

Апробація результатів дослідження: представлення матеріалів на VII Всеукраїнській науково-практичній конференції здобувачів вищої освіти та молодих вчених «Прикладні інформаційні технології» (ПІТ-2026). Наведені у бакалаврські роботі дослідження пов'язані з фундаментальною науково-дослідною роботою «Розвиток методів комп'ютерно-математичного моделювання складних систем та процесів у сфері освіти та діяльності ІТ-підприємств» (№ держреєстрації 0126U003221, 2026-2031 рр.).

РОЗДІЛ 1
ТЕОРЕТИЧНІ АСПЕКТИ ВИКОРИСТАННЯ ІНСТРУМЕНТІВ ТА ПІДХОДІВ ПРИ РОЗРОБЦІ КРОСПЛАТФОРМОВОГО ЗАСТОСУНКУ

1.1. Кросплатформна розробка програмного забезпечення

Кросплатформна розробка [30] передбачає створення програм, які можуть працювати на кількох операційних системах або типах пристроїв без повного переписування коду для кожної платформи. Для студентського органайзера це є важливим, оскільки користувач повинен мати доступ до розкладу, завдань, файлів і нотаток зі смартфона, комп'ютера або браузера. Основною перевагою кросплатформного підходу є зменшення дублювання коду. Спільна бізнес-логіка, моделі даних, правила валідації, робота з API та синхронізація можуть використовуватися в різних клієнтських частинах системи. Це спрощує супровід проєкту, пришвидшує внесення змін і зменшує ризик різної поведінки застосунку на різних пристроях.

Під час розробки програмного забезпечення можна використовувати кілька підходів: native, web[3][14], hybrid і cross-platform native. Native-розробка забезпечує найкращу інтеграцію з платформою, але потребує окремої кодової бази для Android, Windows та інших систем. Web-підхід дає змогу працювати із застосунком через браузер без встановлення, однак має обмеження щодо доступу до можливостей пристрою[16]. Hybrid-підхід дозволяє запускати вебінтерфейс як встановлену програму, але може поступатися за продуктивністю та якістю взаємодії. Cross-platform native є компромісним варіантом, оскільки дозволяє повторно використовувати код і водночас створювати застосунки, наближені до native за поведінкою.

Для студентського органайзера доцільною є комбінована модель: Android/Windows клієнт реалізується за допомогою кросплатформного підходу, вебверсія створюється як окремий вебінтерфейс, а серверна частина надає єдине API для всіх клієнтів. Така архітектура дозволяє централізовано зберігати дані, контролювати права доступу, виконувати авторизацію та

синхронізувати інформацію між пристроями. Підтримка Android важлива для швидкого доступу до розкладу, оголошень і нагадувань. Windows-клієнт доцільний для роботи з файлами, нотатками, завданнями та навчальними матеріалами. Web-версія розширює доступність системи, оскільки дає змогу користуватися органайзером без встановлення додаткового програмного забезпечення.

Важливо, щоб інтерфейс системи був узгодженим, але не обов'язково повністю однаковим на всіх платформах. Мобільна версія має бути простою та зручною для швидких дій, настільна – придатною для роботи з більшими обсягами інформації, а вебверсія – адаптивною до різних розмірів екрана.

Отже, кросплатформна розробка є доцільним підходом для створення студентського органайзера. Вона забезпечує доступ до навчальної інформації з різних пристроїв, зменшує дублювання коду, спрощує підтримку системи та підвищує зручність її використання.

Платформа .NET[4] є сучасною екосистемою для розробки програмного забезпечення, яка дає змогу створювати серверні API, веб-застосунки, десктопні програми, мобільні клієнти, хмарні сервіси, фонові служби та бібліотеки спільної логіки. У межах розробки кросплатформної клієнт-серверної системи для організації навчального процесу студентів .NET виступає базовою технологічною платформою, оскільки дозволяє використовувати єдину мову програмування, спільні бібліотеки та узгоджений підхід до побудови різних частин системи.

Однією з головних переваг .NET є можливість створення повного програмного комплексу в межах однієї технологічної екосистеми. Серверна частина може бути реалізована за допомогою ASP.NET Core Web API[14], веб-версія – за допомогою Blazor Web App, а клієнтський Android/Windows-застосунок – за допомогою .NET MAUI. Такий підхід спрощує розробку, оскільки різні частини системи можуть використовувати спільні моделі даних, DTO-класи, правила валідації, принципи обробки помилок та асинхронні операції. Для студентського органайзера це важливо, оскільки система містить

взаємопов’язані модулі: розклад, завдання, предмети, файли, нотатки, події, групову взаємодію, Q&A-простір та сповіщення.

Мова C# є основною мовою програмування для більшості сучасних .NET-застосунків. Вона поєднує об’єктно-орієнтований, компонентний, узагальнений та асинхронний підходи до програмування. C# підтримує класи, інтерфейси, структури, записи, узагальнення, делегати, події, лямбда-вирази, атрибути, pattern matching, nullable reference types та інші можливості, які роблять її зручною для створення складних прикладних систем. У розроблюваній системі C# може використовуватися для опису доменних сутностей, сервісів бізнес-логіки, контролерів API, клієнтських сервісів, ViewModel-класів, компонентів веб-інтерфейсу та тестів.

Важливою особливістю C# є сувора типізація. Вона дозволяє виявляти значну частину помилок ще на етапі компіляції, а не під час виконання програми[19]. Для клієнт-серверної системи це має практичне значення, оскільки обмін даними між клієнтом і сервером повинен бути передбачуваним. Наприклад, об’єкти користувача, навчальної групи, предмету, завдання, файлу або події можуть бути описані окремими класами з чітко визначеними властивостями.

У межах розроблюваної системи значну роль відіграє асинхронне програмування. Клієнтський застосунок постійно взаємодіє із сервером: отримує розклад, завдання, список предметів, файли, повідомлення, події та сповіщення. Серверна частина виконує запити до бази даних, працює з файловим сховищем, перевіряє права доступу та формує відповіді для клієнтів. Якби ці операції виконувалися синхронно, це могло б призвести до блокування потоків і погіршення швидкодії. Асинхронна модель C# базується на ключових словах async та await, а також на типах Task і Task<T>.

Асинхронність особливо важлива для операцій введення-виведення: HTTP-запитів, звернень до бази даних, читання та запису файлів, завантаження вкладень і синхронізації інформації між пристроями. Наприклад, під час відкриття сторінки предмету застосунок може асинхронно

отримати інформацію про сам предмет, список завдань, файли, нотатки та пов’язані події. При цьому інтерфейс користувача не повинен зависати, а сервер має ефективно обробляти одночасні запити від багатьох користувачів.

Для роботи з колекціями даних у C# широко використовується LINQ. LINQ дозволяє виконувати фільтрацію, сортування, групування, проєкцію та об’єднання даних у декларативному стилі. Особливо важливим є використання LINQ разом з Entity Framework Core, де запити до сутностей можуть перетворюватися на SQL-запити до бази даних.

Застосування .NET і C# позитивно впливає на архітектуру проєкту. Рішення можна поділити на кілька окремих проєктів: доменний шар, прикладний шар, інфраструктурний шар, серверний API, клієнтський MAUI-застосунок, веб-інтерфейс і тести. Така структура дозволяє відокремити бізнес-логіку від деталей збереження даних, інтерфейсу користувача та конкретних технологій доступу до зовнішніх ресурсів.

.NET MAUI є одним з основних інструментів для реалізації клієнтської частини розроблюваної системи, оскільки дозволяє створювати кросплатформні застосунки з використанням C# та спільної кодової бази. У межах бакалаврської роботи .NET MAUI доцільно використовувати для створення Android- та Windows-клієнта студентського органайзера.

Офіційна документація Microsoft визначає .NET MAUI як кросплатформний UI-фреймворк для створення нативних мобільних і десктопних застосунків на C# та XAML[29]. За допомогою .NET MAUI можна створювати застосунки для Android, iOS, macOS і Windows з однієї спільної кодової бази. Даний фреймворк використовує уже наявні бібліотеки для різних платформ через рівень абстракції (рис. 1.1). Для студентського органайзера це важливо, оскільки користувач може працювати із системою як зі смартфона, так і з комп’ютера, не використовуючи різні програми з різною логікою роботи.

Основою побудови інтерфейсу в .NET MAUI є XAML. Він використовується для декларативного опису сторінок, елементів керування, розмітки, стилів і прив’язок даних. Такий підхід відокремлює візуальну

частину застосунку від логіки, що робить проєкт зрозумілішим і зручнішим для підтримки. Наприклад, сторінка розкладу може містити список занять, фільтр за тижнем і кнопку додавання події, а отримання даних з API або локального кешу виконується в окремих класах. Інтерфейс MAUI-застосунку складається зі сторінок, layout-контейнерів і візуальних елементів. Сторінка є верхнім рівнем інтерфейсу, а layout-елементи відповідають за розташування кнопок, списків, полів введення, текстових блоків, зображень та інших компонентів. У студентському органайзері окремими сторінками можуть бути Dashboard, Schedule, Subjects, Tasks, Files, Notes, Events, Group, Announcement Channel, Q&A та Settings.

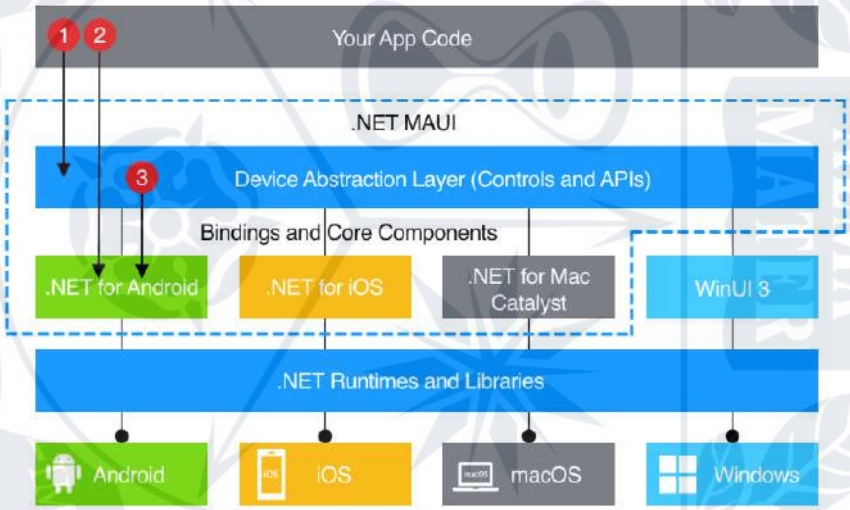

Рисунок 1.1 – Схема архітектури .NET MAUI

Джерело: [30] Сторінка What is .NET MAUI?

Окреме значення мають стилі та ресурси. Вони дозволяють зберігати повторювані значення: кольори, шрифти, розміри, відступи, шаблони елементів і стилі кнопок. Для студентського органайзера це важливо через велику кількість однотипних елементів. Завдяки стилям можна швидко змінювати вигляд усього застосунку, наприклад реалізувати світлу й темну тему або єдине оформлення карток.

Для організації логіки клієнтського застосунку доцільно застосовувати

патерн MVVM (Model-View-ViewModel)[24]. Model описує дані предметної області. View відповідає за інтерфейс і зазвичай реалізується у XAML. ViewModel містить стан сторінки, команди користувача та логіку підготовки даних для відображення. Такий поділ не змішує інтерфейс із бізнес-логікою, спрощує тестування і робить код стуктурованим.

Data binding дозволяє зв'язувати властивості ViewModel з елементами інтерфейсу[18]. Наприклад, список завдань може бути прив'язаний до колекції Tasks, а поле пошуку – до властивості SearchText. Коли дані у ViewModel змінюються, інтерфейс може оновлюватися автоматично. Це корисно для сторінок із частими змінами.

Клієнтський застосунок повинен не лише відображати дані із сервера, а й підтримувати локальне зберігання. Локальне сховище можна використовувати для кешування розкладу, предметів, завдань, подій, нотаток і налаштувань користувача. Для простих налаштувань підходять Preferences, для чутливих значень – SecureStorage, а для структурованих даних – SQLite. Це покращує швидкодію і дозволяє частково працювати із системою навіть за нестабільного інтернет-з'єднання.

1.2. Blazor Web App для вебверсії застосунку

У межах клієнт-серверної системи вебверсія є окремим інтерфейсом, який забезпечує доступ до функцій студентського органайзера через браузер. Вона доповнює Android/Windows застосунок і дозволяє працювати без встановлення окремої програми.

Blazor – це фреймворк екосистеми ASP.NET Core для створення інтерактивних веб-інтерфейсів із використанням C#, HTML та Razor-синтаксису[17]. Основною одиницею побудови інтерфейсу є Razor-компонент, який поєднує розмітку, логіку обробки подій, параметри, прив'язку даних і стан. Компоненти можна вкладати, повторно використовувати та поширювати між проєктами, що зручно для створення карток предметів, списків завдань, таблиць розкладу, форм подій, панелей файлів і повідомлень. Важливою

перевагою Blazor для цього проєкту є використання єдиного технологічного стеку.

Blazor Web App підтримує різні режими рендерингу компонентів: статичний серверний, інтерактивний серверний, WebAssembly та автоматичний режим[1]. Для студентського органайзера це дозволяє гнучко обирати спосіб роботи сторінок. Наприклад, сторінки входу або довідкова інформація можуть використовувати серверний рендеринг, а календар, розклад, фільтрація завдань чи файловий менеджер можуть працювати як інтерактивні компоненти.

Маршрутизація в Blazor дає змогу пов’язувати Razor-компоненти з URL-адресами за допомогою директиви @page. У системі це можна використати для створення окремих сторінок. Така структура спрощує навігацію та робить веб-версію зрозумілою для користувача. Для спільних елементів інтерфейсу в Blazor використовуються layout-компоненти. Вони дозволяють винести повторювані частини сторінок в окремий компонент. У студентському органайзері layout може містити головне меню з переходами до основних модулів, що забезпечує єдину структуру веб-інтерфейсу.

Форми є важливою частиною веб-версії, оскільки користувач створює велику кількість різноманітних об’єктів. У Blazor для цього застосовуються EditForm, моделі даних, обробники подій і механізми валідації. DataAnnotationsValidator дозволяє перевіряти обов’язкові поля, довжину, формат електронної пошти або допустимі значення ще до надсилання запиту на сервер. Серверна частина при цьому додатково перевіряє дані для захисту системи.

Авторизація потрібна для обмеження доступу до особистих і групових даних. У системі можуть бути різні ролі: студент, адміністратор групи, модератор. Веб-клієнт може приховувати недоступні кнопки чи сторінки, але остаточна перевірка прав повинна виконуватися на сервері, оскільки клієнтський код не є надійним місцем для контролю доступу.

Blazor Web App також підходить для створення адаптивного інтерфейсу.

Вебверсія повинна коректно працювати на ПК, ноутбуках, планшетах і смартфонах. Наприклад, на широкому екрані розклад може відображатися як таблиця тижня, а на вузькому – як список занять за днями. Файловий менеджер може мати табличний вигляд на ПК і картковий вигляд на мобільному пристрої.

ASP.NET Core Web API є серверною частиною системи, яка забезпечує взаємодію між клієнтськими застосунками, вебінтерфейсом, базою даних, файловим сховищем та іншими модулями[14][15]. У розроблюваній системі API виступає центральним інтерфейсом, через який .NET MAUI-застосунок і Blazor Web App отримують, створюють, оновлюють і видаляють дані.

ASP.NET Core Web API може будуватися на основі контролерів або Minimal APIs. Для складної системи з багатьма модулями доцільним є контролерний підхід, оскільки він дозволяє розділити endpoint-и за функціональними напрямами: авторизація, групи, предмети, завдання, файли, нотатки, події тощо. Контролери приймають HTTP-запити, викликають відповідні сервіси та повертають HTTP-відповіді.

Основою взаємодії клієнта і сервера є HTTP-протокол. Клієнт звертається до певного endpoint-а, сервер обробляє запит і повертає відповідь, зазвичай у форматі JSON. Завдяки цьому мобільний, Windows- і веб-клієнт можуть працювати з одним API, не дублюючи бізнес-логіку на кожній платформі.

Для організації endpoint-ів доцільно використовувати REST-підхід[38]. У ньому ресурси подаються через зрозумілі URI-адреси, а дії над ними виконуються стандартними HTTP-методами: GET – отримання даних, POST – створення, PUT або PATCH – оновлення, DELETE – видалення[25]. Наприклад, GET /api/subjects може повертати список предметів, а POST /api/subjects – створювати новий предмет. Документація OpenAPI дозволяє бачити список наявних endpoint-ів та їх методи[33].

Нормальним підходом API є використання стандартних HTTP-статусів. Успішні запити можуть повертати 200 OK, 201 Created або 204 No Content.

Помилки можуть позначатися статусами 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found або 409 Conflict. Це спрощує обробку відповідей у клієнтських застосунках.

Для передавання даних між клієнтом і сервером використовуються DTO. Вони описують структуру запитів і відповідей та не дозволяють відкривати клієнту внутрішні сутності бази даних напряму. Наприклад, замість повної сутності завдання клієнт може передати CreateTaskDto з назвою, описом, дедлайном, статусом і предметом. DTO підвищують безпеку, стабільність API та захищають службові поля від небажаної зміни.

Окрему роль відіграють автентифікація та авторизація. Автентифікація визначає користувача, а авторизація перевіряє його права. Для захисту API може використовуватися JWT Bearer authentication: після входу користувач отримує access token і передає його в заголовку запиту. Захищені endpoint-и позначаються атрибутом [Authorize], а доступ до адміністративних дій може обмежуватися ролями або політиками.

Entity Framework Core є сучасною ORM-технологією для платформи .NET, яка використовується для роботи з реляційними базами даних через об’єктно-орієнтовану модель[20]. У кросплатформній клієнт-серверній системі для організації навчального процесу студентів EF Core доцільно застосовувати на серверній частині для збереження користувачів, груп, предметів, розкладу, завдань, подій, файлів, нотаток, повідомлень, питань і відповідей. Ця технологія дає змогу описувати структуру даних засобами C#, а не працювати безпосередньо з великою кількістю SQL-запитів. ORM, або Object-Relational Mapping, забезпечує зв’язок між класами мови програмування та таблицями бази даних. У такому підході клас відповідає таблиці, властивості класу – стовпцям, а окремий об’єкт – запису в таблиці[9]. Це спрощує розробку, оскільки програміст працює з даними як з об’єктами предметної області. Наприклад, предмет, завдання, користувач або подія можуть бути представлені окремими сутностями, які надалі перетворюються на таблиці бази даних.

Основою роботи EF Core є сутності. Сутність – це клас, який описує певний об’єкт системи. Для студентського органайзера такими сутностями можуть бути користувач, група, навчальний рік, семестр, тощо. Кожна сутність містить набір властивостей, які визначають її дані. Наприклад, предмет може мати назву, опис, колір, прив’язку до семестру та зв’язки із завданнями, файлами, нотатками, подіями і розкладом.

Для взаємодії з базою даних у EF Core використовується контекст бази даних. Він є посередником між моделлю застосунку та реальною базою даних. Через контекст виконуються операції додавання, читання, оновлення та видалення даних. Також у ньому визначаються набори сутностей, які відповідають таблицям бази даних. Завдяки цьому серверна частина системи може централізовано працювати з усіма основними об’єктами: користувачами, групами, предметами, завданнями, файлами, нотатками та подіями.

Entity Framework Core підтримує підхід Code First. Його суть полягає в тому, що спочатку створюються класи сутностей, а потім на їх основі формується структура бази даних. Для зміни схеми бази використовуються міграції. Міграція описує, які саме зміни потрібно внести: створити таблицю, додати поле, змінити тип даних, створити індекс або додати зовнішній ключ. Це зручно під час розробки, оскільки структура бази даних може поступово розвиватися разом із системою.

Для вибірки даних EF Core використовує LINQ-запити. LINQ дозволяє фільтрувати, сортувати, групувати та проєктувати дані засобами C#. Наприклад, у студентському органайзері можна отримувати завдання певного предмету, найближчі дедлайни, події на обраний день, заняття поточного розкладу або питання без відповіді. При цьому EF Core перетворює LINQ-запити на SQL-запити до конкретного провайдера бази даних.

Для серверної частини системи доцільно використовувати SQL Server або PostgreSQL. SQL Server добре інтегрується з екосистемою Microsoft і ASP.NET Core, тому є зручним варіантом для .NET-розробки. PostgreSQL також є потужною реляційною СКБД з відкритим вихідним кодом, яка

підтримує складні запити, індекси, транзакції та розширені типи даних[37]. Обидва варіанти можуть використовуватися для збереження основних даних клієнт-серверної системи.

Окрему роль може виконувати SQLite у клієнтському застосунку[40]. Оскільки система передбачає Android/Windows клієнт на .NET MAUI, частина даних може зберігатися локально на пристрої користувача. SQLite добре підходить для локального кешу, оскільки не потребує окремого серверного процесу. У такому кеші можна зберігати розклад, список предметів, нотатки або останні отримані дані для швидкого доступу до них.

Для забезпечення цілісності та швидкодії бази даних важливо використовувати обмеження, індекси та транзакції[27]. Обмеження запобігають збереженню некоректних даних, індекси прискорюють пошук за часто використовуваними полями, а транзакції забезпечують узгодженість операцій, які складаються з кількох дій. Це особливо важливо для системи, де одночасно обробляються користувачі, групи, файли, завдання, події та права доступу.

У контексті файлового сховища база даних не обов’язково повинна зберігати самі фізичні файли. Доцільніше зберігати файли у файловій системі сервера або спеціалізованому сховищі, а в базі даних тримати метадані.

1.3. Автентифікація, авторизація та контроль доступу

У клієнт-серверній системі для організації навчального процесу важливе місце займає захист користувацьких даних. Оскільки студентський органайзер передбачає роботу з особистими файлами, кожна дія користувача має виконуватися лише після перевірки його особи та прав доступу. Для цього застосовуються механізми автентифікації, авторизації та контролю доступу.

Автентифікація – це процес перевірки особи користувача. У системі вона використовується під час реєстрації та входу в обліковий запис[35]. Користувач надає основні облікові дані, після чого система перевіряє їх коректність. Важливою вимогою є безпечне зберігання паролів, оскільки збереження їх у

відкритому вигляді створює значні ризики для конфіденційності. Тому пароль має зберігатися у захищеному вигляді, що унеможливлює його пряме відновлення.

Після успішної автентифікації користувач отримує можливість звертатися до захищених ресурсів системи. Для цього можуть використовуватися токени доступу, які підтверджують особу користувача під час подальшої взаємодії з сервером. Такий підхід дозволяє не передавати облікові дані під час кожного запиту, а використовувати спеціальний засіб підтвердження доступу. Токен може містити основну інформацію про користувача, його роль, належність до групи та термін дії.

Оскільки токен доступу зазвичай має обмежений час дії, у системах часто застосовується механізм оновлення доступу. Його призначення полягає в тому, щоб користувач міг продовжити роботу без повторного введення логіна і пароля. Це підвищує зручність використання системи та водночас зменшує ризики, пов’язані з довготривалим використанням одного токена доступу. Якщо такий токен буде перехоплено, він залишатиметься дійсним лише протягом обмеженого часу.

Авторизація – це процес перевірки прав користувача після встановлення його особи. У студентському органайзері вона потрібна для розмежування доступу між різними категоріями користувачів[34][36]. Наприклад, звичайний студент може переглядати розклад, створювати особисті нотатки, завантажувати власні файли та переглядати навчальні матеріали своєї групи. Адміністратор групи може керувати розкладом, додавати групові файли, створювати оголошення та модерувати окремі елементи спільного простору. Власник групи може мати розширені права, зокрема керувати учасниками, змінювати ролі та видаляти групу.

Для організації таких правил доцільно використовувати рольову модель доступу. Вона передбачає наявність кількох ролей, кожна з яких має власний набір дозволів. У межах студентського органайзера можуть використовуватися ролі власника, адміністратора, модератора, студента. Власник має найвищий

рівень доступу в межах групи, адміністратор керує основними навчальними даними, модератор відповідає за окремі елементи контенту, студент має базовий доступ до навчальних матеріалів.

Тимчасовий доступ до файлів може застосовуватися для обмеженого спільного використання матеріалів. Наприклад, студент може надати доступ до файлу на певний період, після завершення якого посилання стає недійсним. Такий механізм дозволяє зручно передавати матеріали іншим користувачам, не надаючи їм постійних прав доступу. Це особливо корисно для навчальних матеріалів, спільних завдань або короткотривалої перевірки файлів.

Контроль доступу також потрібен для навчальних сутностей системи. Створювати або редагувати предмети можуть лише користувачі з відповідними правами в групі. Групові завдання можуть додавати адміністратори або модератори, тоді як змінювати власний статус виконання завдання може кожен студент.

Особисті нотатки мають бути доступні лише їх автору, а групові події можуть бути видимими всім учасникам групи. У Q&A-модулі питання можуть створювати всі учасники, проте офіційні відповіді або модерація можуть бути доступні лише користувачам із підвищеними правами.

У backend-частині системи перевірка доступу має виконуватися на кількох рівнях. Перший рівень передбачає загальний захист API, коли окремі частини системи доступні лише автентифікованим користувачам або користувачам із певною роллю[26]. Другий рівень пов’язаний із перевіркою прав щодо конкретного об’єкта: групи, файлу, предмета, завдання або події. Це важливо, оскільки одна й та сама роль може надавати права лише в межах певної групи, але не в усій системі. Такий підхід потрібен через те, що користувач може бути учасником кількох груп і в кожній може мати різні ролі.

Під час розробки студентського органайзера важливе значення має не лише функціональність системи, а й якість взаємодії користувача з інтерфейсом. Це забезпечує UI/UX підхід. Оскільки застосунок призначений для щоденної роботи з потрібними користувачу речами, його інтерфейс

повинен бути зрозумілим, логічним, адаптивним і зручним для швидкого доступу до навчальної інформації[31].

UI відповідає за візуальне оформлення системи: структуру екранів, кольори, типографіку, кнопки, іконки, картки, списки та форми. UX охоплює загальний досвід користувача: наскільки легко студент може знайти потрібну інформацію, переглянути розклад, створити завдання, відкрити файл або отримати нагадування про дедлайн. У студентському органайзері ці напрями мають працювати разом, оскільки візуально привабливий інтерфейс без продуманої логіки не забезпечує зручності, а функціональна система без зрозумілого дизайну ускладнює щоденне використання.

Для побудови сучасного інтерфейсу доцільно орієнтуватися на принципи Material Design 3[29]. Ця дизайн-система пропонує єдиний підхід до організації кольорів, типографіки, компонентів, форм, відступів, станів елементів і адаптивності. Її використання дозволяє створити цілісний інтерфейс, який однаково зрозуміло сприймається на мобільних пристроях, у вебверсії та на настільних платформах.

Одним із ключових принципів Material Design 3 є ієрархічна організація інформації. У студентському органайзері це особливо важливо, оскільки система містить багато модулів. Головний екран має виконувати роль інформаційної панелі, де користувач одразу бачить найближчі заняття, актуальні дедлайни, події, оголошення та важливі сповіщення. Такий підхід зменшує кількість переходів між розділами та допомагає швидко оцінити поточний навчальний день або тиждень.

Для подання інформації доцільно використовувати карткову структуру, характерну для Material Design. Окремими блоками можуть бути найближче заняття, завдання з найближчим дедлайном, події на сьогодні, останні оголошення групи та швидкі переходи до основних розділів.

Material Design 3 значну увагу приділяє кольоровій системі (рис. 1.2). Ця система генерації кольорової палітри генерує велику кількість допоміжних кольорів для використання в різних сценаріях в світлій та темні темі.

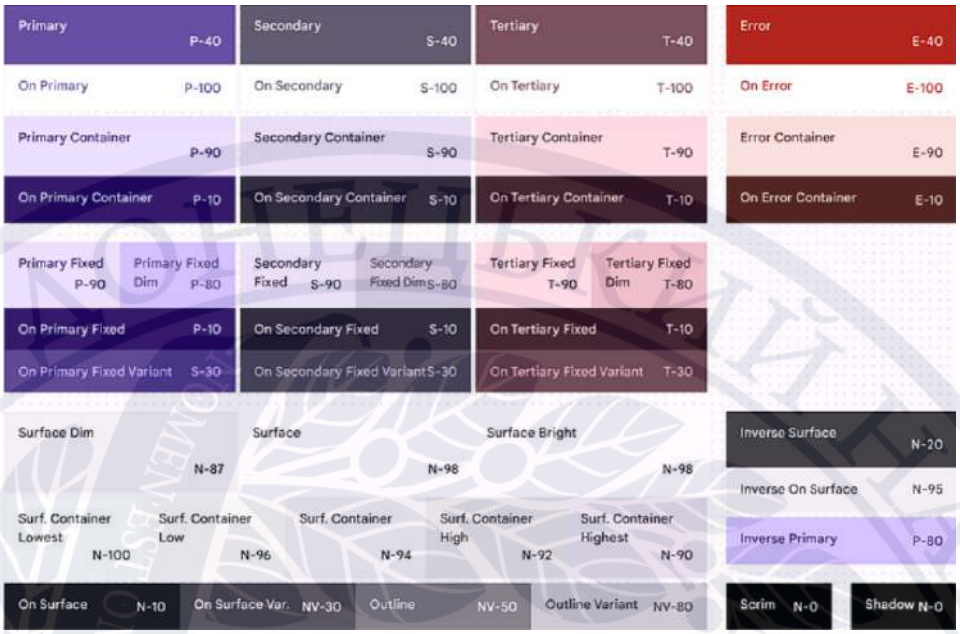

Рисунок 1.2 – Приклад кольорової системи Material Design 3

Джерело: [29], сторінка Styles/Color roles.

Для студентського органайзера кольорова система є корисною під час розділення навчальних предметів, типів подій і статусів завдань. Кожен предмет може мати власний колір, який використовується у розкладі, завданнях, нотатках і файловому просторі. Завдяки цьому користувач швидше орієнтується в системі та візуально пов’язує різні елементи з конкретною дисципліною.

Адаптивність є однією з основних вимог до інтерфейсу кросплатформного застосунку. Програма може використовуватися на Android, Windows і Web, тому структура екранів повинна змінюватися залежно від розміру пристрою. На мобільних екранах доцільними є компактні картки, нижня навігація та швидкі дії. У вебверсії та настільному клієнті можна використовувати бічне меню, таблиці, фільтри та кілька інформаційних блоків на одному екрані.

Material Design 3 також підтримує принцип узгодженої навігації. Основні розділи системи повинні мати однакові назви, подібні іконки та передбачуване розташування на різних платформах. Це зменшує час навчання

користувача і дозволяє легко переходити між мобільною, веб- та настільною версіями органайзера. Світла і темна теми є важливими складовими сучасного UI/UX. Студенти можуть користуватися органайзером у різних умовах освітлення, тому інтерфейс повинен залишатися комфортним як удень, так і вночі. Під час проєктування тем потрібно забезпечити достатній контраст між текстом і фоном, а також перевірити читабельність другорядного тексту, іконок, кнопок і кольорових позначок предметів.

У першому розділі було розглянуто теоретичні засади створення кросплатформної клієнт-серверного застосунку для організації навчального процесу студентів. Визначено основні проблеми, які виникають під час використання розрізнених інструментів для ведення розкладу, зберігання файлів, створення нотаток та групової взаємодії. Розглянуто основні технології та підходи, які доцільно використовувати для реалізації такої системи: .NET, C#, .NET MAUI, Blazor Web App, ASP.NET Core Web API, Entity Framework Core й принципи UI/UX-проєктування. Було обґрунтувано вибір технологічного стеку та сформувати основу для подальшого проєктування структури застосунку, його функціональних модулів, бази даних, API та клієнтських інтерфейсів.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ЗАСТОСУНКУ ОРГАНІЗАЦІЇ НАВЧАЛЬНОЇ ДІЯЛЬНОСТІ

2.1. Загальна концепція розробки кросплатформного застосунку

Розроблювана система є кросплатформним клієнт-серверним програмним комплексом, призначеним для організації навчального процесу студентів та академічних груп. Її основна ідея полягає в об'єднанні в одному середовищі інструментів, які студенти зазвичай використовують окремо: розкладу занять, навчальних завдань, дедлайнів, нотаток, файлового сховища, групових подій, оголошень, Q&A-простору та сповіщень. Завдяки цьому користувач отримує єдину точку доступу до навчальної інформації незалежно від пристрою.

Система проєктується як клієнт-серверна, оскільки такий підхід дозволяє розділити інтерфейс користувача, бізнес-логіку та зберігання даних[16][28]. Клієнтські застосунки відповідають за відображення інтерфейсу, введення даних, локальне кешування та надсилання запитів до сервера. Серверна частина обробляє запити, виконує авторизацію, перевіряє права доступу, реалізує бізнес-правила, працює з базою даних і файловим сховищем. Такий поділ спрощує підтримку системи, її розширення та одночасну роботу на кількох платформах.

Загальна структура системи складається з Android/Windows застосунку на основі .NET MAUI, веб-версії на основі Blazor Web App, серверного API на ASP.NET Core Web API, бази даних, файлового сховища, модуля автентифікації та авторизації, а також механізмів сповіщення й синхронізації даних. Усі ці компоненти виконують окремі функції, але взаємодіють між собою як єдина система.

Клієнтський застосунок на .NET MAUI призначений для щоденного використання на Android і Windows. Через нього студент може переглядати розклад, додавати завдання, контролювати дедлайни, створювати нотатки,

працювати з файлами, переглядати групові події, читати оголошення та користуватися Q&A-простором. Веб-версія на Blazor Web App забезпечує доступ до системи через браузер і є зручною для роботи з комп'ютера, особливо під час керування розкладом, файлами, груповими матеріалами та учасниками.

Центральним компонентом системи є ASP.NET Core Web API, яке виступає посередником між клієнтськими застосунками та базою даних[22]. Усі основні дії користувача виконуються через HTTP-запити до API. Сервер перевіряє токен користувача, визначає його роль і права доступу, виконує потрібну операцію, звертається до бази даних або файлового сховища та повертає результат клієнту у зручному форматі.

База даних використовується для зберігання структурованої інформації: облікових записів користувачів, академічних груп, ролей, навчальних років, семестрів, предметів, розкладів, завдань, подій, нотаток, оголошень, питань, відповідей, сповіщень і метаданих файлів. Самі файли доцільно зберігати у файловій системі сервера, а в базі даних залишати їх опис: назву, тип, розмір, власника, шлях, дату створення, прив'язку до предмету або завдання та права доступу.

Важливим елементом системи є автентифікація та авторизація. Автентифікація визначає особу користувача під час входу, а авторизація перевіряє, які дії він може виконувати. Наприклад, звичайний студент може переглядати груповий розклад і додавати особисті матеріали, але не повинен змінювати дані всієї групи без відповідних прав. Адміністратор або староста може керувати груповими даними, файлами, оголошеннями та Q&A-простором.

Отже, загальна концепція системи полягає у створенні єдиного кросплатформного середовища для організації навчального процесу. Система поєднує MAUI-клієнт, веб-версію на Blazor, серверне API, базу даних, файлове сховище, контроль доступу, сповіщення та базову синхронізацію. Такий підхід забезпечує централізоване зберігання навчальної інформації, доступ із різних

пристроїв, групову взаємодію та можливість подальшого розвитку системи.

Цільова аудиторія розроблюваної системи охоплює студентів, старост, адміністраторів академічних груп та учасників навчальних проєктів. Кожна категорія користувачів має власні потреби, однак усі вони пов’язані з організацією навчальної інформації, доступом до матеріалів, контролем дедлайнів і взаємодією в межах групи.

Основною категорією користувачів є студент. Для нього система виступає особистим навчальним органайзером, у якому можна переглядати розклад занять, контролювати завдання й дедлайни, зберігати файли, створювати нотатки, переглядати події та отримувати важливі повідомлення. Важливою перевагою для студента є доступ до системи з різних пристроїв: смартфона, комп’ютера або браузера. Це дозволяє використовувати застосунок як під час занять, так і вдома під час підготовки до навчання[13].

Староста або адміністратор групи використовує систему не лише для перегляду інформації, а й для її впорядкування та поширення серед інших учасників. До його основних сценаріїв належать редагування розкладу, створення групових оголошень, додавання важливих файлів, керування подіями, підтримка актуальності навчальних матеріалів і модерування Q&A-простору. Для таких користувачів важливою є система ролей і дозволів, яка дає змогу керувати груповими даними без надання зайвих прав усім учасникам.

Під час проєктування також потрібно враховувати різний рівень технічної підготовки користувачів. Частина студентів використовуватиме систему лише для базових дій, таких як перегляд розкладу, дедлайнів і файлів, тому інтерфейс має бути простим і зрозумілим. На смартфоні користувач найчастіше швидко перевірятиме найближчі заняття або повідомлення, а на комп’ютері зручніше працювати з файлами, нотатками, груповими матеріалами та адміністративними функціями[32].

Отже, система орієнтована не лише на окремого студента, а й на академічну групу загалом. Вона має підтримувати особисту організацію навчання, групову взаємодію, керування навчальними матеріалами та

розмежування прав доступу між різними категоріями користувачів.

Перелік можливостей, які повинна надавати розроблювана система користувачам визначають функціональні вимоги. Вони описують основні дії, які користувач може виконувати в застосунку, та визначають межі майбутньої системи. Оскільки проєкт орієнтований на організацію навчального процесу студентів і академічних груп, його функціональність має охоплювати як особисту роботу студента, так і групову взаємодію.

До базових функціональних вимог належить реєстрація та авторизація користувачів. Користувач повинен мати змогу створити обліковий запис, увійти до системи, вийти з облікового запису та отримувати доступ лише до тих даних, які дозволені його роллю[2].

Система повинна підтримувати роботу з академічними групами. Користувачі з відповідними правами мають змогу створювати групи, додавати учасників, призначати ролі та керувати доступом до групових матеріалів. Важливим функціональним модулем є розклад занять. Користувачі з відповідними правами повинні мати змогу додавати та редагувати заняття. Для академічної групи потрібно передбачити підтримку верхнього та нижнього тижня, а також перегляд розкладу за день або тиждень.

Система повинна підтримувати роботу з навчальними завданнями. Користувач має змогу створювати завдання, прив’язувати їх до предметів, задавати тип роботи, опис, дедлайн, статус виконання та прикріплені файли. Завдання можуть фільтруватися за предметом, статусом, типом або дедлайном. На головному екрані доцільно відображати найближчі дедлайни, щоб користувач міг швидко оцінити поточне навчальне навантаження.

Одним із ключових модулів системи є файлове сховище. Застосунок повинен дозволяти завантажувати, переглядати, видаляти та впорядковувати файли[6]. Файли можуть належати до особистого простору користувача, конкретного предмету, навчального завдання або групового простору. Для файлового модуля потрібно передбачити базове розмежування доступу: особисті файли доступні лише власнику, а групові матеріали можуть

переглядати учасники відповідної академічної групи[39]. Також система повинна містити нотатник. Користувач має змогу створювати, редагувати, видаляти та переглядати нотатки.

Окремий блок функціональності становлять події. Система повинна дозволяти створювати особисті та групові події, вказувати їх назву, опис, дату, час, учасників і, за потреби, прив'язку до предмету або семестру. Події можуть використовуватися для планування консультацій, захистів робіт, контрольних заходів, зустрічей навчальних команд та інших важливих дат.

Для групової комунікації потрібно передбачити канали оголошень і Q&A-простір. Канал оголошень призначений для важливих повідомлень, які повинні бачити всі учасники групи. Q&A-простір використовується для створення питань, додавання відповідей і формування бази корисної навчальної інформації.

Нефункціональні вимоги визначають якісні характеристики розроблюваної клієнт-серверної системи. Якщо функціональні вимоги описують, які дії може виконувати користувач, то нефункціональні вимоги визначають, наскільки система має бути зручною, стабільною, безпечною, продуктивною та придатною до подальшого розвитку.

Першою важливою вимогою є кросплатформність. Система повинна бути доступною на Android, Windows і Web. Android-версія потрібна для швидкого щоденного використання зі смартфона: перегляду розкладу, перевірки дедлайнів, отримання повідомлень і доступу до файлів. Windows-версія є зручною для роботи з більшим обсягом інформації, нотатками, навчальними матеріалами та файлами. Web-версія забезпечує доступ через браузер без встановлення окремого застосунку.

Для підтримки кросплатформності всі клієнти повинні взаємодіяти з єдиною серверною частиною через API. Дані про користувачів, групи, предмети, розклад, завдання, файли та події мають зберігатися централізовано. Це дає змогу забезпечити однакову поведінку системи на різних платформах і уникнути дублювання бізнес-логіки в кожному клієнтському застосунку.

Наступною вимогою є адаптивність інтерфейсу. Система повинна коректно працювати як на малих екранах смартфонів, так і на великих екранах комп'ютерів або планшетів. У мобільній версії інтерфейс має бути компактним і зручним для швидкого доступу до основних модулів. У Windows- і Web-версії доцільно використовувати більше екранного простору для таблиць, календарних подань, фільтрів, файлового менеджера та адміністративних дій.

Важливою нефункціональною вимогою є захист серверного API. Усі запити до захищених endpoint-ів повинні виконуватися лише авторизованими користувачами. Сервер має перевіряти автентичність користувача, його роль і права доступу перед виконанням будь-якої критичної дії. Перевірки не повинні виконуватися лише на клієнтському рівні, оскільки клієнтський застосунок може бути змінений або запит може бути сформований вручну.

Окреме значення має контроль доступу до файлів. Файлове сховище може містити особисті матеріали користувача, групові файли предметів і файли навчальних завдань. Тому система повинна визначати, хто має право переглядати, завантажувати, змінювати або видаляти конкретний файл[10]. Особисті файли мають бути доступні лише власнику, а групові матеріали – учасникам відповідної академічної групи або користувачам із наданими дозволами.

Система повинна підтримувати асинхронне виконання операцій. Завантаження файлів, отримання списку завдань, оновлення розкладу, створення подій або робота з груповими матеріалами не повинні блокувати інтерфейс користувача. Під час тривалих дій застосунок має показувати стан завантаження або повідомлення про результат операції. На серверному рівні асинхронна обробка запитів дозволяє ефективніше працювати з базою даних, файловою системою та одночасними зверненнями від різних клієнтів.

Для зручності роботи доцільно підтримати локальне кешування частини даних. Повноцінна offline-first модель може бути залишена для подальшого розвитку, однак застосунок повинен мати змогу зберігати останній отриманий розклад, найближчі завдання або деякі нотатки для швидкого відкриття. Після

відновлення з’єднання клієнтський застосунок має оновлювати локальні дані відповідно до актуального стану сервера.

2.2. Проєктування ролей і прав доступу

Проєктування ролей і прав доступу є важливою частиною клієнт-серверної системи, оскільки застосунок передбачає не лише особисте використання студентом, а й спільну роботу академічної групи. Тому система повинна чітко визначати, які дії доступні кожному типу користувача.

У межах системи доцільно використати рольову модель доступу. Кожен учасник групи має певну роль, а роль визначає набір дозволених дій. Такий підхід спрощує керування правами, зменшує ризик випадкового видалення або зміни важливих даних і робить поведінку системи зрозумілою для користувачів. Основними ролями є Owner, Admin, Moderator та Student.

Найвищим рівнем доступу є роль Owner. Вона надається користувачу, який створив академічну групу або основний навчальний простір. Власник має повний контроль над групою, її учасниками, ролями, навчальними роками, семестрами, предметами, розкладом, груповими файлами, каналами оголошень і Q&A-простором. Також Owner може призначати адміністраторів і модераторів, видаляти учасників або передавати право власності іншому користувачу.

Роль Admin призначена для користувачів, які беруть участь в організації навчального процесу групи. Це може бути староста, заступник старости або інший відповідальний студент. Адміністратор може створювати й редагувати предмети, додавати заняття до розкладу, керувати дедлайнами, створювати групові події, завантажувати групові файли, публікувати оголошення та відповідати на питання в Q&A-просторі. Водночас Admin не повинен мати повного доступу до критичних налаштувань, наприклад передачі права власності або видалення Owner.

Роль Moderator використовується для контролю порядку в комунікаційних модулях. Модератор може редагувати або видаляти некоректні

повідомлення, приховувати недоречні коментарі, закривати дубльовані питання, позначати актуальні відповіді та стежити за змістом Q&A-простору. При цьому Moderator не обов'язково має доступ до редагування розкладу, предметів або структури семестру.

Основною роллю більшості користувачів є Student. Студент може переглядати розклад, предмети, завдання, дедлайни, групові файли, оголошення, події та Q&A-простір. Також він може створювати особисті нотатки, завантажувати власні файли, додавати особисті події, змінювати статуси своїх завдань і ставити питання. У каналах оголошень студент може залишати коментарі, якщо це дозволено налаштуваннями каналу.

Права доступу можна поділити на кілька груп: керування групою, керування навчальними даними, робота з файлами, комунікація та модерація. До керування групою належать запрошення учасників, зміна ролей і редагування параметрів групи. До навчальних даних належать предмети, розклад, семестри, завдання та групові події. Права роботи з файлами визначають, хто може переглядати, завантажувати, змінювати або видаляти матеріали. Комунікаційні права регулюють оголошення, коментарі, питання й відповіді, а права модерації дозволяють керувати контентом інших користувачів.

Перевірка прав доступу має виконуватися не лише в інтерфейсі, а й на рівні серверного API. Клієнтський застосунок може приховувати недоступні кнопки, однак основна перевірка повинна виконуватися на сервері перед кожною критичною дією. Наприклад, при зміні розкладу, видаленні файлу або редагуванні ролей API має перевірити роль користувача та його права в конкретній групі.

Під час проєктування потрібно дотримуватися принципу мінімально необхідних прав. Користувач повинен мати лише ті дозволи, які потрібні для виконання його функцій. Це зменшує ризик випадкового пошкодження даних і підвищує безпеку системи. Також система повинна дозволяти змінювати ролі протягом навчального року без втрати історії дій користувачів.

Одним із важливих етапів планування системи є визначення структури навчальних даних. Вона потрібна для того, щоб інформація про навчальний процес не зберігалася як набір окремих непов’язаних записів, а була впорядкована відповідно до реальної організації навчання. Основними рівнями такої структури є навчальний рік, семестр і предмет.

Навчальний рік виступає верхнім рівнем групування навчальної інформації. Він може містити один або кілька семестрів і мати назву, дату початку, дату завершення, статус активності та зв’язок з академічною групою. Статус активності потрібний для того, щоб система могла відрізняти поточний навчальний рік від завершених періодів.

У межах навчального року створюються семестри. Семестр є основним навчальним контекстом, до якого прив’язуються предмети, розклад, завдання, події, файли, нотатки, оголошення та Q&A-простір. Для семестру доцільно зберігати назву, порядковий номер, дату початку, дату завершення, кількість навчальних тижнів і статус активності. Активний семестр використовується системою за замовчуванням під час відображення головного екрана, розкладу, списку завдань і найближчих подій.

Предмет є центральною одиницею організації навчальної інформації в межах семестру. Він розглядається не лише як назва дисципліни в розкладі, а як окремий інформаційний простір, що об’єднує пов’язані заняття, завдання, файли, нотатки, події, оголошення, питання та відповіді. Такий підхід дозволяє користувачу швидко перейти від конкретної дисципліни до всіх матеріалів, які з нею пов’язані. Основними даними предмету є назва дисципліни, опис, викладач, колір, семестр, група, дата створення та статус. Колір предмету допомагає швидко розрізняти дисципліни в розкладі, завданнях, подіях і нотатках.

Предмет повинен бути пов’язаний із модулем розкладу. Запис заняття в розкладі має містити посилання на відповідний предмет, день тижня, час початку й завершення, тип заняття, аудиторію або посилання на онлайн-заняття. Якщо в навчальному процесі використовується поділ на верхній і

нижній тиждень, система повинна враховувати це під час відображення занять.

Також предмет пов’язується із завданнями. У межах дисципліни можуть створюватися лабораторні, практичні, домашні, курсові, модульні та інші навчальні роботи. Для кожного завдання потрібно зберігати назву, опис, тип, статус виконання, дедлайн, прикріплені файли та зв’язок із предметом. Це дає змогу користувачу переглядати завдання як у загальному списку, так і на сторінці конкретної дисципліни.

Для групової взаємодії предмет може мати пов’язані оголошення та Q&A-простір. Оголошення використовуються для важливої інформації щодо дисципліни, а Q&A-простір – для збереження типових питань і відповідей. Наприклад, студенти можуть залишати питання щодо оформлення звіту, дедлайну або вимог до лабораторної роботи, а адміністратор чи модератор може додавати відповідь.

Після завершення семестру його дані не повинні видалятися. Семестр і пов’язані з ним предмети мають переходити до архівного стану. Архівні дані не відображаються серед активних за замовчуванням, але залишаються доступними для перегляду. Це дозволяє користувачу повернутися до старих предметів, знайти потрібні файли, переглянути виконані завдання або використати попередні матеріали під час підготовки до іспитів чи курсових робіт.

2.3. Проєктування основних модулів застосунку

Модуль розкладу є одним з основних елементів системи, оскільки через нього студент отримує щоденну інформацію про навчальні заняття, їх час, місце проведення, тип заняття та зв’язок із відповідним предметом. Розклад повинен допомагати користувачу швидко зрозуміти, які заняття заплановані на поточний день або тиждень, а також забезпечувати зв’язок з іншими навчальними модулями системи.

Основною сутністю модуля є запис розкладу, який належить до певного семестру, академічної групи та предмету. Груповий розклад використовується

всіма учасниками групи та редагується лише користувачами з відповідними правами доступу. Звичайний студент може переглядати заняття, але не повинен мати можливості змінювати розклад для всієї групи. Адміністратор, староста або користувач із відповідним дозволом може створювати, редагувати та видаляти записи розкладу.

Кожен елемент розкладу повинен містити основну інформацію про заняття: предмет, час початку, час завершення, тип заняття, викладача, аудиторію. До типів занять можуть належати лекція, лабораторна робота, практичне заняття, семінар, консультація або інший навчальний формат. Прив'язка заняття до предмету є важливою, оскільки вона дозволяє швидко перейти зі сторінки розкладу до матеріалів відповідної дисципліни, завдань, файлів, нотаток або Q&A-простору.

Важливою вимогою до модуля є підтримка верхнього та нижнього тижня. У багатьох закладах освіти частина занять проводиться не щотижня, а залежно від номера навчального тижня. Тому для кожного заняття потрібно передбачити тип повторення: щотижня, тільки на верхньому тижні або тільки на нижньому тижні. Для правильного визначення поточного тижня система повинна враховувати дату початку семестру та початковий тип тижня.

Розклад має бути пов'язаний із семестром. Це дозволяє відображати тільки актуальні заняття поточного навчального періоду та не змішувати їх із даними попередніх семестрів. Після завершення семестру розклад не повинен видалятися, а має залишатися доступним в архіві разом із предметами, завданнями, файлами та подіями. Такий підхід дає змогу студенту переглядати історію навчання та повертатися до старих матеріалів за потреби.

Під час відображення розкладу потрібно передбачити кілька зручних режимів перегляду. Основним є перегляд поточного дня, де користувач бачить найближчі заняття, їх час, тип і місце проведення. Також доцільно реалізувати тижневий перегляд, який дозволяє оцінити навчальне навантаження протягом усього тижня. У мобільному застосунку інтерфейс розкладу має бути компактним, а у Web і Windows версії можна використовувати ширший простір

екрана для таблиці або календарного подання.

На рівні структури даних для модуля розкладу доцільно передбачити сутність ScheduleItem. Вона може містити ідентифікатор предмету, семестру й групи, день тижня, час початку та завершення, тип заняття, тип тижня, аудиторію, викладача, посилання на онлайн-заняття, підгрупу та додатковий опис. Така структура дозволяє зберігати всі основні параметри заняття та пов’язувати розклад з іншими модулями системи.

Модуль завдань є одним із ключових елементів системи організації навчального процесу, оскільки він дозволяє студенту контролювати виконання лабораторних, практичних, домашніх, курсових, модульних, контрольних та інших навчальних робіт. Основне призначення цього модуля полягає у централізованому зберіганні інформації про навчальні обов’язки, дедлайни, поточний стан виконання, прив’язку до предметів і пов’язані файли.

У системі навчальне завдання розглядається як окрема інформаційна сутність. До основних атрибутів завдання належать назва, тип, статус виконання, дедлайн, опис, предмет і пов’язані ресурси. Назва повинна коротко відображати зміст роботи, наприклад лабораторна робота, практичне завдання або підготовка до модульного контролю. Тип завдання використовується для класифікації навчальної активності та допомагає користувачу швидко відрізняти різні види робіт.

Важливою характеристикою завдання є статус виконання. Він дає змогу відстежувати поточний стан роботи та формувати загальну картину навчального навантаження. Для модуля доцільно передбачити такі основні статуси: заплановано, у процесі, виконано, здано, перевіряється, завершено та прострочено. Такий поділ дозволяє студенту бачити, які завдання ще потрібно виконати, які вже завершені, а які потребують термінової уваги.

Дедлайн є одним із найважливіших параметрів завдання. Для кожної роботи потрібно передбачити дату та, за потреби, точний час завершення. Це дає змогу відображати найближчі дедлайни на головному екрані, формувати список термінових завдань, виділяти прострочені роботи та створювати

нагадування. Додатково можна передбачити пріоритет завдання, щоб студент міг самостійно визначати важливість конкретної роботи.

Опис завдання призначений для зберігання вимог до виконання. У ньому можуть міститися короткі інструкції, умови лабораторної або практичної роботи, посилання на матеріали, вимоги до оформлення звіту, критерії оцінювання або коментарі адміністратора групи. Завдяки цьому завдання стає не лише записом із датою, а повноцінним навчальним об’єктом, що об’єднує потрібну інформацію для виконання роботи.

Кожне завдання повинно бути пов’язане з певним предметом. Такий зв’язок дозволяє групувати завдання за дисциплінами, переглядати всі активні роботи в межах конкретного предмету та формувати загальну картину навчального навантаження семестру. У загальному списку користувач може переглядати всі завдання одразу, а також фільтрувати їх за предметом, типом, статусом, дедлайном або пріоритетом.

Окрему роль у модулі завдань відіграють файли. Завдання може містити групові ресурси та особисті файли користувача. До групових ресурсів належать методичні вказівки, шаблони звітів, приклади виконання, презентації або інші матеріали, потрібні всім студентам групи. Особисті файли – це індивідуальні матеріали студента, наприклад власний звіт, код програми, нотатки, чернетки або скріншоти результатів.

Для зручності користувача модуль завдань повинен підтримувати різні способи перегляду інформації. Основним варіантом є список завдань, у якому відображаються назва, предмет, тип, статус і дедлайн. Також доцільно передбачити сортування за датою завершення, фільтрацію за предметами, пошук за назвою або описом, а також окреме відображення активних, завершених і прострочених завдань.

Модуль має враховувати різницю між груповими та особистими завданнями. Групові завдання створюються користувачами з відповідними правами та доступні всім учасникам академічної групи. Водночас кожен студент повинен мати можливість змінювати власний статус виконання,

додавати особисті файли, створювати індивідуальні завдання та вести власні коментарі. Такий підхід поєднує централізоване керування навчальними завданнями з особистою організацією роботи користувача.

З погляду прав доступу, створення та редагування групових завдань має бути доступним власнику групи, адміністратору або іншому користувачу з відповідним дозволом. Звичайний студент може переглядати групові завдання, змінювати власний статус виконання, прикріплювати особисті файли та створювати власні записи. Це захищає групову інформацію від випадкових змін, але не обмежує студента в особистому плануванні.

Файлове сховище є одним із ключових модулів системи, оскільки значна частина навчального процесу пов’язана зі зберіганням і повторним використанням електронних матеріалів. До таких матеріалів належать методичні вказівки, лабораторні роботи, практичні завдання, конспекти, презентації, звіти, приклади коду, таблиці та документи для підготовки до іспитів.

Основна мета файлового сховища полягає у створенні впорядкованої структури, яка дозволяє користувачу швидко знаходити потрібні навчальні матеріали, зберігати власні файли та отримувати доступ до групових ресурсів[12]. На відміну від звичайного хмарного диска, файловий модуль у студентському органайзері має бути пов’язаний з іншими сутностями системи: предметами, завданнями, семестрами та академічними групами.

У системі доцільно передбачити кілька основних типів файлових просторів. Першим є особистий простір користувача. Він призначений для файлів, які належать конкретному студенту та не обов’язково прив’язані до певного предмету або завдання. У цьому просторі можуть зберігатися особисті конспекти, шаблони звітів, довідкові матеріали або документи, які стосуються кількох дисциплін одночасно. За замовчуванням доступ до такого простору має лише його власник.

Другим типом є особисті файли предмету. Такий простір створюється в межах конкретної навчальної дисципліни й дозволяє студенту зберігати

матеріали, пов’язані з певним предметом. Наприклад, це можуть бути власні конспекти лекцій, чернетки лабораторних робіт, особисті розрахунки, приклади коду або підготовлені звіти. Завдяки цьому предмет виступає логічним контейнером для відповідних матеріалів.

Третім типом є групові файли предмету. Вони призначені для матеріалів, доступних усім учасникам академічної групи. До таких файлів можуть належати методичні вказівки, списки тем, приклади виконання завдань, презентації викладача, шаблони звітів або документи, завантажені старостою чи адміністратором групи. Оскільки ці матеріали мають спільний характер, їх редагування повинно бути обмежене. Звичайний студент може мати право перегляду й завантаження, а адміністратор або інший користувач із відповідними правами – право додавання, зміни та видалення.

Окремо потрібно передбачити файли завдання. Такі файли прив’язуються не лише до предмету, а й до конкретної навчальної роботи: лабораторної, практичної, домашньої, курсової або модульної. Наприклад, до лабораторної роботи можуть бути прикріплені методичні вказівки, вхідні дані, шаблон звіту, додаткові матеріали та особистий файл зі звітом студента. Це дозволяє користувачу бачити всі матеріали, потрібні для виконання завдання, безпосередньо на сторінці цієї роботи.

Для реалізації файлового сховища доцільно використати комбінований підхід: фізичні файли зберігати у файловій системі сервера, а метадані про них – у базі даних. У базі даних можуть зберігатися ідентифікатор файлу, назва, розширення, MIME-тип, розмір, шлях до фізичного файлу, власник, дата створення, тип файлового простору та зв’язок із предметом або завданням. Такий підхід дозволяє не перевантажувати базу даних великими файлами, але зберігати інформацію, потрібну для пошуку, фільтрації, відображення та перевірки доступу.

Особливу увагу потрібно приділити правам доступу. Для MVP достатньо передбачити базові рівні: власник, читання та зміна. Власник має повний контроль над файлом або файловим простором. Право читання дозволяє

переглядати або завантажувати файл без можливості його редагування. Право зміни дозволяє оновлювати файл або його метадані. Перевірка прав повинна виконуватися на рівні серверного API під час перегляду, завантаження, перейменування, видалення або зміни файлу.

У клієнтському застосунку файлове сховище має бути інтегроване з іншими модулями системи. На сторінці предмету користувач повинен бачити особисті та групові файли цієї дисципліни. На сторінці завдання мають відображатися матеріали, пов’язані саме з цією роботою. Також доцільно передбачити окремий файловий менеджер, у якому користувач може швидко перейти до потрібної категорії матеріалів.

Модулі нотаток і подій є допоміжними елементами системи організації навчального процесу. Вони доповнюють основні модулі предметів, розкладу, завдань і файлів, оскільки дозволяють студенту зберігати власні навчальні записи, планувати важливі активності та швидко знаходити інформацію, пов’язану з конкретним предметом або семестром.

Основними діями з нотатками є створення, редагування, видалення, перегляд, пошук, фільтрація та сортування. Користувач повинен мати можливість створити нотатку як із загального розділу нотатника, так і зі сторінки конкретного предмету або завдання. Якщо нотатка створюється зі сторінки предмету, система може автоматично встановлювати цей предмет як її тему.

Структура нотатки повинна містити назву, текстовий вміст, тему, теги, дату створення, дату останнього редагування, власника та необов’язковий зв’язок із предметом або завданням. Тема використовується як основна категорія нотатки. За замовчуванням темами можуть виступати навчальні предмети поточного семестру, але користувач також може створювати власні теми, наприклад підготовка до екзамену, дипломна робота або корисні посилання.

Теги потрібні для додаткової класифікації записів. Одна нотатка може мати кілька тегів, наприклад лабораторна, код, важливо, екзамен, звіт або

питання. Завдяки цьому користувач може знаходити записи не лише за предметом, а й за типом навчальної інформації. Наприклад, студент може переглянути всі нотатки з тегом екзамен незалежно від дисципліни.

Модуль подій призначений для планування навчальних, організаційних та особистих активностей. Події можуть використовуватися для фіксації консультацій, захистів лабораторних робіт, екзаменів, додаткових занять, зустрічей навчальних команд або інших важливих дат. На відміну від завдань, подія описує конкретну активність у часі, а не роботу, яку потрібно виконати.

У системі доцільно передбачити два основні типи подій: особисті та групові. Особисті події створюються конкретним користувачем і доступні лише йому. Вони можуть використовуватися для індивідуального планування, самостійної підготовки або власних нагадувань. Групові події створюються для всієї академічної групи або окремих її учасників. До них можуть належати консультації, організаційні збори, захисти робіт або спільні навчальні зустрічі.

Кожна подія повинна мати назву, опис, дату й час початку, дату й час завершення, тип події, автора, місце проведення та необов’язковий зв’язок із групою, семестром або предметом. Якщо подія стосується конкретної дисципліни, її доцільно прив’язувати до відповідного предмету. Наприклад, консультація з програмування або захист лабораторної роботи можуть відображатися не лише в загальному календарі, а й на сторінці конкретного предмету.

Особисті та групові події мають відрізнятися кольором, піктограмою або іншим маркером. Події, пов’язані з предметом, можуть використовувати колір відповідної дисципліни. Також доцільно передбачити фільтри, які дозволяють показувати лише особисті події, групові події, події конкретного предмету або події поточного семестру.

Отже, нотатки й події доповнюють основні навчальні модулі та роблять систему зручнішою для щоденного використання. Нотатник дозволяє студенту зберігати й систематизувати власні записи за темами та тегами, а модуль подій забезпечує планування особистих і групових активностей. Разом ці модулі

допомагають користувачу краще організувати навчальний процес і не втрачати важливу інформацію.

2.4. Проєктування групової взаємодії користувачів та організація даних

Модулі оголошень і Q&A-простору призначені для організації групової взаємодії в межах академічної групи або окремого предмету. Їх основне завдання полягає в тому, щоб відокремити важливу навчальну інформацію від звичайного неструктурованого спілкування. Канал оголошень використовується для централізованої публікації повідомлень, а Q&A-простір – для збереження питань, відповідей і типових пояснень.

Канал оголошень є спеціалізованим інформаційним простором, у якому публікувати основні повідомлення можуть лише користувачі з відповідним дозволом. До таких користувачів можуть належати Owner, Admin, Moderator або староста групи. Звичайні студенти можуть переглядати оголошення, залишати коментарі та реагувати на повідомлення, якщо це дозволено налаштуваннями каналу.

Оголошення можуть стосуватися змін у розкладі, нових дедлайнів, навчальних матеріалів, консультацій, організаційних питань, подій або важливих повідомлень від адміністратора групи. На відміну від звичайного чату, канал оголошень має чіткіші правила публікації, тому важлива інформація не губиться серед випадкових повідомлень. Структура оголошення повинна містити заголовок, текст повідомлення, автора, дату створення, дату останнього редагування, ознаку закріплення, прикріплені файли, коментарі та реакції. За потреби оголошення може бути пов’язане з предметом, завданням, подією або файлом. Наприклад, повідомлення про нову лабораторну роботу може містити посилання на відповідне завдання, дедлайн і навчальні матеріали.

Коментарі до оголошень потрібні для уточнень і короткого обговорення. Для MVP достатньо лінійної структури коментарів, де всі відповіді

відображаються під конкретним оголошенням у хронологічному порядку. Модератори або адміністратори повинні мати можливість видаляти некоректні коментарі, редагувати помилкові повідомлення та закріплювати найважливіші оголошення.

Q&A-простір призначений для організації навчальних питань і відповідей. Користувач може створити питання, пов’язати його з групою або предметом, а користувач із відповідними правами може додати офіційну відповідь. Такий формат зручніший за звичайний чат, оскільки питання не губляться серед повідомлень і можуть повторно використовуватися як база знань групи. Основною одиницею Q&A-простору є питання. Воно містить заголовок, текстовий опис, автора, дату створення, зв’язок із групою або предметом, статус, кількість реакцій і відповіді. Заголовок коротко відображає суть питання, а опис дає змогу пояснити проблему детальніше. Наприклад, студент може створити питання щодо вимог до лабораторної роботи, формату здачі файлів, дедлайну або помилки в завданні.

Для питань доцільно передбачити кілька статусів: Open, Answered, Resolved і Closed. Статус Open використовується для нових питань. Answered означає, що до питання додано відповідь. Resolved позначає питання, для якого відповідь прийнята як остаточна. Closed використовується для закритих або неактуальних питань.

В інтерфейсі користувача канал оголошень може бути представлений як список повідомлень у зворотному хронологічному порядку. Закріплені або важливі оголошення доцільно відображати вище інших. Кожне повідомлення повинно містити заголовок, короткий текст, автора, дату публікації, прикріплені файли, кількість коментарів і реакцій.

Проєктування бази даних є одним із ключових етапів планування клієнт-серверної системи для організації навчального процесу студентів. База даних повинна забезпечувати збереження користувачів, академічних груп, навчальних років, семестрів, предметів, розкладу, завдань, файлів, нотаток, подій, оголошень, Q&A-простору, сповіщень і службових даних авторизації.

Для реалізації системи використовується реляційна модель даних, оскільки основні об’єкти мають чітку структуру та зв’язки між собою. Наприклад, користувач може бути учасником кількох груп, група може містити навчальні роки, навчальний рік складається із семестрів, семестр містить предмети, а предмет об’єднує розклад, завдання, файли, нотатки та питання. Така структура добре описується через первинні й зовнішні ключі.

Основою моделі користувачів є таблиця Users. Вона зберігає облікові записи користувачів і містить такі дані, як електронна пошта, ім’я користувача, хеш пароля, ім’я та прізвище. Поля Email і UserName повинні бути унікальними, оскільки вони використовуються для ідентифікації користувача в системі. Пароль не зберігається у відкритому вигляді, тому в базі міститься лише його хеш.

Для підтримки входу та оновлення сесій використовується таблиця RefreshTokens. Вона пов’язана з користувачем і зберігає хеш refresh token, дату завершення дії, дату відкликання та інформацію про заміну токена. Це дозволяє реалізувати безпечну авторизацію без необхідності повторного введення пароля після завершення короткотривалого access token.

Таблиця Groups описує академічні групи або навчальні спільноти. Вона містить назву, опис і код групи. Код використовується для ідентифікації або приєднання до групи, тому має бути унікальним. З групою пов’язані учасники, навчальні роки, розклади, оголошення, події, файли та питання. На ER-діаграмі (рис. 2.1) зображено основні таблиці бази даних, Groups входить до списку критично важливих таблиць, адже саме навколо неї формується групова взаємодія.

Зв’язок між користувачами й групами реалізується через таблицю GroupMembers. Вона містить ідентифікатор користувача, ідентифікатор групи, роль користувача та дату приєднання. Поле Role визначає права користувача в межах групи. У системі використовуються ролі Owner, Admin, Moderator, Student і Guest. Для запобігання дублюванню записів доцільно використовувати унікальне обмеження на пару UserId і GroupId.

Навчальна структура представлена таблицями AcademicYears, Semesters і Subjects. Таблиця AcademicYears зберігає навчальні роки, їх назву, початковий і кінцевий рік, а також ознаку архівації. Кожен навчальний рік належить певній групі. Таблиця Semesters описує семестри в межах навчального року та містить назву, номер, дату початку, дату завершення й ознаку архівації. Семестр є основним контекстом для предметів і розкладів.

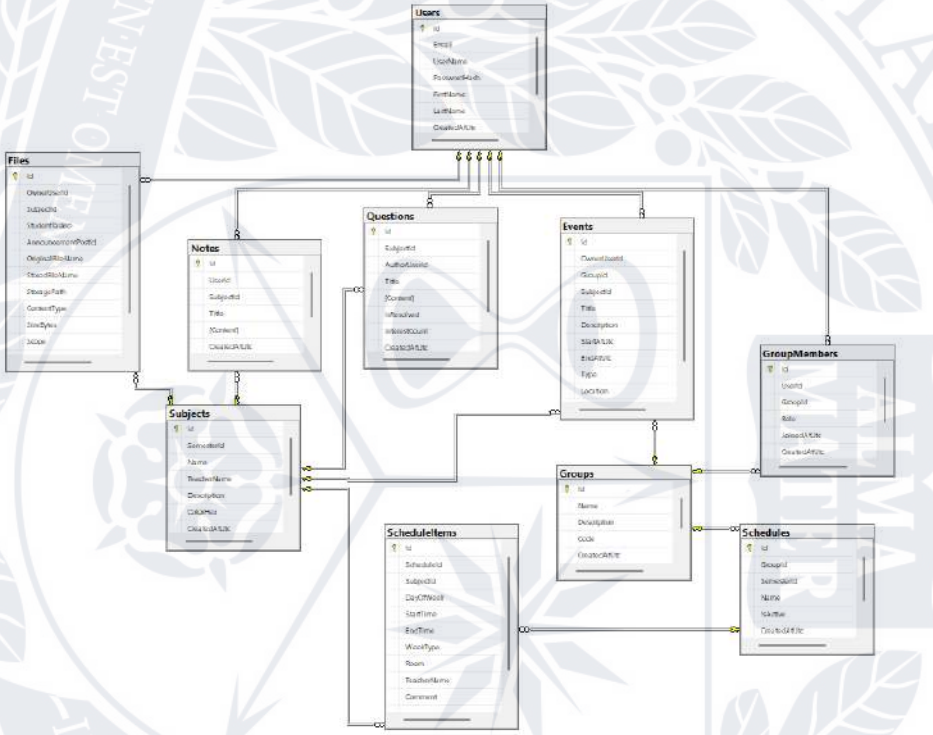

Рисунок 2.3 – Спрощена ER-діаграма бази даних застосунку.

Таблиця Subjects є однією з центральних таблиць системи. Вона описує навчальні дисципліни та містить назву, ім’я викладача, контакти викладача, опис і колірне позначення. Кожен предмет належить до конкретного семестру. З предметом пов’язуються заняття в розкладі, навчальні завдання, файли, нотатки й питання Q&A-простору.

Модуль розкладу реалізується через таблиці Schedules і ScheduleItems. Таблиця Schedules описує розклад групи в межах семестру та містить назву й ознаку активності. Таблиця ScheduleItems зберігає окремі заняття: назву, день

тижня, час початку й завершення, тип тижня, тип заняття, місце проведення, аудиторію, викладача та коментар. Поле WeekType дає змогу підтримувати заняття для кожного тижня, верхнього тижня або нижнього тижня. Поле LessonType визначає тип заняття: лекція, практична, лабораторна, семінар, консультація, екзамен або інший формат.

Навчальні завдання зберігаються в таблиці Tasks, яка відповідає доменній сутності StudentTask. Завдання пов’язане з предметом і користувачем, який його створив. Основними полями є назва, опис, тип завдання, статус і дедлайн. Тип завдання визначається значеннями Laboratory, Practical, Homework, Coursework, ModuleTest, Exam або Other. Статус може мати значення NotStarted, InProgress, Done або Overdue.

Файлове сховище реалізується через таблиці Files і FileShares. Таблиця Files відповідає сутності FileItem і зберігає не сам файл, а його метадані: початкову назву, збережену назву, шлях до файлу, MIME-тип, розмір, власника та прив’язку до групи, предмету, завдання або оголошення. Поле Scope визначає тип файлового простору: Personal, SubjectPersonal, SubjectGroup, TaskFile або AnnouncementAttachment.

Таблиця FileShares використовується для надання доступу до файлів іншим користувачам. Вона містить посилання на файл, користувача, який надав доступ, користувача, який отримав доступ, права читання, права зміни та дату завершення доступу. Такий підхід дозволяє підтримувати як постійний, так і тимчасовий доступ до окремих файлів.

Нотатки користувачів зберігаються в таблиці Notes. Нотатка належить конкретному користувачу та може бути пов’язана з предметом або навчальним завданням. Основними полями є назва й текстовий вміст. Для тегів використовується окрема таблиця NoteTags, де кожен тег пов’язаний з конкретною нотаткою. Це спрощує пошук і фільтрацію записів за навчальними темами.

Події реалізуються через таблицю Events, яка відповідає сутності CalendarEvent. Подія має власника, може бути пов’язана з групою або

предметом, містить назву, опис, дату й час початку, дату й час завершення, тип події та місце проведення. Тип події визначає, чи є вона особистою або груповою. Така структура дозволяє використовувати події як для індивідуального планування, так і для координації академічної групи.

Групові оголошення реалізуються через таблиці AnnouncementPosts і AnnouncementComments. Таблиця AnnouncementPosts зберігає основні повідомлення групи: автора, заголовок, текст, ознаку закріплення та зв’язок із групою. До оголошення можуть бути прикріплені файли через зв’язок із FileItem. Коментарі до оголошень зберігаються в таблиці AnnouncementComments, яка містить посилання на оголошення, автора та текст коментаря.

Q&A-простір реалізується через таблиці Questions, QuestionAnswers і QuestionReactions. Таблиця Questions зберігає питання користувачів, їх заголовок, зміст, автора, статус, ознаку вирішення, кількість реакцій, а також прив’язку до групи або предмету. Статус питання може мати значення Open, Answered, Resolved або Closed. Відповіді зберігаються в таблиці QuestionAnswers, де вказується питання, автор відповіді, текст і ознака прийнятої відповіді. Таблиця QuestionReactions фіксує реакції користувачів на питання та має унікальне обмеження на пару питання й користувача.

Сповіщення зберігаються в таблиці Notifications. Вона містить користувача, заголовок, текст повідомлення, цільове посилання, ознаку прочитання та дату прочитання. Така структура дозволяє реалізувати внутрішні повідомлення про нові події, дедлайни, оголошення, відповіді в Q&A або інші важливі зміни в системі.

Основні зв’язки між сутностями можна описати так: один користувач може бути учасником багатьох груп через GroupMembers; одна група має багато навчальних років; один навчальний рік має багато семестрів; один семестр має багато предметів і розкладів; один предмет має багато занять, завдань, файлів, нотаток і питань; одне завдання може мати багато файлів і нотаток; одне оголошення може мати багато коментарів і прикріплених файлів;

одне питання може мати багато відповідей і реакцій.

Для забезпечення коректності даних у базі потрібно використовувати первинні ключі, зовнішні ключі, індекси та обмеження цілісності. Первинні ключі ідентифікують записи, зовнішні ключі забезпечують зв'язки між таблицями, а індекси пришвидшують пошук і фільтрацію. Наприклад, індекси доцільні для Email, UserName, GroupId, SubjectId, DeadlineUtc, StartAtUtc, CreatedAtUtc і полів, які часто використовуються для сортування або пошуку.

У базі також потрібно передбачити обмеження для недопущення некоректних даних. Наприклад, електронна пошта та ім'я користувача повинні бути унікальними, назви основних сутностей не повинні бути порожніми, дата завершення семестру не може бути меншою за дату початку, час завершення заняття або події повинен бути більшим за час початку, а розмір файлу не може бути від'ємним.

Отже, база даних системи будується навколо користувача, академічної групи, навчального року, семестру та предмету. Саме предмет виступає центральною навчальною сутністю, яка об'єднує розклад, завдання, файли, нотатки та Q&A. Фактична доменна модель також містить окремі сутності для оголошень, коментарів, відповідей, реакцій, сповіщень і refresh token-ів. Така структура забезпечує цілісне зберігання навчальної інформації, контроль доступу та можливість подальшого розвитку системи.

Проєктування API є важливим етапом планування клієнт-серверної системи, оскільки саме серверний API забезпечує взаємодію між клієнтськими застосунками, базою даних і бізнес-логікою системи. У розроблюваному проєкті API використовується як єдина точка доступу до навчальних даних для Android/Windows застосунку на .NET MAUI та веб-версії на Blazor. Завдяки цьому всі клієнти працюють з однаковими правилами доступу, структурою даних і логікою обробки запитів. А групування endpoint-ів робить роботу з API зручнішою.

Серверна частина системи реалізується на основі ASP.NET Core Web API. Клієнтські застосунки не звертаються до бази даних напряму, а

надсилають HTTP-запити до відповідних endpoint-ів. Після отримання запиту сервер перевіряє автентифікацію користувача, визначає його права доступу, виконує потрібну бізнес-операцію та повертає результат клієнту. Такий підхід дозволяє централізовано контролювати доступ до навчальної інформації, файлів, групових матеріалів, розкладу, завдань і Q&A-простору.

API системи має REST-подібну структуру і використовує імена для методів надсилання запитів. Для створення ресурсів використовується метод POST, для отримання даних – GET, для оновлення – PUT або PATCH, а для видалення – DELETE. Основним форматом обміну даними між клієнтом і сервером є JSON. Для зручності роботи з API було вирішено розбити endpoint-и на групи (табл. 2.1) з подібними елементами. Загалом групи збігаються з функціональними модулями застосунку. Беручи до уваги те, що для кожного модуля застосунку треба реалізувати мінімум операції створення, оновлення та видалення, поділ на групи дозволяє організувати endpoint-и і тим самим спростити процес розробки і подальшої роботи з API. Кожен endpoint тепер є зрозумілим і читабельним.

Таблиця 2.1. Групи API з їх призначенням та endpoint-ами

Група API	Призначення	Приклади endpoint-ів
Auth	Реєстрація, вхід, оновлення токена, вихід із системи	/api/auth/register, /api/auth/login, /api/auth/refresh
Groups	Створення груп, доєднання, керування.	/api/groups, /api/groups/my, /api/groups/{groupId}/members
Academic structure	Робота з навчальними роками, семестрами та предметами	/api/groups/{groupId}/academic-years, /api/semesters/{semesterId}/subjects

Продовження табл. 2.1

Група API	Призначення	Приклади endpoint-ів
Schedules	Створення розкладу та керування заняттями	/api/semesters/{semesterId}/schedules, /api/schedules/{scheduleId}/items
Tasks	Створення, перегляд, редагування завдань і зміна статусу	/api/subjects/{subjectId}/tasks, /api/tasks/{taskId}/status
Files	Завантаження, видалення файлів і доступ до матеріалів	/api/files/upload, /api/files/{fileId}/download
Notes	Робота з особистими нотатками користувача	/api/notes, /api/notes/my, /api/notes/search
Events	Створення особистих і групових подій	/api/events, /api/events/my, /api/events/group/{groupId}
Announcements	Оголошення групи та коментарі до них	/api/groups/{groupId}/announcements, /api/announcements/{postId}/comments
Q&A	Питання, відповіді та реакції користувачів	/api/questions, /api/questions/{questionId}/answers, /api/questions/{questionId}/reaction

Продовження табл. 2.1

Група API	Призначення	Приклади endpoint-ів
Dashboard	Зведені дані для головної сторінки	/api/dashboard
Notifications	Отримання та позначення сповіщень як прочитаних	/api/notifications, /api/notifications/{id}/read

Окрему увагу потрібно приділити авторизації. Після входу в систему користувач отримує access token, який надсилається в заголовку запиту до захищених endpoint-ів. Сервер перевіряє цей токен і визначає користувача, від імені якого виконується дія. Для продовження сесії використовується refresh token. Такий підхід дозволяє не зберігати пароль на клієнті та забезпечує контрольований доступ до захищених ресурсів.

Як приклад деталізації API можна розглянути модуль завдань (табл. 2.2). Він охоплює створення завдання в межах предмету, отримання списку завдань, перегляд конкретного запису, редагування, зміну статусу та видалення.

Таблиця 2.2. Endpoint-и функціоналу пов'язаного з завданнями.

HTTP-метод	Endpoint	Призначення
POST	/api/subjects/{subjectId}/tasks	Створення завдання для предмету
GET	/api/subjects/{subjectId}/tasks	Отримання завдань предмету
GET	/api/tasks/{taskId}	Перегляд конкретного завдання
PUT	/api/tasks/{taskId}	Редагування завдання
PATCH	/api/tasks/{taskId}/status	Зміна статусу завдання
DELETE	/api/tasks/{taskId}	Видалення завдання

Такий приклад демонструє загальний принцип побудови endpoint-ів у

системі. Якщо ресурс належить до іншої сутності, його створення або отримання може виконуватися через вкладений маршрут. Наприклад, завдання створюється в межах конкретного предмету, тому використовується маршрут /api/subjects/{subjectId}/tasks. Водночас операції з уже створеним завданням виконуються через його власний ідентифікатор: /api/tasks/{taskId}.

Для обміну даними між клієнтом і сервером доцільно використовувати DTO-моделі. Вони відокремлюють внутрішню доменну модель від зовнішнього API-контракту. Наприклад, клієнт не повинен напряму працювати з усіма полями сутності бази даних або передавати службову інформацію, яку має визначати сервер. DTO дозволяють передавати лише ті дані, які потрібні для конкретної операції: створення завдання, редагування предмету, завантаження файлу або відображення списку подій.

API повинен повертати зрозумілі HTTP-коди відповідей. Успішне створення ресурсу може повертати код 201 Created, успішне отримання або оновлення – 200 OK, видалення без тіла відповіді – 204 No Content. Для помилок доцільно використовувати 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found і 500 Internal Server Error. Завдяки цьому клієнтські застосунки можуть коректно обробляти результати запитів і показувати користувачу зрозумілі повідомлення.

Інтерфейс користувача системи має бути простим, зрозумілим і зручним для щоденного використання студентами. Оскільки застосунок призначений для Android, Windows і Web, під час проєктування потрібно враховувати різні розміри екранів, способи введення та сценарії роботи користувача. Основними екранами системи є сторінка входу, головна сторінка, розклад, предмети, завдання, файлове сховище, нотатки, події, група, оголошення, Q&A-простір і налаштування. Головна сторінка повинна відображати найважливішу інформацію: найближчі заняття, актуальні дедлайни, події та нові оголошення. Це дозволяє студенту швидко оцінити поточне навчальне навантаження після відкриття застосунку.

Навігацію доцільно організувати через основні розділи системи:

розклад, предмети, завдання, файли, події та групові модулі. У мобільній версії варто використовувати компактну нижню або бокову навігацію, а у Windows- і Web-версії – бічне меню, яке краще використовує простір великого екрана. Інтерфейс повинен візуально розділяти різні типи навчальної інформації. Предмети можуть мати власні кольори, завдання – статуси виконання, події – позначення особистих або групових активностей, а файли – тип і рівень доступу. Це спрощує орієнтацію користувача та зменшує навантаження під час роботи з великою кількістю даних. На сторінці предмету потрібно об’єднати пов’язані навчальні матеріали: розклад занять, завдання, файли, нотатки, події та Q&A. Такий підхід дозволяє користувачу не шукати інформацію в різних частинах системи, а працювати з усіма даними конкретної дисципліни в одному місці.

Також важливо передбачити адаптивність, світлу й темну тему, зрозумілі повідомлення про помилки та підтвердження важливих дій, наприклад видалення файлу або зміни групового розкладу. Усі основні дії мають виконуватися за мінімальну кількість кроків.

Отже, інтерфейс системи повинен забезпечувати швидкий доступ до навчальної інформації, зручну навігацію між модулями, візуальне розділення даних і коректну роботу на мобільних, десктопних і веб-платформах.

Отже, у другому розділі було виконано планування кросплатформового застосунку для організації навчального процесу студентів. Сформовано загальну концепцію системи, яка передбачає взаємодію Android/Windows-клієнта, веб-версії, серверного API, бази даних і файлового сховища. Така структура дозволяє об’єднати в однму середовищі роботу з розкладом, предметами, завданнями, файлами, нотатками, подіями та груповою взаємодією.

У межах розділу визначено цільову аудиторію, основні сценарії використання, функціональні та нефункціональні вимоги. Також розглянуто ролі користувачів і права доступу, структуру навчальних даних, основні модулі системи, базу даних, API та інтерфейс користувача

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ОРГАНІЗАЦІЇ НАВЧАЛЬНОЇ ДІЯЛЬНОСТІ

3.1. Структура програмного рішення

Програмне рішення StudentOrganizer реалізовано як клієнт-серверну систему, що складається з серверного API, спільної доменної та прикладної логіки, інфраструктурного шару, мобільно-десктопного клієнта, веб-клієнта та окремого тестового проєкту. Такий поділ дозволяє розділити відповідальність між частинами системи: сервер відповідає за обробку запитів і збереження даних, клієнтські застосунки забезпечують взаємодію з користувачем, а спільні проєкти містять моделі, сервіси та контракти, які використовуються кількома частинами системи[5].

У корені рішення використовується файл StudentOrganizer.slnx, у якому всі проєкти згруповано у дві основні папки: src та tests (рис. 3.1). У папці src розміщено основні проєкти застосунку, а в папці tests – проєкт для автоматизованої перевірки роботи системи.

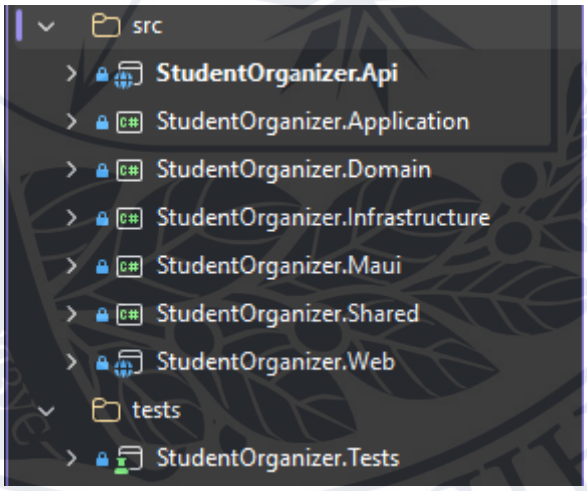

Рисунок 3.4 – Структура рішення

Проєкт StudentOrganizer.Api є серверною частиною системи. Він реалізований як ASP.NET Core Web API та містить контролери для обробки

HTTP-запитів. У цьому проєкті розміщено контролери авторизації, груп, навчальної структури, розкладу, завдань, файлів, нотаток, подій, оголошень, питань, сповіщень і головної сторінки. Саме через API клієнтські застосунки отримують дані, створюють нові записи, оновлюють інформацію та виконують дії, які потребують доступу до серверної бази даних. У Program.cs налаштовуються контролери, Entity Framework Core, JWT-автентифікація, авторизація, Swagger/OpenAPI-документація та реєстрація сервісів через dependency injection.

Проєкт StudentOrganizer.Application містить прикладну логіку системи. У ньому розміщено сервіси, які виконують основні операції над даними: роботу з групами, навчальною структурою, розкладом, завданнями, файлами, нотатками, подіями, оголошеннями, Q&A-модулем, сповіщеннями та Dashboard[7]. Контролери API не виконують складну бізнес-логіку напряму, а передають запити до відповідних сервісів цього шару. Це зменшує зв’язаність між HTTP-рівнем і логікою застосунку та полегшує подальше тестування окремих модулів.

Проєкт StudentOrganizer.Domain містить доменну модель системи. До нього винесено сутності предметної області, зокрема User, Group, GroupMember, AcademicYear, Semester, Subject, Schedule, ScheduleItem, StudentTask, FileItem, FileShare, Note, CalendarEvent, AnnouncementPost, Question, Notification та інші класи. Також у цьому проєкті розміщено переліки, які описують фіксовані стани та типи даних: ролі учасників групи, статуси й типи завдань, типи тижнів, типи занять, типи подій, області використання файлів і статуси питань. Винесення доменної моделі в окремий проєкт робить її незалежною від API, бази даних і клієнтського інтерфейсу.

Проєкт StudentOrganizer.Infrastructure відповідає за роботу з інфраструктурними механізмами. У ньому реалізовано AppDbContext, міграції Entity Framework Core, налаштування таблиць бази даних і локальне файлове сховище. Серверна база даних використовується для збереження користувачів, груп, предметів, розкладу, завдань, нотаток, подій, оголошень, питань,

сповіщень і метаданих файлів. Фізичні файли зберігаються окремо у файловій системі в папці storage, а в базі даних зберігається службова інформація про файл: назва, шлях, тип, розмір, власник і зв'язок із відповідним модулем.

Проєкт StudentOrganizer.Shared містить спільні DTO-моделі, request-моделі, response-моделі та допоміжні типи, які використовуються одночасно сервером, MAUI-клієнтом і веб-клієнтом. Наприклад, у цьому проєкті розміщено моделі для авторизації, груп, розкладу, завдань, файлів, нотаток, подій, оголошень, питань і Dashboard. Такий підхід дозволяє не дублювати однакові класи в різних частинах системи. API повертає клієнтам не доменні сутності напряму, а спеціальні response-моделі, які містять тільки потрібні для інтерфейсу дані.

Проєкт StudentOrganizer.Maui є клієнтським застосунком для Android та Windows. Він реалізує інтерфейс користувача за допомогою .NET MAUI та містить сторінки входу, режстрації, головної сторінки, груп, розкладу, предметів, завдань, файлів, нотаток, подій, оголошень, питань і налаштувань. Для взаємодії з сервером використовується ApiClient, який надсилає HTTP-запити до API та додає JWT-токен до захищених запитів. Також у MAUI-проєкті реалізовано збереження токенів, локальний кеш, клієнтські сервіси та ViewModel-класи для зв'язку між інтерфейсом і даними.

Проєкт StudentOrganizer.Web реалізує веб-версію системи на основі Blazor Web App. Він містить Razor-компоненти, сторінки, layout-и, бокове меню, верхню панель і клієнтський сервіс StudentOrganizerApiClient для роботи з серверним API. Веб-застосунок не використовує окрему локальну базу даних, а отримує дані через API. Завдяки цьому MAUI-клієнт і Blazor Web App працюють з одним серверним джерелом даних і відображають узгоджену інформацію.

Окремо в рішенні передбачено проєкт StudentOrganizer.Tests. Він використовується для unit-тестів та integration-тестів. У тестах перевіряється робота прикладних сервісів, захищених API-запитів, авторизації, ролей, файлового сховища та основних навчальних модулів. Для integration-

тестування використовується окреме тестове оточення, що дозволяє перевіряти поведінку системи без впливу на основну базу даних.

Заальна архітектура застосунку (рис. 3.2) побудована за принципом розділення відповідальності між шарами. Користувач працює з MAUI або Blazor-клієнтом. Клієнт надсилає HTTP-запит до контролера API. Контролер приймає запит, перевіряє авторизацію та передає дані до сервісу прикладного шару. Сервіс виконує бізнес-логіку, звертається до AppDbContext або файлового сховища, після чого повертає результат у вигляді DTO-моделі. Далі API надсилає відповідь клієнту, а клієнтський застосунок оновлює інтерфейс.

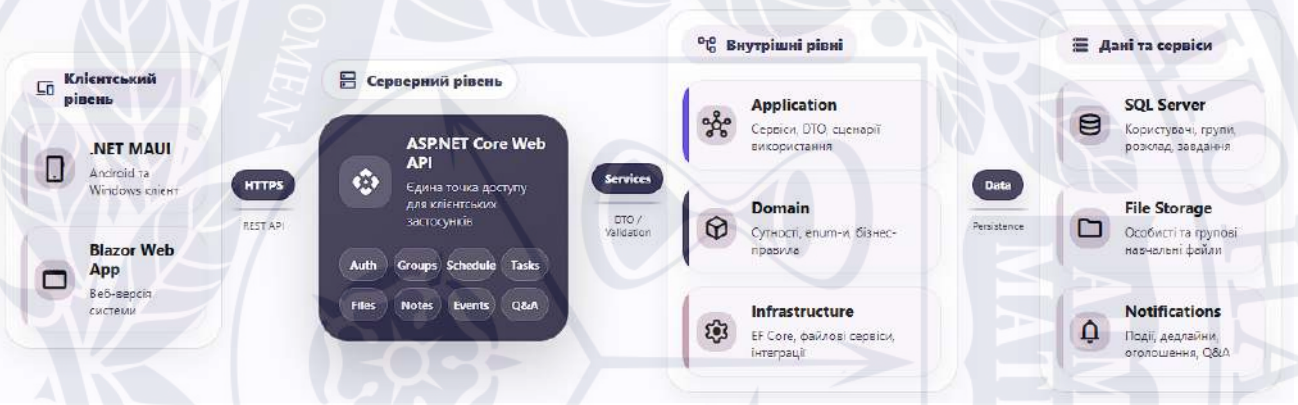

Рисунок 3.2 – Архітектура застосунку

Така структура є зручною для розробки великого застосунку, оскільки кожен модуль має чітке місце в системі. Доменні сутності описують предметну область, сервіси реалізують правила роботи застосунку, інфраструктурний шар відповідає за збереження даних, API забезпечує доступ до функцій системи, а MAUI та Blazor-клієнти надають користувачеві різні способи роботи із застосунком. Завдяки цьому система може поступово розширюватися без повної перебудови архітектури.

Доменна модель застосунку винесена в окремий проєкт StudentOrganizer.Domain. У цьому проєкті зосереджено основні сутності предметної області, які описують користувачів, академічні групи, навчальні роки, семестри, предмети, розклад, завдання, файли, нотатки, події, оголошення, питання, відповіді та сповіщення. Такий підхід дозволяє

відокремити опис предметної області від серверного API, інфраструктурного шару та клієнтських застосунків[21].

Більшість класів доменної моделі наслідують базовий клас Entity, у якому визначено спільні поля для всіх таблиць: унікальний ідентифікатор Id, дату створення CreatedAtUtc та дату останнього оновлення UpdatedAtUtc. Це спрощує структуру сутностей, оскільки однакові службові властивості не дублюються в кожному класі окремо.

Основою системи є сутності User, Group і GroupMember. Клас User описує користувача системи та містить дані для автентифікації, особисту інформацію, а також навігаційні зв’язки з нотатками, файлами, подіями, завданнями, питаннями, відповідями, сповіщеннями та refresh-токенами. Сутність Group описує академічну групу, яка має назву, опис, код приєднання та колекції пов’язаних навчальних даних. Зв’язок між користувачем і групою реалізовано через GroupMember, де додатково зберігається роль користувача в групі. Для ролей використовується перелік GroupRole, який містить значення Owner, Admin, Moderator і Student.

Навчальна структура реалізована через послідовність сутностей AcademicYear, Semester і Subject. Навчальний рік належить певній групі та містить семестри. Семестр належить навчальному року та об’єднує предмети й розклади. Предмет є центральною навчальною одиницею, до якої можуть бути прив’язані завдання, заняття розкладу, файли, нотатки та питання. У класі Subject також передбачено поля для імені викладача, контактів, опису та кольору предмету, що використовується для візуального розділення дисциплін в інтерфейсі.

Модуль розкладу представлений сутностями Schedule і ScheduleItem. Schedule належить групі та семестру, а ScheduleItem описує конкретне заняття. Для заняття зберігаються назва, день тижня, час початку і завершення, предмет, тип тижня, тип заняття, аудиторія, викладач і коментар. Тип тижня задається через WeekType, де передбачено щотижневе заняття, верхній тиждень і нижній тиждень. Тип заняття задається через LessonType: лекція,

практика, лабораторна, семінар, консультація, екзамен або інше.

Завдання студентів реалізовано сутністю StudentTask. Вона пов’язана з предметом і користувачем, який створив завдання. Для завдання зберігаються назва, опис, тип, статус і дедлайн. Тип завдання визначається через TaskType, де передбачено лабораторну, практичну, домашню, курсову, модульну, екзаменаційну роботу та інший тип. Поточний стан виконання задається через TaskStatus: не розпочато, у процесі, виконано або прострочено. До завдання можуть бути прикріплені файли та нотатки.

Файловий модуль у доменній моделі представлений класами FileItem і FileShare. FileItem зберігає метадані файлу: початкову назву, службову назву у сховищі, шлях до файлу, тип вмісту, розмір, власника та область використання. Фізичний файл зберігається у файловій системі сервера, а в базі даних зберігаються тільки його метадані. Область використання файлу визначається через FileScope: особистий файл, особистий файл предмету, груповий файл предмету, файл завдання або вкладення до оголошення. Сутність FileShare використовується для надання доступу до файлу іншому користувачу та містить інформацію про права читання, зміни й можливий строк дії доступу.

Нотатки реалізовано через сутності Note і NoteTag. Нотатка належить користувачу та може бути додатково пов’язана з предметом або завданням. Для класифікації нотаток використовується набір тегів, які зберігаються в окремій таблиці NoteTags. Події реалізовано класом CalendarEvent. Подія має власника, може бути особистою або груповою, може прив’язуватися до групи та предмету, а також містить назву, опис, час початку, час завершення, тип і місце проведення.

Групова взаємодія представлена сутностями AnnouncementPost, AnnouncementComment, Question, QuestionAnswer і QuestionReaction. Оголошення належить групі, має автора, заголовок, текст, ознаку закріплення, коментарі та вкладення. Q&A-модуль побудовано на сутності Question, яка може належати групі або предмету, має автора, заголовок, текст, статус, ознаку вирішення та лічильник реакцій. Відповіді зберігаються в QuestionAnswer, а

реакції користувачів – у QuestionReaction. Для статусу питання використовується QuestionStatus: відкрите, з відповіддю, вирішене або закрите.

Робота з базою даних реалізована в проєкті StudentOrganizer.Infrastructure через клас AppDbContext [Додаток А]. У ньому оголошено DbSet для основних таблиць системи: Users, Groups, GroupMembers, AcademicYears, Semesters, Subjects, Schedules, ScheduleItems, Tasks, Files, FileShares, Notes, NoteTags, Events, AnnouncementPosts, AnnouncementComments, Questions, QuestionAnswers, QuestionReactions, Notifications і RefreshTokens. AppDbContext також реалізує інтерфейс IApplicationDbContext, завдяки чому прикладні сервіси працюють не з конкретною реалізацією контексту, а з абстракцією.

У методі OnModelCreating виконано конфігурацію таблиць, зв’язків, індексів, обмежень і початкових демонстраційних даних. Для текстових полів задано максимальну довжину, для обов’язкових полів – вимогу IsRequired, а для enum-значень використано перетворення у рядок. Такий спосіб збереження робить значення в базі даних зрозумілішими, оскільки замість чисел зберігаються назви станів і типів. Наприклад, для ролі учасника групи в базі зберігається Owner, Admin або Student, а не числовий код.

Для підтримки цілісності даних використано зовнішні ключі, унікальні індекси та check constraints. Наприклад, електронна пошта й ім’я користувача мають унікальні індекси, код групи також є унікальним, а пара UserId і GroupId у таблиці учасників групи не може дублюватися. Для розкладу перевіряється, що час завершення заняття більший за час початку, для семестру – що дата завершення не менша за дату початку, а для файлів – що розмір не є від’ємним.

Зв’язки між сутностями налаштовано переважно з поведінкою DeleteBehavior.Restrict. Це зроблено для того, щоб уникнути випадкового каскадного видалення важливих навчальних даних. Наприклад, видалення користувача або групи не повинно автоматично видалити всі пов’язані предмети, завдання, файли, оголошення та питання без додаткової перевірки

на рівні бізнес-логіки.

Для створення і зміни структури бази даних використовуються міграції Entity Framework Core. У проєкті наявна початкова міграція InitialCreate, а також подальші міграції для додавання refresh-токенів, деталей занять, автора завдання, файлових метаданих, зв'язку нотаток із завданнями, коментарів до оголошень, реакцій у Q&A-модулі та перейменування лічильника реакцій. Основним серверним сховищем виступає SQL Server LocalDB з базою StudentOrganizerDb.

3.2. Реалізація автентифікації, авторизації та ролей користувачів

У серверній частині застосунку реалізовано механізм автентифікації та авторизації користувачів. Автентифікація відповідає за перевірку особи користувача під час реєстрації, входу в систему та оновлення токена доступу. Авторизація визначає, чи має вже автентифікований користувач право виконувати певну дію в межах системи або конкретної академічної групи.

Основна логіка автентифікації розміщена в AuthController та AuthService. Контролер відповідає за приймання HTTP-запитів, а сервіс виконує перевірки, створення користувача, генерацію токенів і роботу з базою даних. Такий поділ дозволяє не перевантажувати контролер бізнес-логікою. Контролер лише отримує request-модель, передає її до сервісу та повертає клієнту відповідь у вигляді AuthResponse або повідомлення про помилку.

У AuthController реалізовано основні endpoint-и для роботи з обліковим записом: register, login, refresh, logout і me. Endpoint-и реєстрації, входу й оновлення токена доступні без попередньої авторизації, тому для них використовується атрибут [AllowAnonymous]. Endpoint-и logout і me захищені атрибутом [Authorize], оскільки вони мають працювати тільки для користувача, який уже має дійсний JWT access token.

Під час реєстрації користувач передає електронну пошту, ім'я користувача, пароль, ім'я та прізвище. Перед створенням нового запису виконується перевірка, чи не існує вже користувача з такою електронною

поштою або іменем користувача. Пароль не зберігається в базі даних у відкритому вигляді. Для нього використовується PasswordHasher<User>, який формує хеш пароля. У таблиці користувачів зберігається тільки результат хешування, що підвищує безпеку системи.

Під час входу в систему користувач може використати електронну пошту або ім’я користувача (рис. 3.3). AuthService знаходить відповідний запис у базі даних і перевіряє пароль через VerifyHashedPassword. Якщо дані правильні, сервер створює пару токенів: JWT access token і refresh token. Access token використовується для доступу до захищених endpoint-ів, а refresh token – для отримання нового access token без повторного введення логіна й пароля.

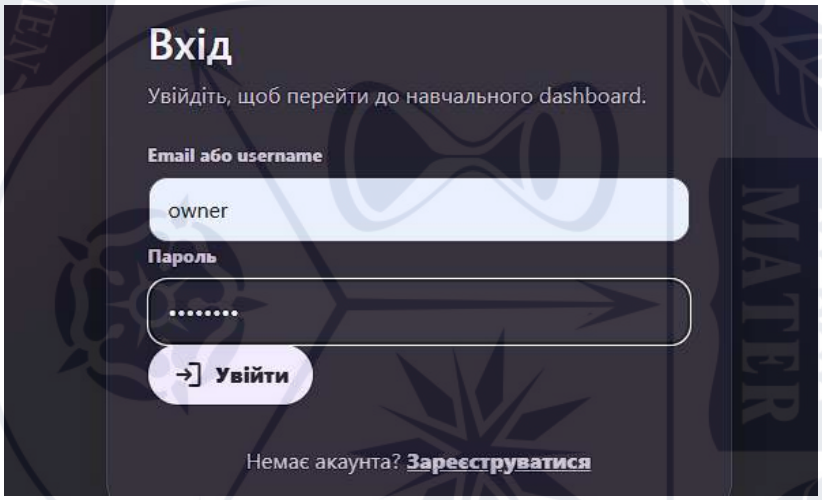

Рисунок 3.3 – Форма для входу

JWT access token створюється на основі налаштувань JwtOptions. У конфігурації задаються issuer, audience, секретний ключ, тривалість дії access token і тривалість дії refresh token. Access token підписується алгоритмом HmacSha256 і містить claims з ідентифікатором користувача, електронною поштою та іменем користувача. На рівні Program.cs налаштовано перевірку issuer, audience, ключа підпису та строку дії токена. Це дозволяє серверу відхиляти підроблені або прострочені токени.

Refresh token реалізовано як окрема сутність RefreshToken. На відміну від access token, клієнт отримує випадково згенероване значення refresh token,

а в базі даних зберігається тільки його SHA-256 хеш. Це зроблено для того, щоб у разі витоку бази даних не зберігати активні refresh tokens у відкритому вигляді. Refresh token має дату завершення дії, дату відкликання та поле для збереження хешу нового токена, який його замінив.

Під час оновлення токена сервер отримує refresh token від клієнта, хешує його та шукає відповідний запис у базі даних. Якщо токен існує, не відкликаний і не прострочений, система створює новий access token і новий refresh token. Старий refresh token відкликається, а новий запис додається в базу даних. Такий підхід зменшує ризик повторного використання старого refresh token.

Захист endpoint-ів реалізовано за допомогою атрибута [Authorize]. Більшість контролерів основних модулів доступні тільки авторизованим користувачам. Це означає, що клієнтський застосунок має передавати access token у заголовку HTTP-запиту. Якщо токен відсутній, неправильний або прострочений, сервер повертає відповідь про відсутність доступу.

Окремо в системі реалізовано ролі користувачів у межах академічної групи (лістинг 3.1). Роль не є глобальною для всієї системи, а зберігається в сутності GroupMember, тобто один і той самий користувач може мати різні ролі в різних групах. Наприклад, в одній групі він може бути адміністратором, а в іншій – звичайним учасником. Лістинг Енум для позначення ролі користувача у групі має вигляд:

```
public enum GroupRole
{
    Owner = 1,
    Admin = 2,
    Moderator = 3,
    Student = 4,
}
```

Роль Owner призначається користувачу, який створив групу. Він має

найвищий рівень доступу та може керувати учасниками. Admin має розширені права для адміністрування групи, але не повинен мати можливість повністю замінити власника. Moderator використовується для користувачів, які можуть допомагати з керуванням контентом, наприклад оголошеннями, завданнями або Q&A. Student є базовою роллю звичайного учасника групи.

Перевірка прав доступу виконується не тільки на рівні контролерів, а й у сервісах прикладної логіки. Це важливо, оскільки [Authorize] лише підтверджує, що користувач увійшов у систему, але не визначає, чи має він право редагувати конкретну групу, розклад, завдання, файл або відповідь у Q&A. Тому сервіси додатково перевіряють роль користувача в групі перед виконанням дії.

Таким чином, у системі поєднано два рівні захисту. Перший рівень – це JWT-автентифікація, яка підтверджує особу користувача та захищає API від неавторизованих запитів. Другий рівень – це рольова авторизація в межах групи, яка визначає, які саме дії може виконувати користувач після входу в систему. Такий підхід відповідає клієнт-серверній структурі застосунку та забезпечує контроль доступу до навчальних даних, файлів і групових функцій.

Серверна частина застосунку реалізована у проєкті StudentOrganizer.Api як ASP.NET Core Web API. Вона відповідає за приймання HTTP-запитів від MAUI-клієнта та Blazor Web App, перевірку автентифікації, передавання даних до сервісів прикладного шару та повернення результатів клієнтам. Основна бізнес-логіка не розміщується безпосередньо в контролерах, а винесена в проєкт StudentOrganizer.Application. Такий підхід дає змогу розділити HTTP-рівень і правила роботи системи.

Загальний принцип роботи API побудований за єдиною схемою. Клієнтський застосунок надсилає HTTP-запит до контролера. Контролер отримує поточного користувача з JWT claims, приймає request-модель із проєкту StudentOrganizer.Shared і передає її до відповідного application-сервісу. Сервіс виконує перевірки, працює з IApplicationDbContext, звертається до доменних сутностей і повертає результат у вигляді response-моделі. Після

цього контролер перетворює результат виконання в HTTP-відповідь зі статусом успіху або помилки.

Більшість контролерів захищені атрибутом [Authorize], тому доступ до основних функцій мають тільки авторизовані користувачі. У контролерах використовується допоміжна логіка отримання ідентифікатора поточного користувача з claim ClaimTypes.NameIdentifier. Після цього userId передається в application-сервіс, де виконується перевірка прав доступу до конкретної групи, предмету, розкладу, завдання, оголошення або питання.

Для уніфікації відповідей application-сервіси повертають об’єкт ApplicationResult<T>. Він містить ознаку успішності операції, значення результату, текст помилки та HTTP-статус. Це дозволяє однаково обробляти успішні й помилкові сценарії в різних модулях. Наприклад, якщо користувач не є учасником групи, сервіс повертає помилку доступу, а контролер формує відповідь із відповідним HTTP-статусом.

Модуль груп реалізовано через GroupsController і GroupService. Він забезпечує створення академічної групи, отримання списку груп поточного користувача, перегляд конкретної групи, приєднання за кодом, перегляд учасників, зміну ролі учасника та видалення учасника з групи. Під час створення групи користувач автоматично стає її власником. Під час приєднання за кодом користувач додається до групи з роллю Student.

У GroupService також реалізовано перевірку прав для керування ролями. Власник групи має найвищий рівень доступу, а адміністратор може керувати тільки користувачами нижчих ролей. Це не дозволяє звичайному адміністратору змінити роль власника або призначити собі вищі права. Також у сервісі передбачено обмеження відображення коду приєднання: його можуть бачити тільки користувачі з відповідними правами.

Навчальна структура реалізована через StudyStructureController і StudyStructureService. Цей модуль відповідає за навчальні роки, семестри та предмети. API дозволяє створювати навчальні роки для групи, отримувати список навчальних років, створювати семестри, переглядати семестри

навчального року, отримувати поточний семестр, створювати й редагувати предмети. Також передбачено архівування семестру, що дозволяє зберігати завершені навчальні періоди без їх видалення.

У StudyStructureService виконуються перевірки, чи має користувач право керувати навчальною структурою. Для створення або редагування навчального року, семестру чи предмету користувач повинен мати роль Owner або Admin. Для перегляду даних достатньо бути учасником групи. Також сервіс перевіряє, чи не належать зміни до архівного семестру або архівного навчального року, оскільки такі дані не повинні редагуватися як активні.

Модуль розкладу реалізовано через SchedulesController і ScheduleService. API підтримує створення розкладу для семестру, отримання розкладів семестру, встановлення активного розкладу, створення занять, перегляд усіх занять розкладу, перегляд занять на конкретний день або тиждень, редагування й видалення занять. У запитах до розкладу використовуються параметри для визначення типу тижня, що дає змогу відображати заняття для звичайного, верхнього або нижнього тижня.

У ScheduleService бізнес-логіка не обмежується простим збереженням записів у базі даних. Сервіс перевіряє доступ користувача до групи, стан семестру, коректність часу заняття, зв’язок заняття з предметом і можливість редагування розкладу. Для читання розкладу достатньо бути учасником відповідної групи, а для створення, редагування чи видалення занять потрібні розширені права.

Модуль завдань реалізовано через TasksController і TaskService. Він забезпечує створення завдання для предмету, отримання завдань конкретного предмету, перегляд особистого списку завдань користувача, отримання найближчих дедлайнів, перегляд конкретного завдання, редагування, зміну статусу та видалення. Для фільтрації використовуються параметри запиту, наприклад статус завдання або ідентифікатор предмету.

У TaskService перевіряється, чи має користувач доступ до предмету, для якого створюється або переглядається завдання. Також сервіс контролює зміну

статусу завдання. Наприклад, студент може оновити стан виконання доступного йому завдання, але не повинен отримувати доступ до завдань групи, учасником якої він не є. Для завдань використовуються DTO-моделі зі спільного проєкту, тому MAUI й Blazor-клієнт отримують однакову структуру даних.

Модуль нотаток реалізовано через NotesController і NoteService. Він працює з особистими нотатками користувача. API дозволяє створювати нотатку, отримувати список власних нотаток, переглядати конкретну нотатку, виконувати пошук, редагувати та видаляти записи. Нотатка може бути пов’язана з предметом або завданням, а також мати набір тегів. Це дозволяє використовувати нотатник не як ізольований текстовий блок, а як частину навчальної структури.

Модуль подій реалізовано через EventsController і EventService. Він підтримує створення особистих і групових подій, перегляд власних подій, перегляд подій групи, отримання найближчих подій, редагування й видалення. Подія може бути прив’язана до групи або предмету. Для групових подій сервіс перевіряє, чи є користувач учасником відповідної групи, а також чи має він право змінювати подію.

Головна сторінка застосунку формується через DashboardController і DashboardService. Цей модуль не створює окрему предметну сутність, а агрегує дані з інших частин системи. Dashboard повертає клієнту стислий набір інформації, потрібний для стартового екрана: групи користувача, найближчі завдання, майбутні події, поточний розклад, останні оголошення та інші важливі дані. Завдяки цьому клієнтський застосунок не повинен виконувати багато окремих запитів для формування головної сторінки.

Групові оголошення реалізовано через AnnouncementsController і AnnouncementService. API дозволяє створювати оголошення в межах групи, отримувати список оголошень групи, переглядати конкретний запис, редагувати й видаляти його. Також реалізовано коментарі до оголошень: користувачі можуть додавати коментарі, переглядати їх і видаляти власні або

доступні для модерації коментарі.

У AnnouncementService перевіряється роль користувача в групі. Створення та редагування оголошень доступне не всім учасникам, оскільки цей модуль призначений для важливих групових повідомлень. Звичайні студенти можуть переглядати оголошення та коментувати їх, але керування публікаціями має бути доступним користувачам з вищими правами.

Q&A-модуль реалізовано через QuestionsController і QuestionService. Він підтримує створення питання, отримання питань групи, отримання питань конкретного предмету, перегляд питання, редагування, видалення, створення відповідей, редагування й видалення відповідей, а також додавання й видалення реакцій. Такий набір операцій дозволяє використовувати модуль як простір для навчальних питань, де студенти можуть позначати зацікавленість, а адміністратори або модератори можуть надавати відповіді.

У QuestionService реалізовано перевірку доступу до групи або предмету, з яким пов’язане питання. Для реакцій враховується користувач, який їх створив, щоб уникнути некоректного дублювання. Статуси питання дають змогу відрізняти відкриті питання, питання з відповіддю, вирішені та закриті питання. Це полегшує сортування й подальше відображення Q&A-простору в клієнтському інтерфейсі.

Модуль сповіщень реалізовано через NotificationsController і NotificationService. Він забезпечує отримання списку сповіщень поточного користувача, позначення окремого сповіщення як прочитаного та позначення всіх сповіщень як прочитаних. У межах поточної реалізації це in-app сповіщення, які зберігаються в базі даних і можуть відображатися в клієнтських застосунках. Повноцінні push-сповіщення можна розглядати як перспективу розвитку.

Файловий модуль має власний контролер FilesController і сервіс FileService, однак детальніше він розглядається в окремому підпункті 3.5, оскільки файлове сховище є важливою частиною теми роботи. У межах API він використовується іншими модулями для прив’язки файлів до предметів,

завдань і оголошень.

Усі основні модулі використовують спільні DTO з проєкту StudentOrganizer.Shared. Для створення й оновлення даних застосовуються request-моделі, а для повернення інформації клієнтам – response-моделі. Це зменшує дублювання класів між API, MAUI та Blazor Web App. Крім того, DTO не розкривають внутрішню структуру доменних сутностей напряму, а повертають тільки ті поля, які потрібні клієнтському інтерфейсу.

Серверний API побудовано за REST-підходом. Для створення даних використовуються POST-запити, для отримання – GET, для повного оновлення – PUT, для часткової зміни стану – PATCH, для видалення – DELETE. Наприклад, завдання створюється через запит до предмету, список власних завдань отримується окремим endpoint-ом, а зміна статусу завдання виконується через PATCH-запит. Такий поділ робить API зрозумілим для клієнтських застосунків і спрощує подальшу підтримку.

Для документування API використовується Swagger/OpenAPI. У режимі розробки доступна сторінка Swagger UI, через яку можна переглядати endpoint-и, request-моделі, response-моделі та перевіряти роботу запитів без запуску клієнтського застосунку. Це особливо корисно під час тестування серверної частини та налагодження взаємодії між API, MAUI-клієнтом і веб-версією.

3.3. Реалізація файлового сховища

Файлове сховище є окремим модулем системи, оскільки застосунок передбачає збереження навчальних матеріалів, особистих файлів користувача, файлів предметів, файлів завдань і вкладень до оголошень. У реалізації використано комбінований підхід: фізичні файли зберігаються у файловій системі сервера, а в базі даних зберігаються їхні метадані. Це дозволяє не перевантажувати базу даних великим бінарним вмістом і водночас зберігати повну інформацію для пошуку, відображення та перевірки доступу.

Серверний доступ до файлового модуля реалізовано через

FilesController. Контролер містить endpoint-и для завантаження файлу на сервер, отримання інформації про файл, завантаження файлу користувачем, видалення файлу, отримання файлів предмету та отримання файлів завдання. Усі дії файлового модуля захищені атрибутом [Authorize], тому працювати з файлами можуть тільки автентифіковані користувачі.

Під час завантаження файлу клієнт надсилає multipart/form-data запит, який містить сам файл і додаткові параметри прив'язки. Файл може бути особистим, пов'язаним із предметом, пов'язаним із завданням або прикріпленим до оголошення. Для обмеження надмірно великих запитів у контролері використано ліміт розміру запиту. Далі контролер відкриває потік читання файлу та передає його до FileService.

Основна бізнес-логіка файлового модуля розміщена у FileService. Цей сервіс перевіряє коректність запиту, визначає ціль завантаження, перевіряє права користувача, формує шлях збереження, викликає сервіс фізичного сховища та створює запис про файл у базі даних. Якщо під час збереження метаданих у базі виникає помилка, фізично збережений файл видаляється зі сховища, щоб не залишати зайві файли без запису в базі даних.

Для опису області використання файлу застосовується перелік FileScope. У системі передбачено такі типи файлів: Personal, SubjectPersonal, SubjectGroup, TaskFile і AnnouncementAttachment. Особисті файли належать конкретному користувачу. Особисті файли предмету прив'язані до дисципліни, але доступні тільки власнику. Групові файли предмету доступні учасникам відповідної групи. Файли завдання пов'язані з конкретним навчальним завданням, а вкладення оголошень використовуються для матеріалів, доданих до групових повідомлень.

Метадані файлів зберігаються в таблиці Files. Для кожного файлу зберігаються ідентифікатор власника, початкова назва файлу, службова назва у сховищі, шлях до файлу, MIME-тип, розмір, область використання та необов'язкові зв'язки з групою, предметом, завданням або оголошенням. Завдяки цьому файл можна відобразити в різних частинах застосунку без

дублювання даних.

Фізичне збереження файлів винесено в інтерфейс IFileStorageService. Він описує три основні операції: збереження файлу, відкриття файлу для читання та видалення файлу. Конкретна реалізація цього інтерфейсу – LocalFileStorageService. Вона зберігає файли в локальній папці storage, формує безпечну службову назву файлу, створює потрібні директорії та перевіряє, щоб шлях не виходив за межі кореневої папки сховища.

Структура фізичного сховища формується залежно від типу файлу. Особисті файли й особисті файли предметів зберігаються в директоріях користувачів. Групові файли предметів, файли завдань і вкладення оголошень зберігаються в директоріях відповідних груп. Такий поділ спрощує організацію файлів на сервері та дозволяє логічно відокремити особисті й групові матеріали.

Перевірка доступу до файлів виконується у FileService. Користувач може читати файл, якщо він є власником файлу, має активний доступ через FileShare або є учасником групи, до якої належить груповий файл. Для особистих файлів і особистих файлів предмету доступ обмежується власником. Для групових файлів доступ на читання надається учасникам групи, а завантаження групових файлів предмету або вкладень до оголошень доступне тільки користувачам із відповідними ролями.

Видалення файлу також має окрему перевірку прав. Власник може видалити власний файл. Якщо файл належить груповому простору, видалення може виконати користувач із роллю Owner або Admin. Після успішного видалення запису з бази даних сервіс видаляє фізичний файл зі сховища.

Для повернення інформації клієнтам використовується FileResponse. У ньому передаються ідентифікатор файлу, власник, зв’язки з групою, предметом, завданням або оголошенням, початкова назва, службова назва, тип вмісту, розмір, шлях у сховищі, область використання та дата створення. Сам вміст файлу передається тільки через окремий endpoint завантаження, що відокремлює перегляд метаданих від отримання фізичного файлу.

Клієнтський застосунок для Android та Windows реалізовано в проєкті StudentOrganizer.Maui з використанням .NET MAUI. MAUI-клієнт не працює напряму з серверною базою даних, а отримує всі актуальні дані через HTTP-запити до StudentOrganizer.Api.

Структура MAUI-проєкту поділена на кілька основних частин. У папці Views розміщено сторінки застосунку, у ViewModels – класи для керування станом сторінок, у Services – клієнтські сервіси для роботи з API, токенами, кешем і окремими модулями. Також у проєкті є папки Resources, Platforms, Theming і Models. Такий поділ відповідає MVVM-підходу, де сторінка відповідає за відображення, ViewModel – за стан і команди, а сервіси – за отримання та збереження даних.

У MauiProgram.cs виконується початкове налаштування MAUI-застосунку, шрифтів, логування та dependency injection. Там реєструються всі сторінки, ViewModels і сервіси. Для клієнтських сервісів використовується singleton-реєстрація, а для сторінок і ViewModels – transient-реєстрація. Лістинг реєстрації singleton сервісу має вигляд:

```
builder.Services.AddSingleton(static _ => {
    var baseAddress = DeviceInfo.Platform ==
DevicePlatform.Android
        ? "http://10.0.2.2:5295/"
        : "http://localhost:5295/";
    return new HttpClient {
        BaseAddress = new Uri(baseAddress)
    };});
```

У цьому фрагменті показано вибір базової адреси API залежно від платформи. Для Android-емулятора використовується адреса 10.0.2.2, оскільки localhost усередині емулятора вказує на сам емулятор, а не на комп’ютер, де запущено API. Для Windows-клієнта використовується localhost, оскільки застосунок і сервер можуть запускатися на одному комп’ютері.

Для взаємодії з сервером створено універсальний сервіс ApiClient. Він інкапсулює виконання HTTP-запитів і підтримує основні методи: GET, POST, PUT, PATCH, DELETE і multipart-запити для завантаження файлів. Перед відправленням захищеного запиту ApiClient отримує access token із TokenStorageService і додає його в заголовок Authorization у форматі Bearer.

Збереження токенів реалізовано через TokenStorageService. Для цього використовується SecureStorage, який призначений для збереження чутливих даних на пристрої. У ньому зберігаються access token і refresh token. Під час входу або реєстрації токени записуються в захищене сховище, а під час виходу з облікового запису видаляються.

Реєстрація та вхід користувача виконуються через клієнтський AuthService. Після успішного входу сервіс зберігає токени через TokenStorageService, записує дані користувача в локальний кеш і переводить користувача на сторінку Dashboard. Під час виходу AuthService надсилає запит до API на logout, а потім очищує токени й локальні кешовані дані. Якщо API тимчасово недоступне, локальний вихід усе одно виконується, щоб користувач міг завершити сесію на пристрої.

Для кожного основного модуля створено окремий клієнтський сервіс: GroupService, SubjectService, ScheduleService, TaskService, FileService, NoteService, EventService, AnnouncementService, QuestionService і NotificationService. Ці сервіси не містять складної бізнес-логіки, а відповідають за формування запитів до потрібних endpoint-ів API. Наприклад, TaskService працює із завданнями, ScheduleService – з розкладом, NoteService – з нотатками, а QuestionService – з Q&A-модулем.

ViewModel-класи реалізують стан сторінок і команди користувача. Спільна логіка винесена в BaseViewModel, де є властивості IsBusy і ErrorMessage, а також допоміжний метод для виконання асинхронних дій. Це дає змогу однаково обробляти завантаження, помилки та блокування повторного запуску команди на різних сторінках. Для виконання асинхронних команд використовується AsyncCommand.

Головна сторінка DashboardPage відображає агреговану інформацію для користувача (рис. 3.4). Вона отримує дані з API через відповідний сервіс і показує основні елементи навчального процесу: групи, найближчі завдання, події, розклад і оголошення. Такий екран виконує роль стартової панелі, з якої користувач може швидко перейти до потрібного модуля.

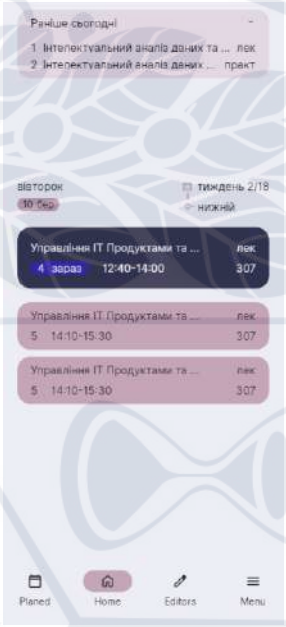

Рисунок 3.4 – Головна сторінка застосунку – DashboardPage (Mobile)

Сторінки навчальних модулів реалізовані за однаковим принципом. Наприклад, SchedulePage відображає заняття, SubjectsPage – список предметів, TasksPage – навчальні завдання, NotesPage – нотатки, EventsPage – події. Під час відкриття сторінки викликається команда завантаження, яка звертається до клієнтського сервісу, отримує дані з API та оновлює колекцію у ViewModel. Завдяки прив’язці даних інтерфейс автоматично відображає оновлений список.

Для локального збереження частини даних використовується LocalCacheService. Він створює SQLite-базу student-organizer-cache.db3 у директорії застосунку та зберігає дані у вигляді JSON. У кеш записуються дані користувача, групи, активний семестр, предмети, заняття розкладу, завдання та події. Це не є повною offline-first синхронізацією, але дозволяє швидше відображати останні отримані дані та зберігати базовий стан клієнта між

запусками.

Інтерфейс застосунку створено переважно програмно через сторінки ContentPage і допоміжний клас Ui. У ньому винесено повторювані елементи оформлення: сторінку, картку, заголовок секції, список, кнопки та поля введення. Для темізації використовується MauiTheme і палітра кольорів із ресурсів. Це дозволяє підтримувати єдиний стиль екранів і зменшує дублювання UI-коду між сторінками.

Проєкт містить платформні налаштування для Android і Windows. Для Android у AndroidManifest.xml передбачено потрібні параметри застосунку, а для Windows використовується відповідний Package.appxmanifest. Завдяки .NET MAUI основна частина коду інтерфейсу, ViewModels і сервісів є спільною для двох платформ, а відмінності обмежуються платформними файлами, налаштуваннями запуску та адресою API.

3.4. Реалізація вебверсії на Blazor Web App

Вебверсія системи реалізована в проєкті StudentOrganizer.Web з використанням Blazor Web App. Вона призначена для роботи із застосунком через браузер і надає доступ до основних модулів системи: авторизації, Dashboard, груп, предметів, розкладу, завдань, файлів, нотаток, подій, оголошень, Q&A та налаштувань. Веб-клієнт не працює напряму з базою даних, а отримує й оновлює інформацію через серверний API.

Проєкт StudentOrganizer.Web має посилання на StudentOrganizer.Shared, тому використовує спільні request- і response-моделі разом із серверною частиною. Це дозволяє не дублювати DTO-класи у веб-застосунку. Наприклад, під час входу використовується LoginRequest, під час реєстрації – RegisterRequest, для головної сторінки – DashboardResponse, для завдань – TaskResponse, а для сторінки предметів (рис. 3.5) – SubjectResponse.

У Program.cs налаштовано Razor Components з інтерактивним серверним режимом (лістинг 3.3). Також зареєстровано StudentOrganizerApiClient, який використовує HttpClient для звернення до API. Базова адреса API береться з

конфігурації Api:BaseUrl, а якщо вона не вказана, використовується http://localhost:5295. Окремо зареєстровано AuthTokenStore, який відповідає за збереження токенів і даних користувача в браузері.

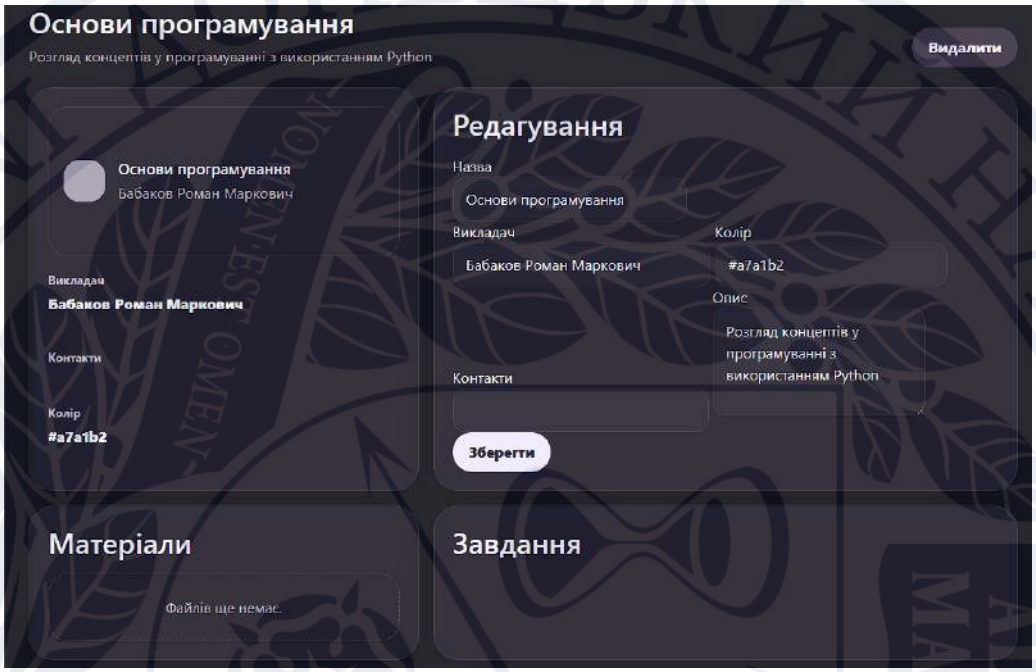

Рисунок 3.5 – Сторінка редагування предмету

Лістинг налаштування Razor компонентів має вигляд:

```
    builder.Services.AddRazorComponents()
        .AddInteractiveServerComponents();
    builder.Services.AddHttpClient<StudentOrganizerApiClient>(cl
ient => {
        var apiBaseUrl = builder.Configuration["Api:BaseUrl"] ??
"http://localhost:5295";
        client.BaseAddress = new Uri(apiBaseUrl);
    });
    builder.Services.AddScoped<AuthTokenStore>();
```

Загальна структура інтерфейсу побудована через компоненти App, Routes, MainLayout, Sidebar і TopBar. Компонент App формує HTML-основу

сторінки, підключає стилі, шрифти Material Symbols Rounded, ThemeStyles, HeadOutlet, маршрути та Blazor-скрипт. Компонент Routes відповідає за маршрутизацію між сторінками. За замовчуванням для сторінок використовується MainLayout.

MainLayout формує основний каркас веб-застосунку. Він містить бокову панель навігації Sidebar, верхню панель TopBar і область основного вмісту сторінки. Така структура дозволяє не дублювати меню та верхню панель на кожній сторінці. Сторінки авторизації використовують окремий AuthLayout, оскільки для входу та реєстрації не потрібна основна навігація застосунку.

Компонент Sidebar містить навігаційні посилання на основні розділи системи: головну сторінку, групи, предмети, завдання, нотатки, файли, оголошення, Q&A та налаштування. Для навігації використовується NavLink, який дозволяє автоматично визначати активний пункт меню. Також у Sidebar передбачено нижню навігацію для мобільного вигляду, що покращує зручність роботи з веб-версією на вузьких екранах.

Головна сторінка реалізована як Home.razor і доступна за маршрутами / та /dashboard. Вона отримує агреговані дані через GetDashboardAsync і відображає сьогоднішні заняття, найближчі дедлайни, майбутні події, останні оголошення та останні питання Q&A. Для відображення повторюваних елементів використовуються окремі карткові компоненти, наприклад TaskCard, EventCard, AnnouncementCard і QuestionCard.

Групова взаємодія реалізована через сторінки Announcements і Questions. Сторінка оголошень дозволяє переглядати публікації групи, створювати нові оголошення, редагувати їх, видаляти та працювати з коментарями. Сторінка Q&A дозволяє створювати питання, переглядати питання групи або предмету, додавати відповіді та реакції. Ці сторінки використовують ті самі DTO-моделі, що й API та MAUI-клієнт.

Основним класом для взаємодії веб-інтерфейсу з сервером є StudentOrganizerApiClient. Він містить методи для всіх основних модулів: авторизації, Dashboard, груп, навчальної структури, розкладу, завдань, подій,

нотаток, оголошень, питань, сповіщень і файлів. Усередині клієнта реалізовано універсальні методи для GET, POST, PUT, PATCH і DELETE запитів. Перед відправленням захищеного запиту клієнт отримує access token з AuthTokenStore і додає його в заголовок Authorization у форматі Bearer.

Вебінтерфейс не використовує локальні демонстраційні дані як основне джерело. Сторінки отримують інформацію через StudentOrganizerApiClient, а отже працюють із тими самими серверними даними, що й MAUI-клієнт. Це забезпечує узгодженість між веб-версією, Android/Windows-застосунком і серверною базою даних.

Адаптивність інтерфейсу реалізована через CSS-файли компонентів і загальний файл app.css. Для кожної великої сторінки та частини інтерфейсу створено окремий CSS-файл, наприклад для Dashboard, груп, предметів, завдань, файлів, нотаток, подій, оголошень і Q&A. Бокова навігація використовується на більших екранах (рис. 3.6), а нижня навігаційна панель – на мобільних. Це дозволяє зручно працювати з веб-застосунком як на комп’ютері, так і на пристроях із меншим екраном.

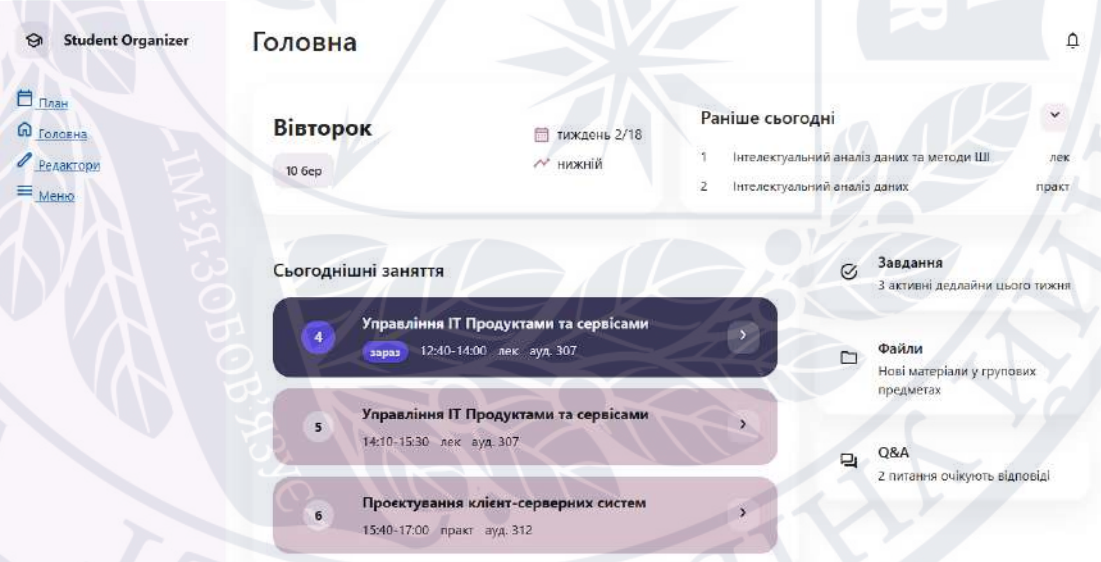

Рисунок 3.6 – Головна сторінка сайту – DashboardPage (Web)

Таким чином, вебверсія на Blazor Web App реалізує браузерний інтерфейс до основних функцій студентського органайзера. Вона використовує

Razor-компоненти, маршрутизацію, layout-и, форми з валідацією, окремий API-клієнт, збереження JWT у localStorage та адаптивний інтерфейс. Завдяки роботі через серверний API веб-клієнт використовує спільну бізнес-логіку й дані разом із MAUI-застосунком.

3.5. Тестування та демонстрація роботи системи

Проєкт StudentOrganizer.Tests призначений для перевірки основних модулів серверної частини застосунку. У ньому використовуються unit-тести, integration-тести та manual testing checklist. Такий підхід дозволяє перевірити як окремі сервіси прикладної логіки, так і повний сценарій роботи API через HTTP-запити[23]. Для автоматизованого тестування використано xUnit[41]. У тестовому проєкті також підключено Microsoft.AspNetCore.Mvc.Testing, Microsoft. EntityFrameworkCore.InMemory, Microsoft.NET.Test.Sdk і coverlet.collector. Це дозволяє запускати тести без реальної серверної бази даних, створювати тестовий екземпляр API та перевіряти HTTP-запити до контролерів[11].

Unit-тести зосереджені на перевірці окремих правил бізнес-логіки. Вони працюють без повного запуску Web API та звертаються безпосередньо до application-сервісів. Для цього використовується допоміжний клас TestDb, який створює AppDbContext на основі EF Core InMemory database. Під час створення контексту застосовуються початкові демонстраційні дані, що дає змогу перевіряти логіку на підготовленій структурі користувачів, груп, предметів, розкладу, завдань і файлів.

У unit-тестах перевіряється рольова модель груп, доступ до файлів, фільтрація завдань, фільтрація розкладу за типом тижня, створення сповіщень, робота сервісу сповіщень, публікація оголошень і видалення оголошень із пов’язаними коментарями та вкладеннями. Наприклад, окремий тест перевіряє, що студент не може змінити роль власника групи. Інший тест перевіряє, що особистий файл одного користувача не відображається іншому учаснику групи.

Integration-тести перевіряють роботу API як цілісної системи. Для цього використовується клас StudentOrganizerApiFactory, який наслідує WebApplicationFactory<Program>. Він запускає API в тестовому середовищі, підміняє реальну базу даних на InMemory database, задає тестові JWT-налаштування та створює тимчасову папку файлового сховища. Після завершення тестів тимчасова директорія видаляється, тому тестові файли не залишаються в основному сховищі застосунку.

Перший integration-тест (рис. 3.7) перевіряє захист endpoint-ів. Він надсилає запит до захищеного маршруту без токена й очікує відповідь Unauthorized. Це підтверджує, що базовий механізм [Authorize] працює коректно та API не дозволяє неавторизований доступ до приватних даних користувача.

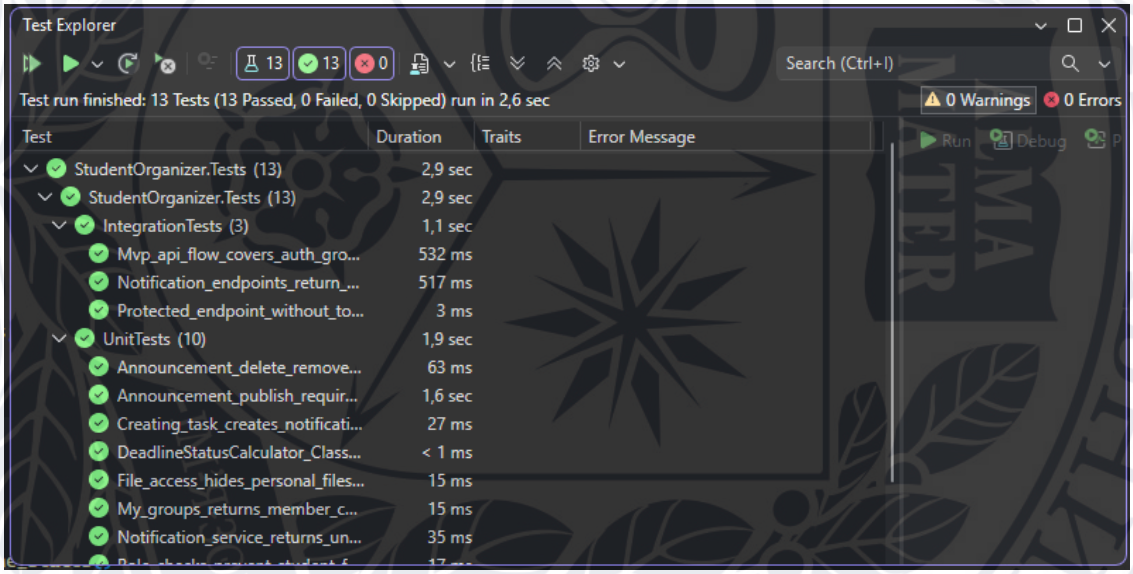

Рисунок 3.7 – Результати тестування у Visual Studio

Основний integration-тест перевіряє повний MVP-сценарій роботи системи. У межах цього сценарію виконується реєстрація користувача, вхід у систему, створення групи, створення оголошення, додавання коментаря, завантаження вкладення до оголошення, створення навчального року, семестру, предмету, завдання, розкладу, заняття, питання Q&A, відповіді на питання, реакції та файлу предмету. Наприкінці перевіряється завантаження

файлу з API. Такий тест підтверджує, що основні модулі не тільки працюють окремо, а й коректно взаємодіють між собою.

Отже, було здійснено практичну реалізацію кросплатформового застосунку для організації навчального процесу студентів. Розглянуто структуру програмного рішення, основні проєкти системи та їх призначення. Окрему увагу приділено реалізації доменної моделі, бази даних, автентифікації, авторизації, ролей користувачів, серверного API та бізнес-логіки основних модулів. У розділі було розглянуто реалізацію файлового сховища, MAUI-клієнта для Android і Windows, веб-версії на Blazor Web App та спільних DTO, які забезпечують узгоджену взаємодію між клієнтами й сервером. Завершальним етапом стало тестування системи та демонстрація роботи основних функцій. Отримані результати підтверджують, що розроблена система реалізує ключові можливості студентського органайзера та може використовуватися як основа для подальшого розширення функціональності.

ВИСНОВКИ

У ході виконання бакалаврської роботи було досліджено, спроєктовано та реалізовано кросплатформний застосунок для організації навчальної діяльності. Актуальність теми зумовлена тим, що навчальна інформація часто розміщується у різних незалежних сервісах: месенджерах, хмарних сховищах, календарях, електронних таблицях, нотатниках і системах дистанційного навчання. Це ускладнює пошук матеріалів, контроль дедлайнів, ведення розкладу, збереження файлів і координацію роботи академічної групи. Розроблена система спрямована на об'єднання основних навчальних інструментів в одному програмному середовищі. За результатами роботи можна зробити наступні висновки:

1. Дослідження теоретичних основи створення сучасних клієнт-серверних застосунків показало, що використовуються підходи до кросплатформної розробки з використанням платформи .NET, мови C#, ASP.NET Core Web API, Entity Framework Core, Blazor Web App та .NET MAUI.

2. На основі аналізу було визначено загальну концепцію системи, її цільову аудиторію, функціональні та нефункціональні вимоги, а також основні сценарії використання. Визначено, що базовими функціональними вимогами є реєстрація та авторизація користувачів. Система повинна підтримувати роботу з академічними групами, навчальними завданнями, подіями. Нефункціональні вимоги забезпечують зручність, стабільність, безпечність, продуктивність та подальший розвиток. Основними такими вимогами є: централізоване зберігання даних, адаптивність інтерфейсу, захист серверного API, контроль доступу до файлів, підтримка асинхронного виконання операцій.

3. Систему було спроєктовано як комплекс, що поєднує серверне API, базу даних, файлове сховище, Android/Windows-клієнт і вебверсію.

4. Під час реалізації було створено програмне рішення з модульною структурою, у якому серверна частина відповідає за обробку запитів, бізнес-логіку, роботу з базою даних і файловим сховищем, а клієнтські застосунки

забезпечують доступ користувача до основних функцій системи. Було реалізовано роботу з навчальними даними, групами, розкладом, завданнями, файлами, нотатками, подіями та груповою взаємодією. Також було передбачено механізми автентифікації, авторизації та розмежування прав доступу, що дозволяє захистити дані користувачів і керувати доступом до окремих дій відповідно до ролі учасника.

5. Тестування розробленої системи, перевірити коректність роботи основних функцій, взаємодію клієнтів із сервером. Для перевірки основних модулів серверної частини застосунку використано unit-тести, integration-тести та manual testing checklist. Використання застосунку надає змогу зменшити розрізненості навчальної інформації, спрощститиі доступ до матеріалів, покращити контроль навчальних завдань і створити єдиний простір для навчальної взаємодії. Подальший розвиток системи може передбачати розширення синхронізації, впровадження push-сповіщень, інтеграцію з календарями, імпорт розкладу з файлів, пошук по вмісту документів і додаткові інструменти для автоматизації роботи з навчальними матеріалами.

Отже, у результаті виконання бакалаврської роботи було створено систему, яка об’єднує основні інструменти для організації навчального процесу студентів і може використовуватися як для особистої роботи, так і для координації діяльності академічної групи.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Баран С. В. Основи web-програмування : навчальний посібник. Кривий Ріг : Державний університет економіки і технологій, 2023. 316 с. URL: https://dspace.duet.edu.ua/handle/123456789/832 (дата звернення: 01.03.2026).

2. Бородкіна І. Л., Бородкін Г. О. Інженерія програмного забезпечення : посібник для студентів вищих навчальних закладів. Київ, 2018. URL: https://www.knuba.edu.ua/wp-content/uploads/2022/10/Інженерія-програмного-забезпечення.pdf (дата звернення: 05.04.2026).

3. Заволодько Г. Є., Касілов О. В., Бронін С. В. Основи веброзробки : навчально-методичний посібник. Харків : НТУ ХПІ, 2025. 322 с. URL: https://repository.kpi.kharkov.ua/handle/KhPI-Press/93821 (дата звернення: 01.03.2026).

4. Коноваленко І. В., Марущак П. О. Платформа .NET та мова програмування C# 8.0 : навчальний посібник. Тернопіль : ФОП Паляниця В. А., 2020. 320 с. URL: https://elartu.tntu.edu.ua/bitstream/lib/32825/1/Konovalenko%20I.%20.NET-C%23.pdf (дата звернення: 16.05.2026).

5. Корнієнко Б. Я. Компоненти програмної інженерії. Архітектура програмного забезпечення. Лабораторний практикум : навчальний посібник. Київ : КПІ ім. Ігоря Сікорського, 2023. 88 с. URL: https://ela.kpi.ua/handle/123456789/55781 (дата звернення: 01.06.2026).

6. Лавріщева К. М. Програмна інженерія : підручник. Київ, 2008. 319 с. URL: https://csc.knu.ua/library/books/lavrishcheva-6.pdf (дата звернення: 26.05.2026).

7. Левченко Л. О., Кучук Н. Г., Тарнавський Ю. А., Колумбет В. П. Архітектура системного програмного забезпечення : підручник. КПІ ім. Ігоря Сікорського, 2022. 497 с. URL: https://ela.kpi.ua/bitstream/123456789/49759/1/Ar_system_prohram_2022.pdf (дата звернення: 26.05.2026).

8. Молодченко Д. В., Потапова Н. А. Кросплатформовий застосунок для організації навчального процесу. Прикладні інформаційні технології: Збірник

тез доповідей, м. Вінниця, 15 травня 2026 р. Вінниця: ДонНУ імені Василя Стуса, 2026.

9. Павловський В. І. Бази даних та засоби управління : підручник / В. І. Павловський, А. В. Петрашенко. Київ : КПІ ім. Ігоря Сікорського, 2024. 326 с. URL: https://ela.kpi.ua/handle/123456789/70913 (дата звернення: 02.04.2026).

10. Проєктування інформаційних систем : навчальний посібник. Київ : КПІ ім. Ігоря Сікорського, 2022. URL: https://ela.kpi.ua/server/api/core/bitstreams/6c50f6dd-87f2-48f0-ba13-7d0332e1673e/content (дата звернення: 07.04.2026).

11. Трофименко О. Г. Тестування та забезпечення якості програмних систем : навчально-методичний посібник / О. Г. Трофименко, А. І. Дика. Одеса : Фенікс, 2024. 140 с. URL: https://hdl.handle.net/11300/27246 (дата звернення: 21.05.2026).

12. Хіцко Я. В. Технологія проектування програмних систем : лабораторний практикум / Я. В. Хіцко, Л. А. Люшенко, Ю. В. Бухтіяров. Київ : КПІ ім. Ігоря Сікорського, 2019. 44 с. URL: https://ela.kpi.ua/bitstream/123456789/38077/1/2020_TPS_praktykum.pdf (дата звернення: 01.06.2026).

13. Цифрова трансформація освіти і науки. Міністерство освіти і науки України. URL: https://mon.gov.ua/tag/tsifrova-transformatsiya-osviti-i-nauki (дата звернення: 26.05.2026).

14. ASP.NET Core documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/core/ (дата звернення: 01.03.2026).

15. ASP.NET Core Web API documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/core/web-api/ (дата звернення: 01.03.2026).

16. Bass L. Software Architecture in Practice / L. Bass, P. Clements, R. Kazman. 4th ed. Boston : Addison-Wesley Professional, 2021. 464 p.

17. Blazor documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/core/blazor/ (дата звернення:

24.05.2026).

18. CommunityToolkit.Mvvm documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/communitytoolkit/mvvm/ (дата звернення: 26.05.2026).

19. ECMA-334. C# Language Specification. ECMA International. URL: https://ecma-international.org/publications-and-standards/standards/ecma-334/ (дата звернення: 16.05.2026).

20. Entity Framework Core documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 02.04.2026).

21. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley Professional, 2003. 560 p.

22. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley Professional, 2002. 560 p.

23. Fowler M. Test Pyramid. URL: https://martinfowler.com/bliki/TestPyramid.html (дата звернення: 26.05.2026).

24. Gamma E. Design Patterns: Elements of Reusable Object-Oriented Software / E. Gamma, R. Helm, R. Johnson, J. Vlissides. Boston : Addison-Wesley Professional, 1994. 395 p.

25. HTTP Semantics : RFC 9110 / R. Fielding, M. Nottingham, J. Reschke. Internet Engineering Task Force, 2022. URL: https://datatracker.ietf.org/doc/html/rfc9110 (дата звернення: 01.03.2026).

26. JSON Web Token (JWT) : RFC 7519. M. Jones, J. Bradley, N. Sakimura. Internet Engineering Task Force, 2017. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 02.03.2026).

27. Kleppmann M. Designing Data-Intensive Applications. Sebastopol : O’Reilly Media, 2017. 616 p.

28. Martin R. C. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.

29. Material Design 3. Google. URL: https://m3.material.io/ (дата звернення: 23.04.2026).

30. .NET MAUI documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/maui/ (дата звернення: 09.03.2026).

31. Nielsen J. 10 Usability Heuristics for User Interface Design. Nielsen Norman Group. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення: 20.04.2026).

32. OECD Digital Education Outlook 2023: Towards an Effective Digital Education Ecosystem. OECD Publishing, 2023. URL: https://www.oecd.org/en/publications/oecd-digital-education-outlook-2023_c74f03de-en.html (дата звернення: 26.05.2026).

33. OpenAPI Specification. OpenAPI Initiative. URL: https://spec.openapis.org/oas/latest.html (дата звернення: 02.03.2026).

34. OWASP API Security Top 10 2023. OWASP Foundation. URL: https://owasp.org/API-Security/editions/2023/en/0x11-t10/ (дата звернення: 02.03.2026).

35. OWASP Application Security Verification Standard. OWASP Foundation. URL: https://owasp.org/www-project-application-security-verification-standard/ (дата звернення: 02.03.2026).

36. OWASP REST Security Cheat Sheet. OWASP Cheat Sheet Series. URL: https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html (дата звернення: 02.03.2026).

37. PostgreSQL Documentation. PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs/ (дата звернення: 02.04.2026).

38. Richardson L. RESTful Web APIs: Services for a Changing World / L. Richardson, M. Amundsen, S. Ruby. Sebastopol : O'Reilly Media, 2013. 406 p.

39. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2015. 816 p.

40. SQLite Documentation. SQLite Consortium. URL: https://sqlite.org/docs.html (дата звернення: 02.04.2026).

41. xUnit.net documentation. URL: https://xunit.net/ (дата звернення: 25.05.2026).

Додаток А

Клас AppDbContext

```
     public sealed class
AppDbContext(DbContextOptions<AppDbContext> options)
         : DbContext(options), IApplicationDbContext {
         public DbSet<User> Users => Set<User>();
         public DbSet<Group> Groups => Set<Group>();
         public DbSet<GroupMember> GroupMembers =>
Set<GroupMember>();
         public DbSet<AcademicYear> AcademicYears =>
Set<AcademicYear>();
         public DbSet<Semester> Semesters => Set<Semester>();
         public DbSet<Subject> Subjects => Set<Subject>();
         public DbSet<Schedule> Schedules => Set<Schedule>();
         public DbSet<ScheduleItem> ScheduleItems =>
Set<ScheduleItem>();
         public DbSet<StudentTask> Tasks => Set<StudentTask>();
         public DbSet<FileItem> Files => Set<FileItem>();
         public DbSet<FileShare> FileShares => Set<FileShare>();
         public DbSet<Note> Notes => Set<Note>();
         public DbSet<NoteTag> NoteTags => Set<NoteTag>();
         public DbSet<CalendarEvent> Events =>
Set<CalendarEvent>();
         public DbSet<AnnouncementPost> AnnouncementPosts =>
Set<AnnouncementPost>();
         public DbSet<AnnouncementComment> AnnouncementComments
=> Set<AnnouncementComment>();
         public DbSet<Question> Questions => Set<Question>();
         public DbSet<QuestionAnswer> QuestionAnswers =>
Set<QuestionAnswer>();
         public DbSet<QuestionReaction> QuestionReactions =>
Set<QuestionReaction>();
         public DbSet<Notification> Notifications =>
Set<Notification>();
         public DbSet<RefreshToken> RefreshTokens =>
Set<RefreshToken>();
     }
```