Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: *Інтелектуальна система управління розкладом користувача на основі ONION-архітектури*

Виконав: студент 6 курсу групи КН-63м
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Доманський В.М.
(прізвище та ініціали)

Керівник Яцишин С.І.
(прізвище та ініціали)

Рецензент Сторожук О.М.
(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
_____ Борецька І.Б.
"10" грудня 2025 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Доманському Вадиму Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Інтелектуальна система управління розкладом користувача на основі ONION-архітектури*
керівник роботи *Яцишин Світлана Іванівна, к.т.н., доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "29" *квітня 2025року*
№ С-288
2. Термін подання студентом роботи 10 грудня 2025 р.
3. Вихідні дані до роботи Розробити систему керування та оптимізації розкладу для закладів освіти, система має автоматично повідомляти викладачів та студентів про щоденний розклад та його зміни в реальному часі та надати зрозумілий інтерфейс для керування
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Стан проблемної області
Інформаційне забезпечення
Математичне забезпечення
Програмне забезпечення
Розроблення стартап-проєкту
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка матеріалів до доповіді

6. Дата видачі завдання 1 травня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області та існуючих рішень для автоматизації навчальних розкладів	01.05.25 - 05.05.25	Виконано
2	Формування вимог до системи, постановка задачі та визначення Onion-архітектури	06.05.25 - 20.05.25	Виконано
3	Вибір технологій і інструментів (.NET 9, EF Core, React, Telegram Bot API, Hangfire, MSSQL)	21.05.25 - 27.05.25	Виконано
4	Розроблення інформаційного забезпечення: моделі даних, ER-діаграми, структура БД	28.05.25 - 10.06.25	Виконано
5	Розроблення алгоритмічного забезпечення: логіка генерації розкладу, обробка шаблонів, валідація подій	11.06.25 - 20.06.25	Виконано
6	Реалізація серверної частини системи на .NET 9 з підтримкою EF Core, Hangfire та Telegram Bot API	21.06.25 - 20.07.25	Виконано
7	Реалізація фронтенд-частини (React + Material UI): інтерфейс адміністратора та викладача	21.07.25 - 20.08.25	Виконано
8	Інтеграція системи, тестування API, тестування Telegram-бота та фонових задач	21.08.25 10.09.25	Виконано
9	Оформлення пояснювальної записки, підготовка додатків та презентації	11.09.25 - 20.11.25	Виконано

Студент _____ Доманський В.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Яцишин С.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська робота присвячена розробці інтелектуальної системи керування навчальним розкладом, що ґрунтується на принципах Onion-архітектури. У межах дослідження створено програмний комплекс, який об’єднує вебінтерфейс для викладачів і адміністраторів, серверну частину та Telegram-бота для студентів. Вебзастосунок реалізовано з використанням React, що забезпечує зручність роботи та інтуїтивність взаємодії користувача з системою. Серверна частина розроблена на платформі .NET 9 і бази даних MSSQL, що дозволило забезпечити надійність, структурованість та масштабованість рішення.

Для студентів реалізовано Telegram-бот, який надає можливість оперативно переглядати розклад та отримувати сповіщення про зміни. У роботі детально описано механізм взаємодії складових системи, принципи формування тижневих шаблонів, особливості автоматизації розкладу та інтеграцію механізмів фонового опрацювання задач.

ABSTRACT

The master's thesis is devoted to the development of an intelligent school schedule management system based on the principles of Onion architecture. As part of the research, a software complex was created that combines a web interface for teachers and administrators, a server part, and a Telegram bot for students. The web application was implemented using React, which ensures ease of use and intuitive user interaction with the system. The server part was developed on the .NET 9 platform and MSSQL database, which ensured the reliability, structure, and scalability of the solution.

A Telegram bot has been implemented for students, which allows them to quickly view the schedule and receive notifications about changes. The work describes in detail the mechanism of interaction between the components of the system, the principles of forming weekly templates, the features of schedule automation, and the integration of background task processing mechanisms.

ТЕХНІЧНЕ ЗАВДАННЯ

Необхідно розробити інтелектуальну систему управління навчальним розкладом з використанням Onion-архітектури, яка включає:

1. Реалізацію вебінтерфейсу для адміністраторів та викладачів з використанням React і Material UI;

2. Розробку серверної частини на .NET 9 із застосуванням Onion-архітектури та EF Core;

3. Створення Telegram-бота для студентів, який дозволяє переглядати розклад на день або тиждень та отримувати сповіщення;

4. Здійснення авторизації користувачів з розмежуванням прав доступу (адмін, викладач, студент);

5. Розробку функціоналу керування університетами, викладачами, групами, предметами та розкладом;

6. Зберігання інформації в базі даних MSSQL;

7. Тестування працездатності основних сценаріїв та перевірка коректності Telegram-сповіщень.

ЗМІСТ

ВСТУП 7
РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ 9
1.1 Актуальність автоматизації розкладу в освітніх закладах 9
1.1.1 Проблеми традиційного підходу до створення розкладу 9
1.1.2 Необхідність інтелектуальних систем у сфері освіти 9
1.1.3 Переваги впровадження цифрових рішень 10
1.2 Аналіз сучасних програмних рішень для управління розкладом 10
1.3 Вибір архітектурного підходу: обґрунтування Onion-архітектури 11
Висновки до розділу 11
РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ 12
2.1 Visual Studio 2022 12
2.2 Docker 15
2.3 DBeaver 18
2.4 .NET 9 SDK 20
2.5 Microsoft SQL Server 21
2.6 Node.js 22
2.7 React 23
РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ 24
3.1 Формалізація предметної області 24
3.2 Базові множини 24
3.3 Алгоритмічне забезпечення розк 26
3.4 Система автоматичного сповіщення 28
3.5 Інтеграція математичних моделей у Onion-архітектуру 29
Висновки до розділу 32
РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ 33
4.1 Основні функціональні вимоги до системи 33
4.2 Модель користувачів та авторизація 35

4.3 Інформаційні сутності та їх взаємозв’язки 38
4.4 Рівні архітектурної підтримки даних 41
4.5 Інтеграція Telegram-бота як джерела/отримувача даних 46
4.6 UI: веб інтерфейс викладача 49
Висновки до розділу 51
Розділ 5. Розроблення стартап-проєкту 52
5.1 Опис ідеї проєкту 52
5.2. Аналіз технологічних можливостей реалізації проєкту 57
5.3 Аналіз ринкових можливостей 60
5.4. Розроблення ринкової стратегії проєкту 64
5.4.1. Стратегія охоплення ринку 65
5.4.2. Позиціонування продукту 66
5.4.3. Шляхи виходу на ринок 66
5.5. Маркетингова програма стартап-проєкту 67
5.5.1. Концепція продукту 67
5.5.2. Цінова політика 68
5.5.3. Стратегія збуту 68
5.5.4. Стратегія просування 69
ВИСНОВКИ 70
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 72
ДОДАТОК А Структура бази даних системи 74
ДОДАТОК В Архітектура програмної системи 75
ДОДАТОК С 77

ВСТУП

У сучасних умовах цифрової трансформації освіти дедалі більшої актуальності набуває впровадження інтелектуальних інформаційних систем, які спрощують адміністративні та навчальні процеси в закладах вищої освіти. Однією з ключових складових організації навчального процесу є побудова розкладу занять. Традиційне ручне або частково автоматизоване планування розкладу не відповідає вимогам гнучкості, масштабованості та інтегрованості в сучасні цифрові середовища. Особливо це стосується розподілених навчальних структур, великої кількості груп і частих змін у розкладі впродовж семестру.

У зв’язку з цим виникає потреба у створенні інтелектуальної системи управління розкладом, яка б дозволяла централізовано створювати, редагувати, контролювати та оперативно поширювати розклад серед учасників освітнього процесу. Ефективне вирішення цієї задачі можливе за рахунок поєднання сучасних архітектурних рішень, зокрема Onion-архітектури, алгоритмічного апарату планування та технологій інтерактивної взаємодії з користувачами, таких як Telegram-боти.

Об’єкт дослідження — процеси автоматизованого управління навчальним розкладом у закладах вищої освіти з використанням сучасних IT-рішень.

Предмет дослідження — методи побудови, зберігання, валідації та динамічного оновлення навчального розкладу із застосуванням сучасних технологій та архітектурних підходів.

Мета роботи — розробити інтелектуальну систему управління розкладом занять, яка базується на Onion-архітектурі та забезпечує зручне створення, оновлення і поширення розкладу з урахуванням гнучкості та масштабованості.

Завдання дослідження:

1. Проаналізувати існуючі підходи до автоматизації управління розкладом.
2. Обґрунтувати вибір Onion-архітектури як основи для побудови системи.

3. Реалізувати серверну частину на базі .NET 9 із використанням Entity Framework Core та Hangfire.

4. Розробити фронтенд-інтерфейс для адміністраторів і викладачів за допомогою React [5] та Material UI [6].

5. Створити Telegram-бота для сповіщення студентів про актуальний розклад.

6. Провести тестування системи та оцінити її ефективність у реальних умовах.

Наукова новизна роботи полягає в поєднанні Onion-архітектури, автоматизованих задач планування та інструментів інтерактивної комунікації (Telegram Bot API) для створення адаптивної, масштабованої та зручної у використанні системи управління розкладом. Запропонований підхід дозволяє забезпечити цілісність даних, високу продуктивність системи та зручність розширення функціоналу.

У роботі використано такі технології: .NET 9, Entity Framework Core, Onion Architecture, React + Material UI, Telegram Bot API, Hangfire, MSSQL. Запропонована система дозволяє створювати розклад на основі тижневих шаблонів, призначати періоди повторення, видаляти або коригувати окремі заняття, а також здійснювати щоденні сповіщення студентів.

Практична цінність роботи полягає в можливості реального впровадження системи в освітній установі з подальшою адаптацією до специфіки навчального процесу. Результати дослідження можуть бути основою для подальшої розробки універсальних платформ керування розкладом у закладах освіти.

РОЗДІЛ 1
СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Актуальність автоматизації розкладу в освітніх закладах

Планування навчального розкладу є однією з ключових адміністративних задач у закладах вищої освіти. Традиційне ручне формування розкладу є трудомістким, неефективним і часто призводить до конфліктів у навантаженні викладачів, дублювання пар або неузгодженостей із наявністю аудиторій. За умов збільшення кількості груп, викладачів та спеціальностей ручне планування перестає бути практичним.

Автоматизація цього процесу дозволяє не лише зменшити навантаження на працівників, але й забезпечити узгодженість, прозорість та швидке внесення змін у розклад. Застосування інтелектуальних систем дозволяє підвищити якість управління освітнім процесом.

1.1.1 Проблеми традиційного підходу до створення розкладу

- Основні проблеми ручного підходу включають:
- людський фактор і помилки при плануванні;
- складність коригування розкладу при змінах;
- відсутність централізованого доступу для учасників освітнього процесу;
- недостатнє інформування студентів про зміни.

1.1.2 Необхідність інтелектуальних систем у сфері освіти

Інтелектуальні програмні рішення допомагають оптимізувати та автоматизувати повторювані процеси. Вони дозволяють враховувати обмеження (наприклад, навантаження викладача, наявність аудиторій, перетини в розкладі), генерувати зручний для всіх користувачів розклад і оперативно інформувати студентів про зміни.

Особливої актуальності набуває використання месенджерів, таких як Telegram, як каналу сповіщення. Молодь активно користується цими сервісами, що дозволяє забезпечити миттєву комунікацію з користувачем.

1.1.3 Переваги впровадження цифрових рішень

- Переваги інтелектуальної системи управління розкладом:
- централізоване зберігання та доступ до даних;
- можливість роботи з будь-якого пристрою;
- розмежування ролей (адміністратор, викладач, студент);
- оперативне внесення змін;
- інтеграція з Telegram для щоденних нагадувань і повідомлень про скасування пар.

1.2 Аналіз сучасних програмних рішень для управління розкладом

На сьогодні існує декілька програмних рішень, що пропонують управління розкладом — як для шкіл, так і для університетів. Серед них можна відзначити Google Calendar, АСУ ВНЗ, розширення Moodle, а також спеціалізовані сервіси типу "Розклад ПНУ".

Більшість із них:

- не мають гнучкої системи ролей користувачів;
- не дозволяють інтеграцію з Telegram;
- мають складний або застарілий інтерфейс;
- обмежені в можливостях розширення.

Таким чином, є потреба у створенні сучасної адаптивної системи з чіткою архітектурою та підтримкою широкого спектру функцій.

1.3 Вибір архітектурного підходу: обґрунтування Onion-архітектури

Onion-архітектура — це сучасний архітектурний підхід до побудови програмного забезпечення, що дозволяє забезпечити:

- розділення логіки на шари (доменна, прикладна, інфраструктурна);
- інверсію залежностей (внутрішні шари не знають про зовнішні);
- високу модульність та тестованість.

Використання Onion-архітектури дозволяє створити стабільну основу для системи, яку в майбутньому можна масштабувати та підтримувати. У поєднанні з сучасними технологіями (React, .NET, EF Core, Telegram Bot API, Hangfire) вона дозволяє реалізувати ефективне та зручне для користувача рішення.

Висновки до розділу

В результаті аналізу проблемної області було виявлено низку недоліків у традиційному підході до управління розкладом та обґрунтовано необхідність створення інтелектуальної системи. Для вирішення поставлених задач доцільно використовувати Onion-архітектуру, що забезпечує надійну, модульну та розширювану основу для побудови сучасного веб-застосунку з інтеграцією месенджерів.

РОЗДІЛ 2
ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Visual Studio 2022

Visual Studio [15] — це інтегроване середовище розробки (IDE), розроблене компанією Microsoft. Воно використовується для розробки комп'ютерних програм, включаючи веб-сайти, веб-додатки, веб-сервіси та мобільні додатки. Visual Studio використовує платформи розробки програмного забезпечення Microsoft, включаючи Windows API, Windows Forms, Windows Presentation Foundation (WPF), Microsoft Store та Microsoft Silverlight. Воно може створювати як нативний код, так і керований код.

Visual Studio включає редактор коду, що підтримує IntelliSense (компонент автодоповнення коду), а також рефакторинг коду. Інтегрований відладчик працює як відладчик на рівні вихідного коду, так і відладчик на рівні машини. Інші вбудовані інструменти включають профілювальник коду, конструктор для створення графічних інтерфейсів користувача, веб-дизайнер, конструктор класів та конструктор схем баз даних. Він підтримує плагіни, що розширюють функціональність майже на всіх рівнях, включаючи додавання підтримки систем контролю версій (таких як Subversion та Git) та додавання нових наборів інструментів, таких як редактори та візуальні конструктори для мов, специфічних для певної області, або наборів інструментів для інших аспектів життєвого циклу розробки програмного забезпечення (таких як клієнт Azure DevOps: Team Explorer).

Visual Studio підтримує 36 різних мов програмування[потрібне джерело] і дозволяє редактору коду та відладчику підтримувати (в різній мірі) майже будь-яку мову програмування, за умови існування специфічної для мови служби. Вбудовані мови включають C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML та CSS. Підтримка інших мов, таких як Python, Ruby, Node.js та M, серед інших, доступна через плагіни. Java (та J#) підтримувалися в минулому.

Visual Studio пропонується в декількох редакціях, причому редакція Community доступна безкоштовно для студентів, учасників відкритих проектів та індивідуальних розробників. Microsoft зазвичай випускає нові основні версії кожні кілька років. Visual Studio 2026 — це остання стабільна версія, готова до використання. Старіші версії, включаючи Visual Studio 2017, 2019 та 2022, залишаються під розширеною підтримкою.

Visual Studio не підтримує жодної мови програмування, рішення або інструменту; натомість, вона дозволяє підключати функціональність, закодовану як VSPackage. Після встановлення функціональність стає доступною як послуга. IDE надає три послуги: SVsSolution, яка надає можливість перелічувати проекти та рішення; SVsUIShell, що забезпечує функціональність вікон та інтерфейсу користувача (включаючи вкладки, панелі інструментів та вікна інструментів); та SVsShell, що займається реєстрацією VSPackages. Крім того, IDE також відповідає за координацію та забезпечення комунікації між службами. Усі редактори, дизайнери, типи проектів та інші інструменти реалізовані як VSPackages. Visual Studio використовує COM для доступу до VSPackages. Visual Studio SDK також включає Managed Package Framework (MPF), який є набором керованих обгорток навколо інтерфейсів COM, що дозволяють писати пакети будь-якою мовою, сумісною з CLI. Однак MPF не надає всієї функціональності, що надається інтерфейсами COM Visual Studio. Потім служби можуть бути використані для створення інших пакетів, які додають функціональність до IDE Visual Studio.

Підтримка мов програмування додається за допомогою спеціального VSPackage, який називається Language Service. Мовна служба визначає різні інтерфейси, які реалізація VSPackage може впровадити для додавання підтримки різних функціональних можливостей. Функціональні можливості, які можна додати таким чином, включають підсвічування синтаксису, автозавершення операторів, підбір дужок, підказки з інформацією про параметри, списки членів та маркери помилок для фонової компіляції. Якщо інтерфейс реалізовано, функціональність

буде доступна для мови. Мовні служби реалізуються для кожної мови окремо. Реалізації можуть повторно використовувати код з парсера або компілятора для мови. Мовні служби можуть бути реалізовані як у нативному коді, так і в керованому коді. Для нативного коду можна використовувати нативні інтерфейси COM або Babel Framework (частина Visual Studio SDK). Для керованого коду MPF включає обгортки для написання керованих мовних служб.

Visual Studio включає редактор коду, який підтримує підсвічування синтаксису та автозавершення коду за допомогою IntelliSense для змінних, функцій, методів, циклів та запитів LINQ. IntelliSense підтримується для включених мов, а також для XML, каскадних таблиць стилів та JavaScript під час розробки веб-сайтів та веб-додатків. Підказки автозавершення з'являються в безмодальному списку над вікном редактора коду, поблизу курсору редагування. У Visual Studio 2008 і пізніших версіях його можна тимчасово зробити напівпрозорим, щоб побачити код, який він закриває. Редактор коду використовується для всіх підтримуваних мов.

Редактор коду в Visual Studio також підтримує встановлення закладок у коді для швидкої навігації. Інші засоби навігації включають згортання блоків коду та інкрементний пошук, крім звичайного пошуку тексту та пошуку за регулярними виразами. Редактор коду також включає багатоелементний буфер обміну та список завдань. Редактор коду підтримує фрагменти коду, які є збереженими шаблонами для повторюваного коду і можуть бути вставлені в код та налаштовані для проекту, над яким працюють. Також вбудовано інструмент управління фрагментами коду. Ці інструменти відображаються у вигляді плаваючих вікон, які можна налаштувати так, щоб вони автоматично ховалися, коли не використовуються, або прикріплювалися до бокової частини екрана. Редактор коду в Visual Studio також підтримує рефакторинг коду, включаючи переупорядкування параметрів, перейменування змінних і методів, вилучення інтерфейсів та інкапсуляцію членів класу всередині властивостей, серед іншого.

2.2 Docker

Docker — це набір продуктів «платформа як послуга» (PaaS), що використовує віртуалізацію на рівні операційної системи для постачання програмного забезпечення у вигляді пакетів, які називаються контейнерами. Docker автоматизує розгортання додатків у легких контейнерах, що дозволяє їм стабільно працювати в різних обчислювальних середовищах [14].

Контейнери ізольовані один від одного і містять власне програмне забезпечення, бібліотеки та файли конфігурації; вони можуть спілкуватися між собою через чітко визначені канали. Оскільки всі контейнери використовують послуги одного ядра операційної системи, вони споживають менше ресурсів, ніж віртуальні машини.

Docker може упакувати додаток та його залежності у віртуальний контейнер, який може працювати на будь-якому комп'ютері з ОС Linux, Windows або macOS. Це дозволяє додатку працювати в різних місцях, таких як локальна мережа, публічна (див. децентралізовані обчислення, розподілені обчислення та хмарні обчислення) або приватна хмара. Під час роботи в Linux Docker використовує функції ізоляції ресурсів ядра Linux (такі як cgroups і простори імен ядра) та файлову систему з можливістю об'єднання (таку як OverlayFS), щоб контейнери могли працювати в одному екземплярі Linux, уникаючи накладних витрат на запуск і обслуговування віртуальних машин. Docker на macOS використовує віртуальну машину Linux для запуску контейнерів.

Оскільки контейнери Docker є легкими, один сервер або віртуальна машина можуть одночасно запускати кілька контейнерів. Аналіз 2018 року показав, що типовий випадок використання Docker передбачає запуск восьми контейнерів на один хост, а чверть проаналізованих організацій запускають 18 або більше контейнерів на один хост. Його також можна встановити на одноплатний комп'ютер, такий як Raspberry Pi.

Підтримка ядра Linux для просторів імен в основному ізолює вигляд операційного середовища додатка, включаючи дерева процесів, мережу, ідентифікатори користувачів і змонтовані файлові системи, тоді як cgroups ядра забезпечують обмеження ресурсів для пам'яті та процесора. Починаючи з версії 0.9, Docker включає власний компонент (під назвою libcontainer) для використання засобів віртуалізації, що надаються безпосередньо ядром Linux, на додаток до використання абстрактних інтерфейсів віртуалізації через libvirt, LXC та systemd-nspawn.

Docker реалізує API високого рівня для надання легких контейнерів, які запускають процеси в ізоляції.

Програмне забезпечення Docker як послуга складається з трьох компонентів:

- Програмне забезпечення: Демон Docker, який називається dockerd, є постійним процесом, який управляє контейнерами Docker і обробляє об'єкти контейнерів. Демон очікує на запити, які надсилаються через API Docker Engine. Клієнтська програма Docker, яка називається docker, надає інтерфейс командного рядка (CLI), який дозволяє користувачам взаємодіяти з демонами Docker.
- Об'єкти: Об'єкти Docker — це різні сутності, що використовуються для складання додатка в Docker. Основними класами об'єктів Docker є образи, контейнери та служби.
    - Контейнер Docker — це стандартизоване, інкапсульоване середовище, в якому виконуються додатки. Контейнер управляється за допомогою API або CLI Docker.
    - Образ Docker — це шаблон тільки для читання, який використовується для створення контейнерів. Образи використовуються для зберігання та доставки додатків.
    - Служба Docker дозволяє масштабувати контейнери на декількох демонах Docker. Результат називається роєм — набором взаємодіючих демонів, які спілкуються через API Docker.

- Реєстри: Реєстр Docker — це сховище образів Docker. Клієнти Docker підключаються до реєстрів, щоб завантажувати («pull») образи для використання або завантажувати («push») образи, які вони створили. Реєстри можуть бути публічними або приватними. Основним публічним реєстром є Docker Hub. Docker Hub — це реєстр за замовчуванням, де Docker шукає образи. Реєстри Docker також дозволяють створювати сповіщення на основі подій.

Dockerfile — це текстовий файл, який зазвичай визначає кілька аспектів контейнера Docker: дистрибутив Linux, команди інсталяції для середовища виконання мови програмування та вихідний код програми.

Інструменти

- Docker Compose — це інструмент для визначення та запуску багатоконтейнерних додатків Docker. Він використовує файли YAML для налаштування служб додатка та виконує процес створення та запуску всіх контейнерів за допомогою однієї команди. Утиліта CLI docker compose дозволяє користувачам виконувати команди на декількох контейнерах одночасно, наприклад, створювати образи, масштабувати контейнери, запускати зупинені контейнери тощо. Команди, пов'язані з обробкою образів, або інтерактивні опції користувача, не мають значення в Docker Compose, оскільки вони стосуються одного контейнера. Файл docker-compose.yml використовується для визначення служб додатка і містить різні опції конфігурації. Наприклад, опція build визначає опції конфігурації, такі як шлях до Dockerfile, опція command дозволяє замінити стандартні команди Docker тощо. Перша публічна бета-версія Docker Compose (версія 0.0.1) була випущена 21 грудня 2013 року. Перша версія, готова до виробництва (1.0), стала доступною 16 жовтня 2014 року.
- Docker Swarm надає вбудовану функціональність кластеризації для контейнерів Docker, яка перетворює групу двигунів Docker в один віртуальний двигун Docker. У Docker 1.12 і вище режим Swarm інтегрований з Docker Engine. Утиліта docker swarm CLI дозволяє користувачам запускати контейнери Swarm,

створювати токени виявлення, перелічувати вузли в кластері тощо. Утиліта docker node CLI дозволяє користувачам виконувати різні команди для управління вузлами в рої, наприклад, перелічувати вузли в рої, оновлювати вузли та видаляти вузли з рою. Docker управляє роями за допомогою алгоритму консенсусу Raft. Згідно з Raft, для виконання оновлення більшість вузлів Swarm повинні погодитися з оновленням. На додаток до docker swarm CLI, docker stack — це інструмент, призначений для управління службами Swarm з більшою гнучкістю. Він може використовувати файл конфігурації, дуже схожий на docker-compose.yml, з деякими нюансами. Використання docker stack замість docker compose має кілька переваг, таких як можливість керувати кластером Swarm на декількох машинах або можливість працювати з docker secret у поєднанні з docker context, функцією, яка дозволяє виконувати команди Docker на віддаленому хості, забезпечуючи віддалене керування контейнерами.

- Docker Volume сприяє незалежній стійкості даних, дозволяючи їм залишатися навіть після видалення або повторного створення контейнера.

2.3 DBeaver

DBeaver — це програмне забезпечення для роботи з SQL та інструмент для адміністрування баз даних. Для реляційних баз даних воно використовує інтерфейс прикладного програмування (API) JDBC для взаємодії з базами даних через драйвер JDBC. Для інших баз даних (NoSQL) він використовує власні драйвери баз даних. Він надає редактор, який підтримує автодоповнення коду та підсвічування синтаксису. Він надає архітектуру плагінів (на основі архітектури плагінів Eclipse), яка дозволяє користувачам змінювати більшу частину поведінки програми, щоб забезпечити функціональність, специфічну для бази даних, або функції, незалежні від бази даних. Він написаний на Java і базується на платформі Eclipse.

Функції DBeaver включають:

- Виконання SQL-запитів

- Браузер/редактор даних з величезною кількістю функцій
- Підсвічування синтаксису та автозавершення SQL
- Перегляд та редагування структури бази даних (метаданих)
- Управління SQL-скриптами
- Генерація DDL
- Рендеринг ERD (діаграм відносин між сутностями)
- SSH-тунелювання
- Підтримка SSL (MySQL та PostgreSQL)
- Експорт/міграція даних
- Імпорт, експорт та резервне копіювання даних (MySQL та PostgreSQL)
- Генерація фіктивних даних для тестування баз даних

Існують відмінності у функціях, доступних для різних баз даних.

DBeaver включає розширену підтримку наступних баз даних:

- TiDB
- MySQL та MariaDB
- PostgreSQL
- Greenplum
- Oracle
- IBM Db2
- Exasol
- SQL Server
- Mimer SQL
- Sybase
- Firebird
- Teradata
- Vertica
- SAP HANA

- Apache Phoenix
- Netezza
- Informix
- Apache Derby
- H2
- Salesforce Data Cloud
- SQLite
- SnappyData
- Snowflake
- Будь-яка інша база даних, яка має драйвер JDBC або ODBC.

2.4 .NET 9 SDK

.NET 9 SDK [1] — це набір засобів розробки програмного забезпечення для створення додатків за допомогою .NET 9. Він включає .NET Runtime, ASP.NET Core [3] Runtime, а також різні інструменти, бібліотеки та функції, що полегшують розробку.

Ключові аспекти .NET 9 SDK:

- Компоненти: містить необхідні компілятори, відладчики, бібліотеки та інструменти для розробки додатків, орієнтованих на .NET 9 Runtime.
- Узгодження версій: Основні та другорядні версії .NET SDK узгоджуються з відповідною версією .NET Runtime, яку він підтримує (наприклад, .NET 9 SDK підтримує .NET 9 Runtime).
- Функції: .NET 9 впроваджує нові функції та вдосконалення в різних областях, включаючи вдосконалення мови C#, оптимізацію продуктивності, оновлення інструментів та конкретні вдосконалення для розробки настільних та мобільних додатків (наприклад, .NET MAUI).

- Управління робочим навантаженням: SDK включає такі інструменти, як історія робочого навантаження dotnet, для управління та відстеження інсталяцій і модифікацій робочого навантаження для певної інсталяції SDK.
- Підтримка платформ: .NET 9 SDK підтримує розробку для різних операційних систем, включаючи Windows, macOS і Linux, з конкретними середовищами виконання та функціями, доступними для кожної платформи.
- Інтеграція: SDK інтегрується з середовищами розробки, такими як Visual Studio та Visual Studio Code, забезпечуючи комплексний досвід розробки.
- Microsoft Learn: офіційна документація на Microsoft Learn містить вичерпну інформацію про новинки в .NET 9, включаючи оновлення SDK та інструментів, істотні зміни та вимоги до версій.

2.5 Microsoft SQL Server

Microsoft SQL Server — це пропрієтарна система управління реляційними базами даних, розроблена компанією Microsoft з використанням мови структурованих запитів (SQL, часто вимовляється як «секвел»). Як сервер баз даних, це програмний продукт, основною функцією якого є зберігання та вилучення даних на запит інших програмних додатків, які можуть працювати як на тому самому комп'ютері, так і на іншому комп'ютері в мережі (включно з Інтернетом). Microsoft продає щонайменше десяток різних версій Microsoft SQL Server, призначених для різних аудиторій і для робочих навантажень, що варіюються від невеликих одномашинних додатків до великих інтернет-додатків з великою кількістю одночасних користувачів.

Протокольний рівень реалізує зовнішній інтерфейс до SQL Server. Всі операції, які можуть бути викликані на SQL Server, передаються йому через визначений Microsoft формат, який називається Tabular Data Stream (TDS). TDS — це протокол прикладного рівня, який використовується для передачі даних між сервером бази даних і клієнтом. Спочатку розроблений і створений компанією

Sybase Inc. для свого реляційного механізму бази даних Sybase SQL Server у 1984 році, а пізніше компанією Microsoft у Microsoft SQL Server, пакети TDS можуть бути вбудовані в інші фізичні протоколи, що залежать від транспорту, включаючи TCP/IP, іменовані канали та спільну пам'ять. Отже, доступ до SQL Server доступний через ці протоколи. Крім того, API SQL Server також доступний через веб-сервіси.

2.6 Node.js

Node.js — це кросплатформна середовище виконання JavaScript з відкритим кодом, яке може працювати на Windows, Linux, Unix, macOS та інших операційних системах. Node.js працює на движку V8 JavaScript і виконує код JavaScript поза веб-браузером. Згідно з опитуванням розробників Stack Overflow, Node.js є однією з найпоширеніших веб-технологій.

Node.js дозволяє розробникам використовувати JavaScript для написання командних рядків та серверних скриптів. Можливість запускати код JavaScript на сервері часто використовується для генерації динамічного вмісту веб-сторінки перед її відправкою до веб-браузера користувача. Отже, Node.js представляє парадигму «JavaScript всюди», об'єднуючи розробку веб-додатків навколо єдиної мови програмування, на відміну від використання різних мов для серверного та клієнтського програмування.

Node.js має архітектуру, керовану подіями, здатну до асинхронного вводу-виводу. Ці конструктивні рішення спрямовані на оптимізацію пропускної здатності та масштабованості веб-додатків з великою кількістю операцій вводу-виводу, а також для веб-додатків, що працюють у режимі реального часу (наприклад, програми для спілкування в режимі реального часу та браузерні ігри)

Проект розподіленої розробки Node.js раніше керувався Node.js Foundation, а зараз об'єднався з JS Foundation, утворивши OpenJS Foundation. OpenJS Foundation підтримується програмою спільних проектів Linux Foundation.

2.7 React

React (також відомий як React.js або ReactJS) [5] — це безкоштовна бібліотека JavaScript з відкритим кодом, яка має на меті зробити створення користувацьких інтерфейсів на основі компонентів більш «безшовним». Її підтримує Meta (раніше Facebook) та спільнота індивідуальних розробників і компаній. Згідно з опитуванням розробників Stack Overflow, React є однією з найпоширеніших веб-технологій.

React можна використовувати для розробки односторінкових, мобільних або серверних додатків за допомогою фреймворків, таких як Next.js і React Router. Оскільки React займається лише користувацьким інтерфейсом і рендерингом компонентів у DOM, додатки React часто покладаються на бібліотеки для маршрутизації та інших функцій на стороні клієнта. Ключовою перевагою React є те, що він перерендерить лише ті частини сторінки, які змінилися, уникаючи непотрібного перерендерингу незмінних елементів DOM. React використовується приблизно 6% всіх веб-сайтів.

React дотримується парадигми декларативного програмування. Розробники проектують види для кожного стану додатка, а React оновлює та візуалізує компоненти при зміні даних. Це контрастує з імперативним програмуванням.

Код React складається з об'єктів, які називаються компонентами. Ці компоненти є модульними і можуть бути використані повторно. Додатки React зазвичай складаються з багатьох шарів компонентів. Компоненти візуалізуються в кореневому елементі DOM за допомогою бібліотеки React DOM. Під час візуалізації компонента значення передаються між компонентами через props (скорочення від «properties»). Внутрішні значення компонента називаються його станом.

РОЗДІЛ 3
МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Формалізація предметної області

У системі інтелектуального керування розкладом навчального процесу важливою передумовою побудови ефективного алгоритмічного ядра є формалізація предметної області. Це дозволяє перейти від опису логіки на рівні користувацьких дій до математичного подання даних, обмежень і процесів, які лежать в основі управління розкладом. Формалізація охоплює сутності (групи, викладачі, аудиторії, предмети), правила взаємодії між ними, обмеження часу та простору, а також дії користувачів.

Розклад розглядається як **множина упорядкованих подій**, де кожна подія — це пара (або інший навчальний захід), що характеризується наступними атрибутами:

- G — група студентів
- S — навчальний предмет
- T — викладач
- R — аудиторія
- D — дата
- P — порядковий номер пари (1–5)

Таким чином, одну пару можна представити як 6-мірний вектор:

$$Pair = (G, S, T, R, D, P) \tag{3.1}$$

3.2 Базові множини

Позначимо множини:

- G — множина всіх навчальних груп
- S — множина всіх предметів
- T — множина викладачів
- R — множина аудиторій

- D — допустимий календарний період
- P = {1,2,3,4,5} — множина номерів пар

Пари формуються згідно з функцією відображення:

$$f: G \times S \times T \times R \times D \times P \rightarrow \text{Розклад} \quad (3.2)$$

Обмеження

Система повинна дотримуватись набору обмежень, які з математичної точки зору можна представити як логічні предикати. Основні обмеження включають:

- Унікальність ресурсу в момент часу:
  - Один викладач не може читати дві пари одночасно:

$$\forall T_i, D_k, P_m : \text{Count}(T_i, D_k, P_m) \leq 1 \quad (3.3)$$

  - Аудиторія не може бути використана одночасно двома групами:

$$\forall R_j, D_k, P_m : \text{Count}(R_j, D_k, P_m) \leq 1 \quad (3.4)$$

- Наявність викладача у викладанні предмета:
  - T ∈ Teachers(S), тобто викладач закріплений за предметом
- Група повинна бути закріплена за університетом:
  - G∈Groups(U), де U — університет користувача

Модель користувача

Кожен користувач системи належить до певної ролі R∈{Admin,Teacher,Student} що визначає допустимі операції над розкладом:

- Admin → повний доступ
- Teacher → доступ до створення та редагування предметів і пар
- Student → лише читання через Telegram-інтерфейс

Користувачі мають обмежений контекст, тобто взаємодіють лише з даними свого університету:

$$\text{Scope}(U_i d) = \{G, T, S\} \text{ де } G.\text{UniversityId} = T.\text{UniversityId} = S.\text{UniversityId} = U_i d \quad (3.5)$$

3.3 Алгоритмічне забезпечення розкладу

Алгоритмічне забезпечення в системі управління розкладом відіграє центральну роль, оскільки саме воно відповідає за логіку створення, перевірки, збереження, видалення та повторення навчальних занять. У запропонованій системі реалізовано підхід, що дозволяє створювати гнучкий, повторюваний та легко модифікований розклад для кожної навчальної групи.

Основна ідея алгоритму полягає у створенні циклічного шаблону розкладу, що базується на навчальних тижнях — перший, другий та третій тиждень. Адміністратор або викладач задає період, у межах якого цей тижневий шаблон має циклічно повторюватися. Наприклад, якщо період встановлено з 1 вересня по 31 грудня, то після третього тижня знову повторюється перший, і так далі до кінця періоду.

Побудова розкладу

1. Створення тижневого шаблону. Для кожного тижня (1–3) користувач може додати заняття по днях і парах (1–5).

2. Вибір періоду дії. Задаються startDate і endDate — межі, в яких обраний шаблон буде повторюватися.

3. Генерація розкладу. На основі обраного шаблону і періоду, система автоматично створює розклад занять по всіх датах у періоді, враховуючи номер тижня в циклі.

4. Валідація. Перед збереженням кожної пари перевіряється, чи не зайнятий цей час іншою парою для тієї ж групи, викладача чи аудиторії.

Взаємодія з користувачем

Система дозволяє:

- видаляти як окрему пару (по даті, групі та парі), так і весь розклад групи;
- додавати індивідуальні заняття на вільний час без порушення шаблону;
- перевіряти на конфлікти перед додаванням нових пар.

При додаванні нової пари, алгоритм перевіряє, чи не існує вже пари:

- для цієї групи на цю дату та пару (PairNumber);
- чи викладач не має іншого заняття в цей самий час;
- чи не зайнята вказана аудиторія іншим предметом.

Структура алгоритму перевірки:

1. Отримати список пар для заданої дати та номера пари.

2. Для кожної знайденої пари перевірити:

- Чи співпадає GroupId?

- Чи співпадає TeacherId?

- Чи співпадає дата та номер заняття?

3. Якщо є збіг — показати повідомлення про конфлікт.

4. Інакше — дозволити збереження.

Гнучкість змін

- Якщо потрібно оновити лише одну пару — змінюється лише запис у таблиці Schedules за певною датою.
- Якщо потрібно змінити весь шаблон — оновлення здійснюється на рівні WeeklySchedules, а старі пари видаляються.
- Видалення розкладу — очищає записи в заданому діапазоні дат для певної групи.

Переваги підходу

- Масштабованість. Один шаблон може покривати довгий період без дублювання даних.
- Гнучкість. Можна як працювати з шаблоном, так і вручну втручатись в окремі дні.
- Прозорість. Користувач бачить, які пари буде створено ще до збереження.

Таким чином, алгоритмічне забезпечення в даній системі не просто автоматизує рутинну роботу зі створення розкладу, а й дозволяє створювати гнучкі шаблони, які легко масштабуються, підтримуються і можуть бути змінені без ризику порушення логіки або виникнення конфліктів.

3.4 Система автоматичного сповіщення

Система автоматичного сповіщення — це окремий логічний компонент, інтегрований у програмну платформу управління розкладом, який відповідає за надсилання повідомлень студентам через Telegram-бота. Вона реалізована на базі бібліотеки Hangfire [4], що дозволяє налаштовувати відкладене та періодичне виконання фонових завдань, не порушуючи основну логіку роботи застосунку.

Загальна ідея

Щодня вранці система формує повідомлення для кожного зареєстрованого Telegram-користувача, який підписаний на сповіщення. Повідомлення включає список пар на поточний день. У випадку скасування пар або змін у розкладі протягом дня, система може повторно сповістити про оновлення. Таким чином, забезпечується інформованість студентів у режимі реального часу.

Технічна реалізація

Серверна частина використовує Hangfire для організації періодичних завдань:

- завдання на 07:30 щодня — надсилання розкладу на поточний день;
- завдання на запит (через API або зміну в розкладі) — повторне сповіщення;

Після запуску API, у методі Configure (або Program.cs) налаштовується періодична задача:

```
RecurringJob.AddOrUpdate(
   "daily-schedule-notification",
   () => telegramNotificationService.SendDailySchedulesAsync(),
   cronExpression);
```

Алгоритм роботи автоматичного сповіщення

1. Отримання поточної дати DateOnly.Today.
2. Визначення всіх користувачів з увімкненою Telegram-підпискою (IsTelegramSubscribed == true).
3. Для кожного користувача:
    - Перевірка, чи має він зв’язок з UniversityId та GroupId.

- Отримання розкладу на сьогодні для цієї групи.

4. Побудова повідомлення.
5. Надсилання повідомлення через Telegram Bot API.
6. Логування успішних/неуспішних спроб.

Масштабованість і безпечність

Оскільки Hangfire дозволяє паралельне виконання задач і має власну чергу обробки, система може масштабуватися під більшу кількість користувачів. Уся обробка виконується у фоновому режимі, не блокуючи основні запити до API.

Також можлива реалізація розсилок за іншими подіями:

- Реєстрація в системі.
- Підтвердження вибору групи.
- Щотижнева розсилка з оновленнями.

Автоматична система сповіщень дозволяє підтримувати зворотній зв'язок між системою управління розкладом та кінцевим користувачем. Вона мінімізує ризик того, що студент пропустить заняття або не дізнається про зміни в розкладі, тим самим підвищуючи зручність і надійність користування платформою.

3.5 Інтеграція математичних моделей у Onion-архітектуру

Однією з важливих особливостей побудови сучасних систем керування є поєднання структурного підходу до архітектури програмного забезпечення з формальним описом бізнес-логіки у вигляді математичних моделей. У запропонованій системі розкладу інтеграція математичних моделей (формалізація обмежень, алгоритм генерації розкладу, логіка перевірки конфліктів тощо) реалізована відповідно до принципів Onion-архітектури.

Основи Onion-архітектури

Onion-архітектура в системі EduNotify реалізована через розподіл проєкту на окремі проєкти/модулі, кожен із яких відповідає певному логічному шару. Структура виглядає наступним чином:

- EduNotify.Domain

Містить основні сутності домену (Entities), перерахування (Enums), інтерфейси репозиторіїв. Саме тут формуються математичні об'єкти предметної області — SchedulesEntity, SubjectEntity, GroupEntity тощо. Всі обмеження, такі як обов’язковість полів, типи пари, прив'язка до університету, виражені через класи сутностей.

- EduNotify.Application

Реалізує прикладну логіку та алгоритми: генерацію розкладу, перевірку конфліктів, інтерфейси сервісів .У цьому шарі безпосередньо реалізовані алгоритми перевірки конфліктів між викладачами, групами та аудиторіями, перетворення тижневого шаблону у щоденний розклад тощо.

- EduNotify.Infrastructure

Реалізує інтерфейси з домену та аплікейшну, надаючи доступ до зовнішніх сервісів — Telegram-бота, логування, Hangfire. Саме тут реалізується TelegramBotService, який викликає сервіси з Application, не змінюючи логіку розкладу, а лише відправляючи сформовані повідомлення.

- EduNotify.Persistence

Відповідає за роботу з базою даних (наприклад, EF Core реалізації інтерфейсів IGroupRepository, IScheduleRepository), міграції, конфігурацію контексту. Цей шар залежить від Domain та Application, але сам не містить логіки перевірок — він лише інструмент збереження.

- EduNotify.Api

Презентаційний рівень, який реалізує точки доступу для користувачів — контролери HTTP. Через DI (Dependency Injection) звертається до сервісів з Application. Тут не реалізується жодна перевірка — всі обмеження йдуть зсередини системи.

- EduNotify.Common

Допоміжна бібліотека, яка містить загальні інструменти: DTO, валідатори,

константи, відповіді (ApiResponse<T>), винятки тощо. Вона використовується всіма шарами, де це доцільно.

Таким чином, структура рішення повністю відповідає принципам Onion-архітектури, де:

- ядро (Domain) не залежить ні від чого;
- Application залежить тільки від ядра;
- Infrastructure та Persistence реалізують інтерфейси з ядра й прикладної логіки;
- API взаємодіє з Application, не маючи прямого доступу до БД або логіки.

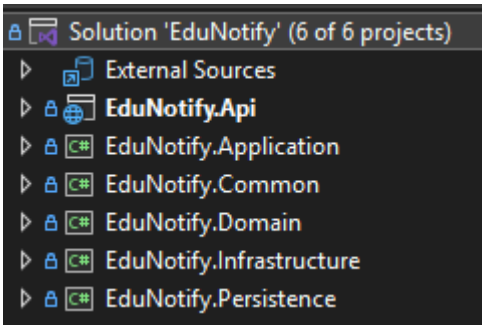


Рисунок 3.1 – Структура EduNotify проєкту.

Вбудовування моделей у домен

У Domain Layer зберігаються сутності SchedulesEntity, GroupEntity, SubjectEntity, які формують основу предметної області. Окрім базових полів, ці класи можуть містити математичні обмеження, описані у вигляді атрибутів або службових методів (наприклад, перевірка, чи допустимий PairNumber).

Формалізація обмежень типу:

$$\forall T_i, D_k, P_m : \text{Count}(T_i, D_k, P_m) \leq 1 \qquad (3.6)$$

реалізується через методи-валидатори в Application Layer, що використовують інтерфейси репозиторіїв із Domain Layer.

Використання сервісів у Infrastructure

У Infrastructure Layer реалізовано:

- підключення до бази даних через EF Core;
- TelegramBotService — споживач сервісів Application Layer;

- Алгоритм створення розкладу
- HangfireJobs — задачі для автоматичної генерації, оновлення, перевірки даних.

При цьому Infrastructure не змінює математичну модель, а лише її використовує.

Взаємодія з API та ботом

У Presentation/API шарі реалізовані:

- контролери (ScheduleController, SubjectController) для створення, редагування та перегляду розкладу;
- виклики Application-сервісів без прямого доступу до даних;
- TelegramBotService — викликає методи Application для формування повідомлень.

Таким чином, математична модель — наприклад, правила розміщення пар у тижні чи унікальність пар по викладачах — дотримується автоматично завдяки централізованій бізнес-логіці.

Інтеграція математичних моделей у структуру Onion-архітектури дозволяє:

- чітко розмежувати відповідальність;
- забезпечити перевірку та валідацію на рівні бізнес-логіки;
- мінімізувати дублювання коду;
- повторно використовувати алгоритми як для API, так і для Telegram-бота.

РОЗДІЛ 4.
ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Основні функціональні вимоги до системи

Інтелектуальна система управління розкладом призначена для автоматизації процесів формування, редагування та перегляду навчального розкладу в закладах вищої освіти. Основна мета полягає у створенні зручного та гнучкого інструменту для адміністраторів, викладачів і студентів, який забезпечує ефективну взаємодію з навчальним процесом та оперативне інформування користувачів про зміни в розкладі.

Функціональність системи базується на ролях користувачів. Кожна роль має свій набір прав і можливостей:

Адміністратор — найвищий рівень доступу. Його основні функції включають створення університетів, додавання та редагування облікових записів викладачів, моніторинг активності, а також загальний контроль над розкладом та структурою даних. Адміністратор може бачити повну картину всієї системи, включаючи всі університети, викладачів, групи, предмети та розклади. Йому також доступна функція перегляду статистики, створення резервних копій та оновлення системи.

Викладач — основний користувач системи, відповідальний за створення навчального контенту. Викладач має змогу додавати нові предмети, створювати навчальні групи, формувати розклад, редагувати або видаляти пари. Інтерфейс викладача дозволяє здійснювати фільтрацію розкладу за датою, групою або предметом. Викладач може бачити лише ті групи, предмети та розклади, які створені ним або належать до його університету.

Студент — кінцевий користувач, який отримує інформацію про розклад. Студенти не мають доступу до вебінтерфейсу, а взаємодіють із системою через Telegram-бот. Основна функція бота — надання актуальної інформації про пари на день або тиждень, а також надсилання сповіщень про скасування або зміну пар. Бот

ідентифікує користувача за Telegram ID і дозволяє прив’язати його до конкретної групи.

Важливою функціональною вимогою є реалізація аутентифікації та авторизації. Для цього використовується механізм JWT (JSON Web Token), що дозволяє забезпечити безпечний обмін даними між клієнтом та сервером. Кожен авторизований користувач отримує токен, який прикріплюється до всіх наступних HTTP-запитів і використовується для визначення його ролі та дозволів. У фронтенді це дозволяє динамічно обмежувати або відкривати доступ до відповідних сторінок та компонентів.

Окремо варто згадати про валідацію даних. Вона реалізується як на клієнтському, так і на серверному рівнях. Для цього використовується бібліотека FluentValidation, яка дозволяє описувати правила валідації у вигляді окремих класів. Це гарантує, що при збереженні даних у базі не буде дублювання або логічних помилок (наприклад, створення пари без зазначення предмета чи групи).

Система повинна підтримувати можливість масштабування та розширення функціоналу. В майбутньому передбачається можливість додавання нових ролей, підтримка декількох мов, синхронізація з іншими платформами та інтеграція календаря. Для забезпечення цього використовується Onion-архітектура, яка дозволяє розділити систему на логічні шари: доменну модель, сервісну логіку, інфраструктуру та API-рівень. Такий підхід значно полегшує розширення функціоналу, тестування і супровід системи.

Підсумовуючи, функціональні вимоги до системи охоплюють не лише створення та редагування розкладу, а й забезпечення безпеки, масштабованості, розширюваності, зручності інтерфейсу та ефективного сповіщення користувачів. Реалізація цих вимог дозволяє сформувати надійну та сучасну систему управління навчальним розкладом, що відповідає актуальним потребам закладів вищої освіти.

4.2 Модель користувачів та авторизація

Модель користувачів у системі управління розкладом побудована на основі чіткого розмежування прав доступу за допомогою ролевої архітектури. Основні типи користувачів — це адміністратори, викладачі та студенти. Для кожної з ролей визначено окремий набір можливостей, а також обмеження щодо доступу до ресурсів системи. Такий підхід дозволяє не лише підвищити безпеку, а й оптимізувати інтерфейс взаємодії для кожної категорії користувачів, приховуючи непотрібний функціонал.

Аутентифікація у системі реалізована за допомогою JWT (JSON Web Token) — сучасного механізму, який дозволяє безпечно ідентифікувати користувача після входу в систему. Після авторизації сервер генерує токен, який містить основну інформацію про користувача: його ID, роль, тривалість дії токена. Цей токен зберігається на клієнті (наприклад, у localStorage або sessionStorage) і автоматично прикріплюється до всіх наступних HTTP-запитів до сервера через заголовок Authorization. Таким чином, користувач не вводить логін і пароль кожного разу, але сервер має змогу перевірити його особу.

Додатково реалізована функція оновлення токена (refresh token), яка дозволяє користувачеві залишатися в системі протягом тривалого часу без повторної авторизації. Коли основний токен закінчує термін дії, клієнт автоматично відправляє refresh token на сервер і отримує новий токен доступу. Це забезпечує зручність використання без зниження рівня безпеки.

Для забезпечення гнучкості доступу реалізовано атрибути контролю доступу на рівні API. Наприклад, у контролерах .NET використовуються атрибути [Authorize(Roles = "Admin")], що дозволяє точно вказати, яка роль має доступ до конкретного ендпоінту. Це значно спрощує розмежування доступу на серверній частині та підвищує контроль над безпечністю системи.

Користувацька модель зберігається у базі даних у вигляді таблиці Users, яка має поля Id, Email, PasswordHash, Role, UniversityId тощо. Для зберігання паролів

використовується алгоритм хешування (наприклад, SHA-256 або bcrypt), що унеможливлює доступ до паролів навіть при компрометації бази. Крім того, система логування фіксує всі спроби входу, помилки аутентифікації та зміну ролей, що дозволяє проводити аудит безпеки.

Узагальнюючи, модель користувачів і система авторизації є ключовими компонентами безпеки та зручності використання системи. Розмежування прав, гнучке керування ролями, токен-базована аутентифікація і контроль доступу на рівні API створюють надійну основу для подальшого масштабування системи та її інтеграції з іншими сервісами.

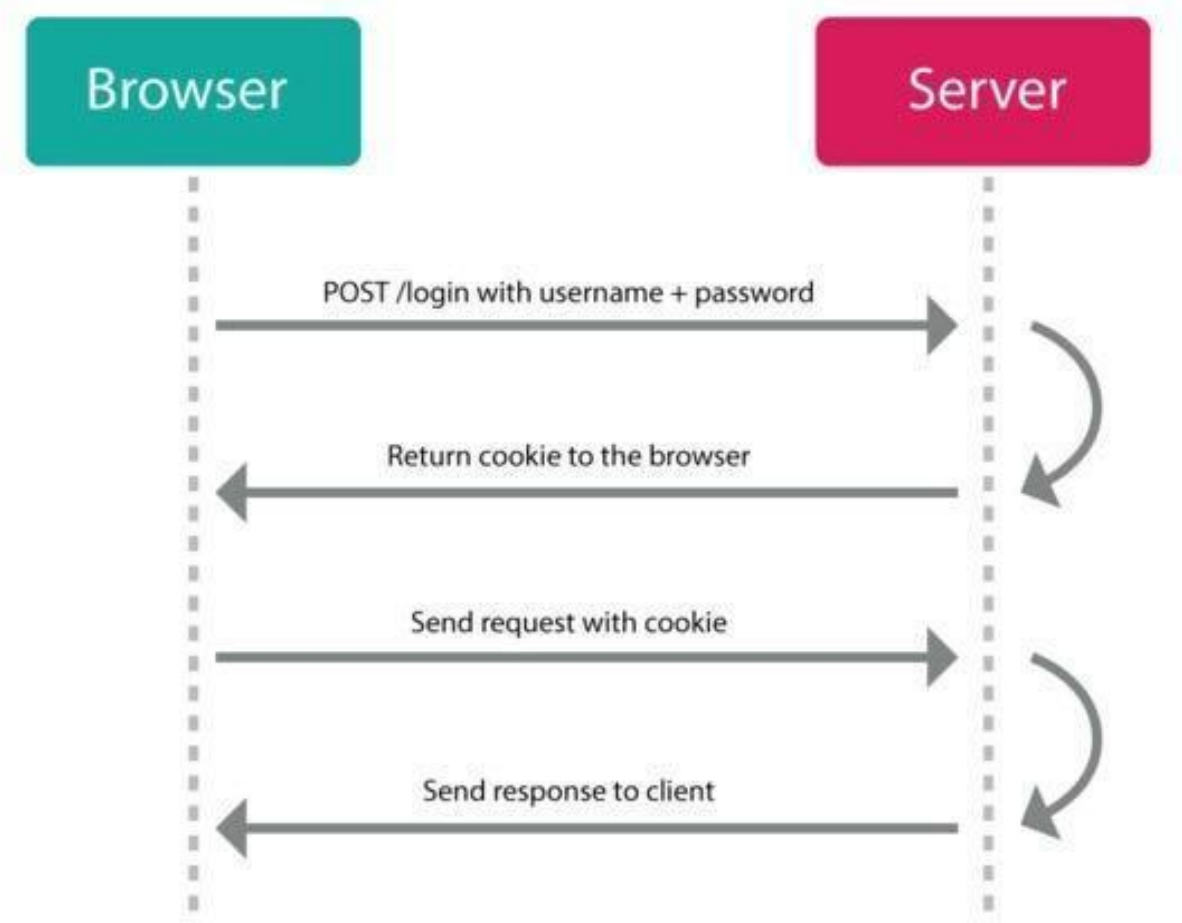


Рисунок 3.1 – Візуалізація використання JWT токена.

Авторизація, у свою чергу, забезпечує обмеження доступу до функціоналу в залежності від ролі користувача. Наприклад, користувач із роллю "Teacher" не має доступу до створення інших користувачів або зміни налаштувань університету. У фронтенді це реалізовано шляхом перевірки ролі при рендерінгу компонентів або маршрутування. Таким чином, непотрібні сторінки та елементи інтерфейсу не відображаються для ролей, яким вони не призначені.

Додатково реалізована функція оновлення токена (refresh token), яка дозволяє користувачеві залишатися в системі протягом тривалого часу без повторної авторизації. Коли основний токен закінчує термін дії, клієнт автоматично відправляє refresh token на сервер і отримує новий токен доступу. Це забезпечує зручність використання без зниження рівня безпеки.

Для забезпечення гнучкості доступу реалізовано атрибути контролю доступу на рівні API. Наприклад, у контролерах .NET використовуються атрибути [Authorize(Roles = "Admin")], що дозволяє точно вказати, яка роль має доступ до конкретного ендпоінту. Це значно спрощує розмежування доступу на серверній частині та підвищує контроль над безпечністю системи.

Користувацька модель зберігається у базі даних у вигляді таблиці Users, яка має поля Id, Email, PasswordHash, Role, UniversityId тощо. Для зберігання паролів використовується алгоритм хешування (наприклад, SHA-256 або bcrypt), що унеможливлює доступ до паролів навіть при компрометації бази. Крім того, система логування фіксує всі спроби входу, помилки аутентифікації та зміну ролей, що дозволяє проводити аудит безпеки.

Узагальнюючи, модель користувачів і система авторизації є ключовими компонентами безпеки та зручності використання системи. Розмежування прав, гнучке керування ролями, токен-базована аутентифікація і контроль доступу на рівні API створюють надійну основу для подальшого масштабування системи та її інтеграції з іншими сервісами.

4.3 Інформаційні сутності та їх взаємозв’язки

Інформаційна модель системи управління розкладом базується на реляційній структурі бази даних, яка відображає всі сутності та взаємозв’язки між ними. Основні сутності включають університети, користувачів, ролі, предмети, групи, розклади, а також додаткові сутності для зберігання зв’язків та службової інформації. Побудова правильної структури є критично важливою для забезпечення логічної цілісності, коректності запитів та ефективного функціонування всієї системи.

Центральним елементом є таблиця AspNetUsers, яка містить основну інформацію про користувачів системи: ID, роль (через зв’язок з таблицями AspNetUserRoles та AspNetRoles), університет, групу, контактні дані. Через поле UniversityId кожен користувач пов’язаний з певним університетом із таблиці Universities. Для викладачів додатково використовується таблиця зв’язку SubjectEntityUserEntity, яка реалізує зв’язок «багато до багатьох» між предметами та викладачами.

Таблиця Subjects містить перелік навчальних дисциплін, що викладаються в межах університету. Кожен предмет має свій унікальний ідентифікатор (Id) і зовнішній ключ UniversityId, що вказує на заклад освіти. Один предмет може викладатися кількома викладачами, а один викладач — викладати кілька предметів.

Структура навчального процесу представлена таблицею Groups, де зберігаються навчальні групи. Кожна група має зовнішній ключ UniversityId, що вказує на університет, у якому вона існує. Крім того, користувачі можуть бути прив’язані до груп через поле GroupId у таблиці AspNetUsers — це використовується в основному для студентів, які взаємодіють із розкладом через Telegram-бота.

Таблиця Schedules представляє розклади, які зв’язують групи з предметами через відповідні зовнішні ключі GroupId та SubjectId. Це дозволяє побудувати логіку відображення розкладу на конкретний день чи тиждень для кожної групи.

Записи в розкладі є агрегованими — тобто один розклад може включати багато пар. Кожен запис у розкладі відповідає унікальній комбінації група-предмет і використовується для формування щоденних повідомлень.

Для забезпечення гнучкості доступу до системи реалізована повна підтримка ролевої моделі. Таблиці AspNetRoles, AspNetUserRoles, AspNetUserClaims та AspNetRoleClaims реалізують прив’язку прав доступу до користувачів. Наприклад, роль "Teacher" дозволяє доступ до створення та редагування розкладів, тоді як роль "Admin" дає повний контроль над усією системою. Ця модель дозволяє чітко регламентувати, який функціонал доступний для кожного користувача відповідно до його призначення.

Також у структурі бази даних присутні таблиці AspNetUserLogins та AspNetUserTokens, які відповідають за інтеграцію з системами зовнішньої аутентифікації та збереження токенів доступу. Це дає змогу реалізувати захищений доступ до API з використанням JWT-токенів. Крім цього, таблиці Relations та FAQs використовуються для реалізації додаткового функціоналу, наприклад, керування внутрішніми зв’язками користувачів або створення системи довідкової інформації.

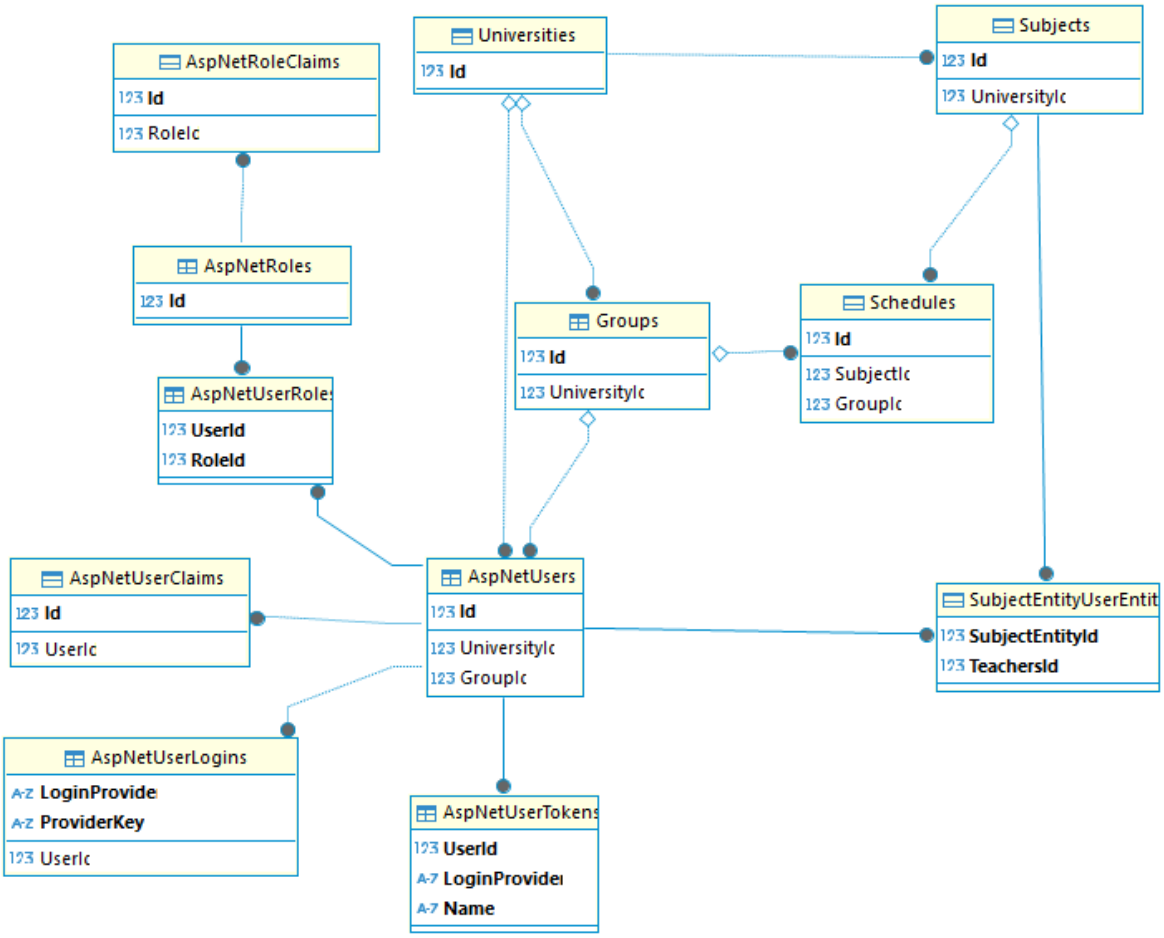


Рисунок 3.2 – ER-схемі бази даних.

На ER-схемі можна чітко прослідкувати всі залежності та типи зв’язків між таблицями: один-до-багатьох (Universities → Groups), багато-до-багатьох (Subjects ↔ Users через SubjectEntityUserEntity), один-к-одному (Users → Groups). Така структура дозволяє ефективно реалізувати всі операції, пов’язані з управлінням розкладом, забезпечити повноту даних, спростити виконання складних SQL-запитів та зменшити кількість дублювань.

Таким чином, реляційна модель є основою для зберігання та обробки інформації в системі. Її продумана структура дозволяє досягти високої продуктивності, масштабованості та зручності при подальшому розвитку функціоналу. Вона також забезпечує базу для реалізації автоматичних перевірок конфліктів у розкладі, генерації повідомлень студентам та звітності для викладачів та адміністраторів.

4.4 Рівні архітектурної підтримки даних

Інформаційна система управління розкладом побудована з використанням Onion-архітектури, яка забезпечує чітке розділення відповідальностей між різними рівнями системи. Такий підхід дозволяє досягти високої модульності, масштабованості, тестованості та стійкості до змін технологічного стеку.

Onion-архітектура базується на ідеї, що найбільш стабільні та фундаментальні частини системи (доменна модель) мають бути в центрі, тоді як менш стабільні (інфраструктура, UI) — на периферії. Усі залежності спрямовані до центру, і жоден внутрішній шар не залежить від зовнішнього [7].

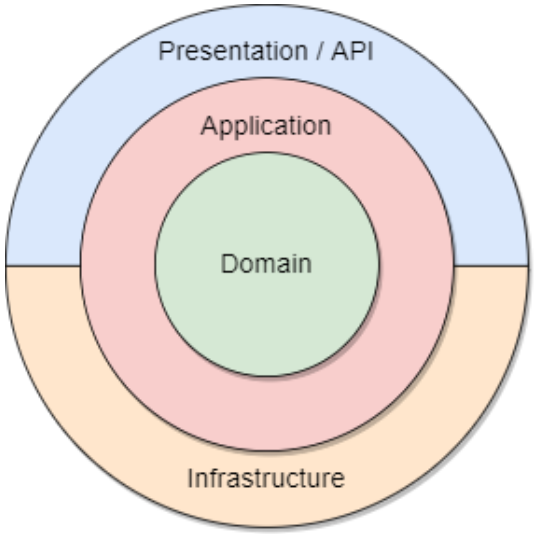


Рисунок 3.3 - Схема Onion-архітектури.

Доменний шар (Domain Layer)

Це ядро системи. Він містить:

- сутності (ScheduleEntity, SubjectEntity, UserEntity),
- інтерфейси репозиторіїв (IScheduleRepository),
- бізнес-логіку: наприклад, перевірку конфліктів занять або унікальності пари,
- авторизаційні правила для ролей.

У цьому шарі не використовуються зовнішні бібліотеки, а лише чиста логіка предметної області.

Приклад моделі юзера в Domain Layer:

```
public class UserEntity : IdentityUser<long>
{
    public string? FirstName { get; set; }
    public string? LastName { get; set; }
    public long? TelegramChatId { get; set; }
    public bool IsTelegramSubscribed { get; set; }
    public long? UniversityId { get; set; }
    public UniversityEntity? University { get; set; }
    public long? GroupId { get; set; }
    public GroupEntity? Group { get; set; }
}
```

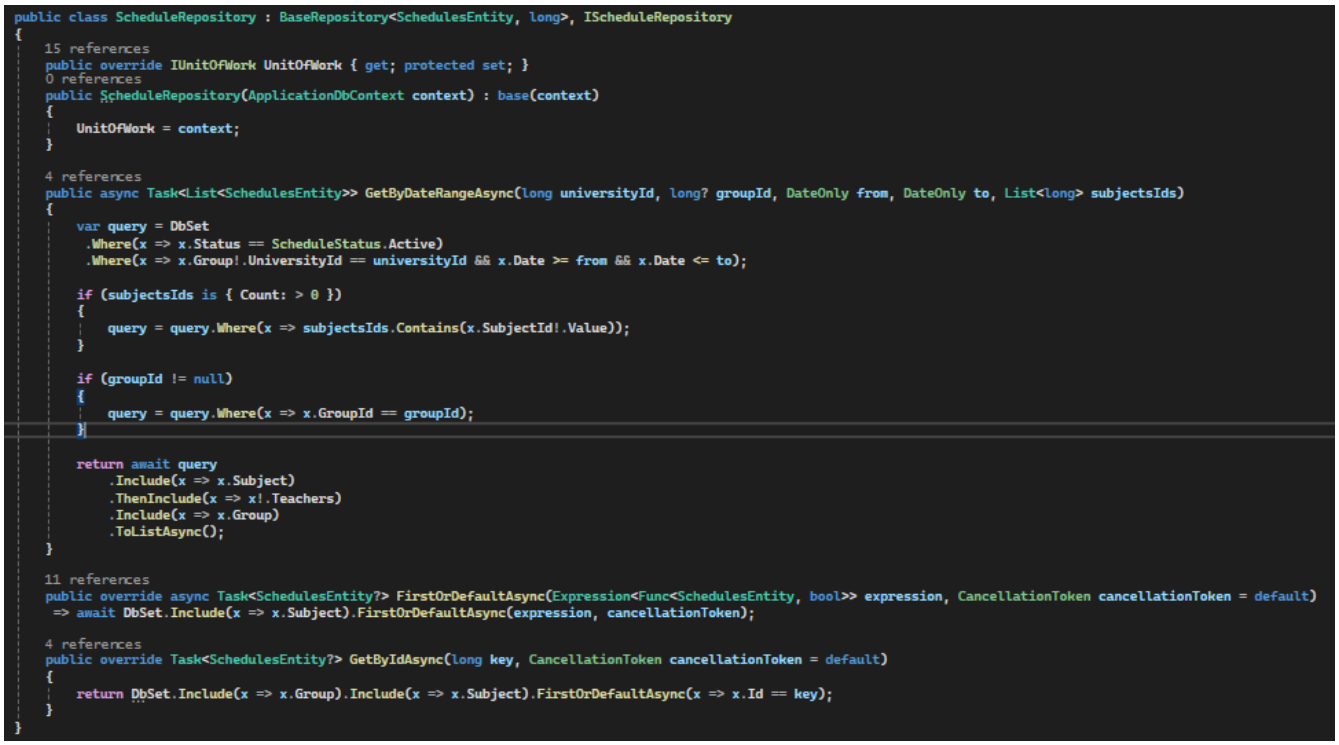
Шар застосунку (Application) У цьому шарі розміщується прикладна логіка, яка координує роботу доменної моделі. Саме тут реалізуються сценарії використання (Use Cases), які звертаються до інтерфейсів репозиторіїв з доменного шару. Цей шар знає про домен, але не про інфраструктуру. Наприклад, тут реалізуються сценарії створення пари, розкладу, редагування викладача тощо. Також тут використовуються мапер (AutoMapper) і валідатор (FluentValidation).

Інфраструктурний шар (Infrastructure Layer)

Він відповідає за реалізацію інтерфейсів з Domain і Application шарів, а також за:

- роботу з БД через Entity Framework Core;
- взаємодію з Telegram Bot API;
- логування;
- надсилання email/Telegram сповіщень;
- сервіси роботи з файлами.

Цей шар інжектується через Dependency Injection (DI).



```
public class ScheduleRepository : BaseRepository<SchedulesEntity, long>, IScheduleRepository
{
    15 references
    public override IUnitOfWork UnitOfWork { get; protected set; }
    0 references
    public ScheduleRepository(ApplicationDbContext context) : base(context)
    {
        UnitOfWork = context;
    }

    4 references
    public async Task<List<SchedulesEntity>> GetByDateRangeAsync(long universityId, long? groupId, DateOnly from, DateOnly to, List<long> subjectsIds)
    {
        var query = DbSet
         .Where(x => x.Status == ScheduleStatus.Active)
         .Where(x => x.Group!.UniversityId == universityId && x.Date >= from && x.Date <= to);

        if (subjectsIds is { Count: > 0 })
        {
            query = query.Where(x => subjectsIds.Contains(x.SubjectId!.Value));
        }

        if (groupId != null)
        {
            query = query.Where(x => x.GroupId == groupId);
        }

        return await query
            .Include(x => x.Subject)
            .ThenInclude(x => x!.Teachers)
            .Include(x => x.Group)
            .ToListAsync();
    }

    11 references
    public override async Task<SchedulesEntity?> FirstOrDefaultAsync(Expression<Func<SchedulesEntity, bool>> expression, CancellationToken cancellationToken = default)
     => await DbSet.Include(x => x.Subject).FirstOrDefaultAsync(expression, cancellationToken);

    4 references
    public override Task<SchedulesEntity?> GetByIdAsync(long key, CancellationToken cancellationToken = default)
    {
        return DbSet.Include(x => x.Group).Include(x => x.Subject).FirstOrDefaultAsync(x => x.Id == key);
    }
}
```

Рисунок 3.4 – Приклад імплементації репозиторію.

Презентаційний шар (Presentation Layer / API)

Це зовнішній шар, який реалізує:

- Web API-контролери (ASP.NET Core),
- обробку HTTP-запитів (через ScheduleController тощо),
- авторизацію/аутентифікацію (через атрибути [Authorize]),
- взаємодію з фронтендом (React) та Telegram-ботом.

Приклад контролера в презентаційному шарі:

```
 [Route("[controller]")]
 [Authorize]
 public class SubjectController(SubjectService subjectService) :
BaseController
 {
     [HttpPost]
     [ProducesResponseType(StatusCodes.Status204NoContent)]
     public async Task<IActionResult> CreateSubject([FromBody]
CreateSubjectRequest request)
     {
         var result = await
subjectService.CreateSubjectAsync(request.Name, request.TeachersIds
?? new ());
         return MapResponse(result);
     }

     [HttpGet("{universityId}")]
     [ProducesResponseType(StatusCodes.Status200OK)]
     public async Task<IActionResult> GetAll([FromRoute] long
universityId)
     {
         var result = await
subjectService.GetAllSubjectsByUniversityIdAsync(universityId);
         return MapResponse(result);
     }

     [HttpGet]
     [ProducesResponseType(StatusCodes.Status200OK)]
     public async Task<IActionResult> GetUserSubjects()
     {
         var result = await subjectService.GetUserSubjectsAsync();
         return MapResponse(result);
     }

     [HttpDelete("{id}")]
     [Authorize(Roles = IdentityRoles.Teacher)]
     [ProducesResponseType(StatusCodes.Status200OK)]
     public async Task<IActionResult> DeleteById([FromRoute] long
id)
     {
         var result = await subjectService.DeleteSubjectAsync(id);
         return MapResponse(result);
     }
 }
```

Інверсія залежностей Ключовий принцип Onion-архітектури — інверсія залежностей [10]. Наприклад, замість того щоб доменна логіка залежала від

реалізації з EF Core, у домені визначено інтерфейс IScheduleRepository, а його реалізація через ScheduleRepository : IScheduleRepository розміщується в Infrastructure. Це дозволяє легко замінити технології (наприклад, перейти з EF на MongoDB) без зміни бізнес-логіки. Приклад використання Коли викладач створює розклад, API приймає DTO, далі передає дані до сервіса який в свою чергу викликає репозиторій. Далі, через інтерфейси репозиторіїв, дані зберігаються в базу, а результат повертається назад. Вся логіка перевірки конфліктів, перевірки повноважень викладача відбувається у внутрішніх шарах, а не в контролері.

Переваги підходу

• Тестованість: легко тестувати Application і Domain шари окремо, без підключення БД [11];

• Гнучкість: зміна інтерфейсів користувача або джерел зберігання даних не потребує змін у бізнес-логіці;

• Підтримка: легше підтримувати код, оскільки відповідальність чітко розділена;

• Безпека: логіка перевірок зосереджена у внутрішніх шарах, що унеможливлює її обхід.

```
1 reference
public static void AddRepositories(this IServiceCollection services)
{
    services.AddTransient<IUniversityRepository, UniversityRepository>();
    services.AddTransient<IGroupRepository, GroupRepository>();
    services.AddTransient<ISubjectRepository, SubjectRepository>();
    services.AddTransient<IScheduleRepository, ScheduleRepository>();
}
```

Рисунок 3.5 – Приклад реєстрації репозиторіїв в DI контейнері.

Таким чином, Onion-архітектура дозволяє побудувати стабільну, масштабовану систему з чітким розділенням відповідальності, де дані проходять кілька рівнів обробки, перш ніж потрапити в базу або повернутись користувачу. Це створює умови для розширення проєкту без порушення існуючої логіки та підвищує загальну якість коду.

4.5 Інтеграція Telegram-бота як джерела/отримувача даних

Одним з ключових каналів взаємодії студентів із системою є Telegram-бот. Його інтеграція дозволяє реалізувати швидкий та зручний доступ до актуального розкладу занять, зворотний зв’язок із системою, а також надсилання автоматичних повідомлень про зміни. Бот реалізований з використанням бібліотеки Telegram.Bot і функціонує як споживач вебхуків (Webhook) [13], обробляючи запити користувачів у режимі реального часу.

Основна логіка роботи

Telegram-бот працює на основі централізованого методу HandleUpdateAsync, який обробляє всі вхідні події — текстові повідомлення, контакти, команди. Для кожної команди реалізований окремий обробник [13]:

- /start — початок роботи з ботом, перевірка існування користувача;
- налаштування — меню вибору університету та групи;
- обрати університет / групу — виклик списків та збереження вибору;
- розклад на сьогодні / тиждень — відправка відповідного розкладу;
- допомога — відображення списку команд.

Усі відповіді відправляються через Telegram API з використанням розмітки Markdown.

Зв’язок з системою

- Telegram-бот напряму взаємодіє з такими сервісами:
- UserManager — перевірка, створення та оновлення користувачів;
- UserService — реєстрація нових Telegram-користувачів;
- UniversityRepository, GroupRepository — для отримання списків університетів і груп;
- ScheduleRepository — для отримання розкладу.

Після авторизації користувач має змогу прив’язати Telegram-акаунт до свого облікового запису за номером телефону. З цього моменту всі подальші взаємодії

(вибір університету, групи, отримання розкладу) здійснюються через чат-інтерфейс бота.

Обробка розкладу

На запит розкладу (на день або тиждень) бот звертається до репозиторію, отримує записи, сортує їх за датами та формує повідомлення. Результат надсилається у форматованому вигляді з такими елементами:

- дата та день тижня;
- час пари;
- назва предмета;
- тип заняття;
- викладач;
- аудиторія;
- додаткова інформація.

У разі відсутності занять бот надсилає повідомлення: “🔔 Пари у вказаний період відсутні.”

Автоматичне сповіщення

Крім ручного запиту, бот також використовується для автоматичних щоденних сповіщень. Щоранку (о 07:30) за допомогою Hangfire запускається задача, яка розсилає повідомлення всім користувачам, прив’язаним до груп, що мають пари в поточний день. Це дозволяє студентам не пропускати заняття та оперативно реагувати на зміни.

Особливості реалізації

- Вибір університету та групи відбувається через клавіатури Telegram (ReplyKeyboardMarkup);
- Користувачі можуть надсилати контакт (номер телефону) для реєстрації;
- Всі відповіді підтримують багатомовність завдяки адаптації повідомлень;

- Валідація введених даних (назви університетів/груп) реалізована через базу даних;
- Додано логіку очищення вибору групи при зміні університету, щоб уникнути конфліктів.

Переваги підходу

- Мобільність: бот працює на будь-якому смартфоні з Telegram;
- Простота: користувачу не потрібно встановлювати додатки чи заходити на сайт;
- Швидкість: відповіді на запити приходять миттєво;
- Інформативність: бот формує повні й структуровані повідомлення;
- Безпека: дані користувача передаються через захищені канали Telegram API.

Завдяки Telegram-боту студенти мають постійний доступ до розкладу й оперативно отримують важливу інформацію. Така форма взаємодії підвищує загальну ефективність освітнього процесу й покращує інформування в режимі реального часу.

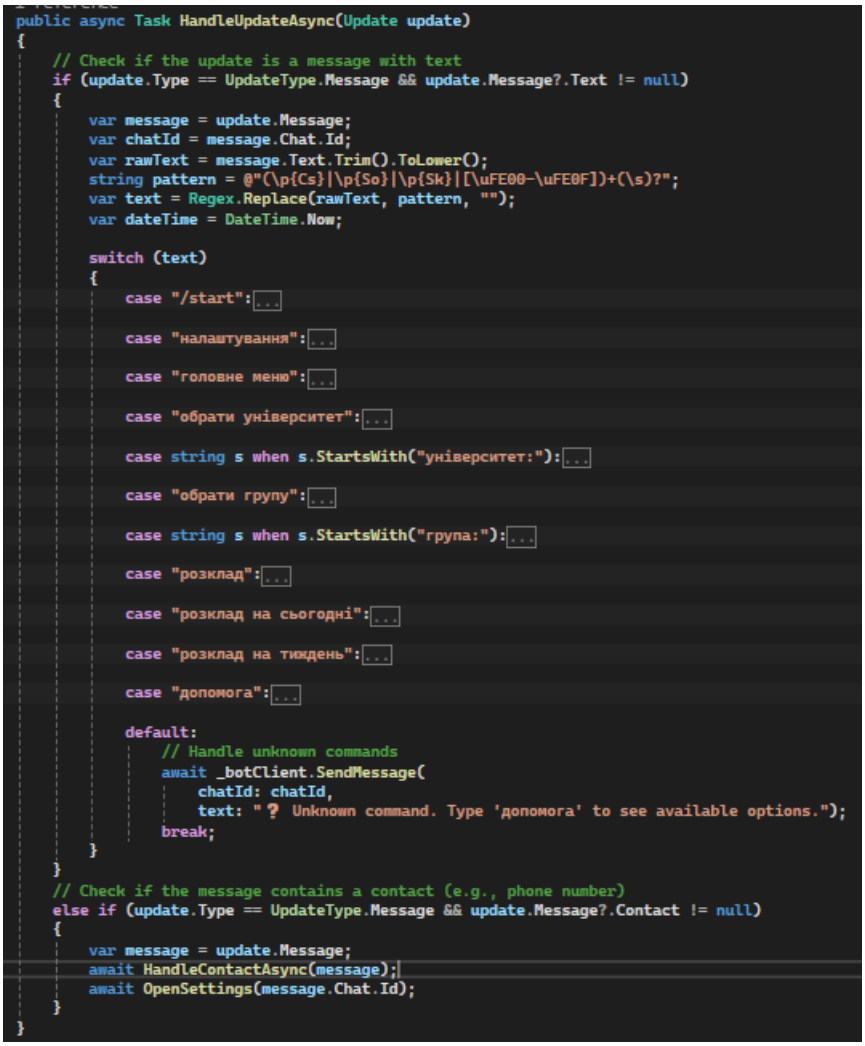


Рисунок 3.6 – Приклад обробки телеграм команд юзера.

4.6 UI: веб-інтерфейс викладача

Клієнтська частина системи розроблена з використанням фреймворку React у поєднанні з бібліотекою Material UI [6], що дозволяє створити інтуїтивно зрозумілий, адаптивний та візуально привабливий інтерфейс. Основний функціонал доступний для викладачів через особистий кабінет, який складається з вкладок для перегляду розкладу, створення та редагування пар, а також керування навчальними групами й предметами.

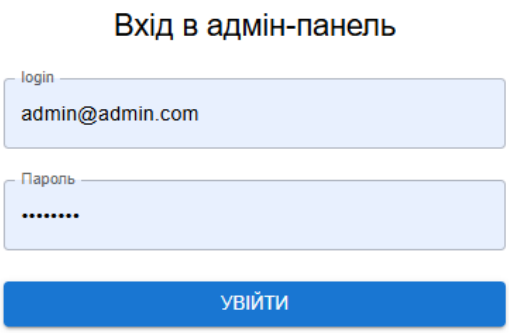


Рисунок 3.7 – Сторнка входу в адмін панель.

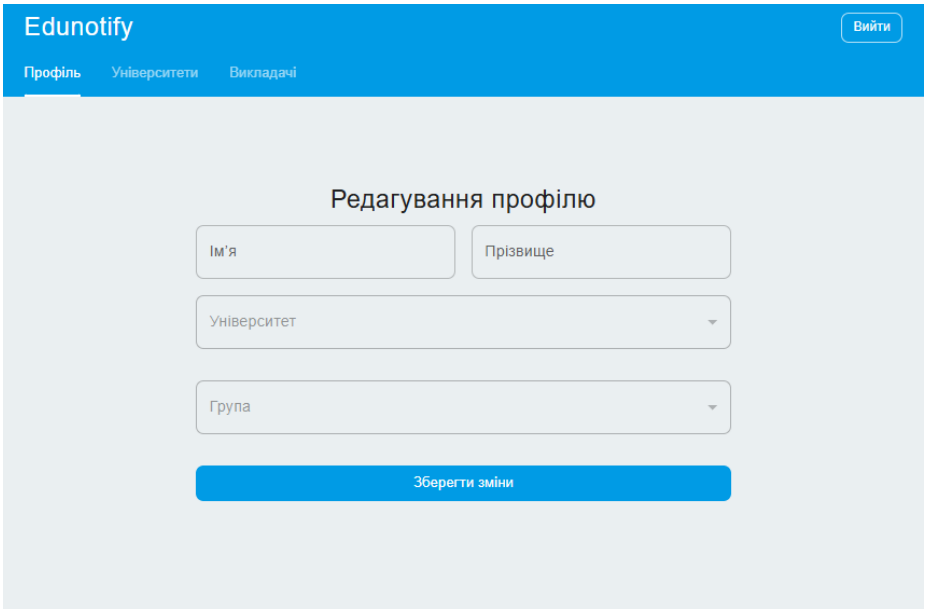


Рисунок 3.7 – Панель адміна.

Головна сторінка інтерфейсу викладача реалізована у вигляді табличного перегляду розкладу. Для зручності реалізовано фільтрацію та сортування за параметрами: дата, предмет, група, номер пари. Кожен запис має контекстне меню з можливістю редагування або видалення. Додавання нової пари виконується через модальне вікно з обов’язковим заповненням форми, де перевірка коректності даних відбувається одразу при введенні.

Для покращення UX застосовано компоненти Snackbar, Dialog, Select, Accordion та інші елементи з бібліотеки Material UI. Логіка керування станом

реалізована через React Context або Zustand, що дозволяє ефективно управляти глобальними та локальними станами компонентів.

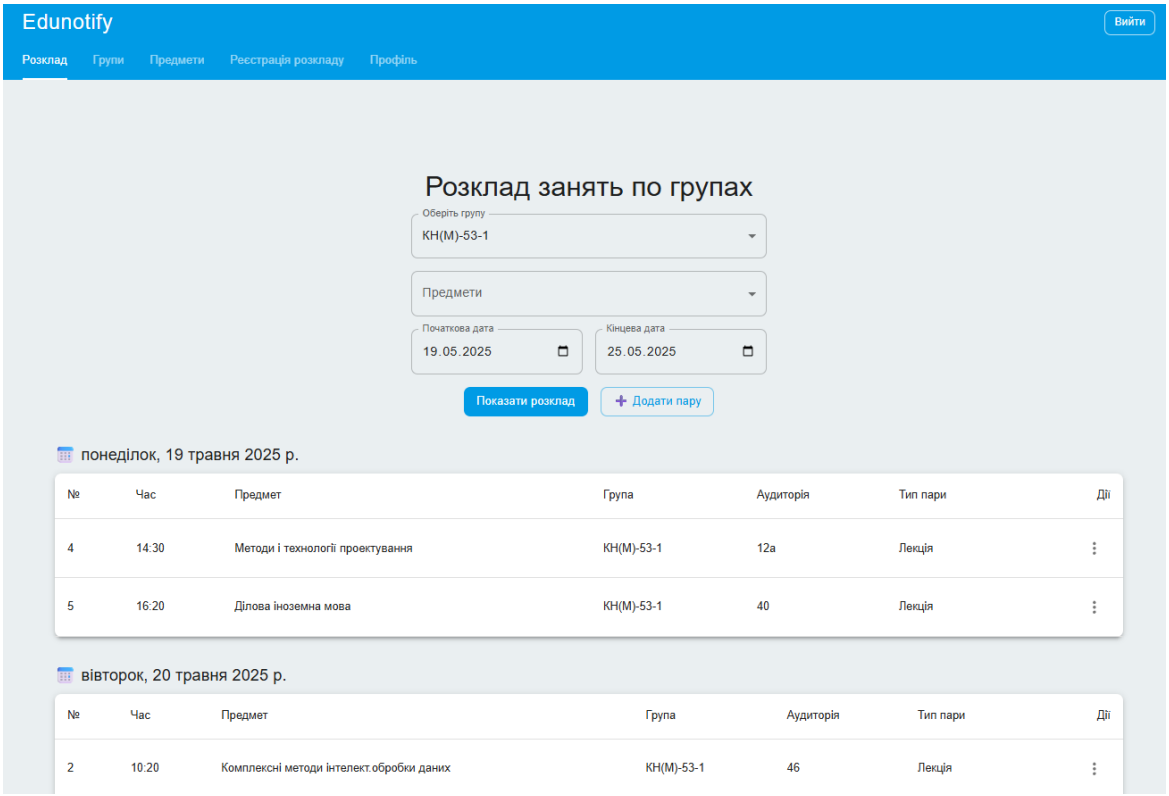


Рисунок 3.7 – Сторінка відображення розкладу.

Висновки до розділу

У цьому розділі було розглянуто інформаційне забезпечення системи інтелектуального управління розкладом. Було сформовано основні функціональні вимоги, визначено модель користувачів та логіку авторизації, описано структуру бази даних і реалізацію архітектурної підтримки даних на основі Onion-архітектури. Особливу увагу приділено інтеграції Telegram-бота як зручного інтерфейсу для студентів та реалізації автоматичних щоденних сповіщень. Завдяки структурованому підходу до інформаційної моделі система є гнучкою, масштабованою та зручною для різних ролей користувачів.

РОЗДІЛ 5.
РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї проєкту

Ідея стартап-проєкту полягає у створенні інтелектуальної системи управління навчальним розкладом, яка поєднує сучасні архітектурні підходи, автоматизовані алгоритми формування розкладу та інтерактивні інструменти комунікації з учасниками освітнього процесу. На відміну від традиційних систем, що зосереджені лише на статичному відображенні розкладу, запропонований проєкт забезпечує динамічне, гнучке та централізоване управління навчальними подіями, враховуючи особливості навчальних груп, викладачів, аудиторій та календарних циклів.

В основі проєкту лежить ідея єдиного інформаційного середовища, у межах якого адміністратори та викладачі можуть створювати й редагувати розклад в інтуїтивному інтерфейсі, а студенти отримують актуальну інформацію через вебінтерфейс або Telegram-бота. Система дозволяє працювати як зі статичними тижневими шаблонами, так і з динамічними подіями, які можуть змінюватися впродовж семестру. Особливу увагу приділено автоматизації: зміна одного елемента розкладу автоматично відображається в інших пов’язаних компонентах, що мінімізує помилки та дублювання даних.

До змісту ідеї також входить забезпечення розподіленості та масштабованості. Використання Onion-архітектури, реалізованої на серверній частині за допомогою .NET 9 [1], гарантує чітке розмежування бізнес-логіки, доступу до даних та інтерфейсів, що спрощує модифікацію та подальше розширення функціоналу. База даних MSSQL виступає центральним сховищем, де зберігаються сутності груп, дисциплін, викладачів, календарних тижнів, занять, а також історія змін.

Іншим ключовим компонентом ідеї є автоматизація інформування студентів, що досягається інтеграцією з Telegram Bot API. Система щоденно формує персоналізовані повідомлення із розкладом, використовуючи Hangfire для

виконання фонових задач. Таким чином, студенти не потребують самостійного пошуку актуальної інформації – система забезпечує її адресну доставку.

Запропонована система має широкі можливості застосування в закладах вищої освіти, профільних коледжах, мовних школах, освітніх центрах та корпоративних навчальних програмах. Її використання забезпечує оперативність оновлення розкладу, підвищує прозорість організації навчального процесу та значно зменшує адміністративне навантаження.

Основними вигодами для користувачів є:

- Адміністратори отримують зручний інструмент централізованого керування всіма групами та подіями.
- Викладачі можуть переглядати розклад, зміни та навантаження.
- Студенти отримують миттєві сповіщення про заняття, їх перенесення чи скасування.
- Освітня установа покращує комунікацію, мінімізує ризик дезорганізації, оптимізує використання аудиторій та ресурсів.

Важливою складовою ідеї є її відмінність від існуючих аналогів на ринку. Більшість систем не забезпечує комплексності: або пропонуються статичні електронні таблиці, або обмежені модулі розкладів у великих CRM/ERP-системах. Пропонований проєкт вирізняється:

- повною підтримкою динамічних змін розкладу;
- автоматичними сповіщеннями студентів;
- адаптацією під структуру конкретної установи;
- сучасною Onion-архітектурою;
- інтегрованим Telegram-ботом;
- можливістю швидкого масштабування.

Таким чином, зміст ідеї полягає у створенні сучасної, гнучкої та інтелектуальної екосистеми управління навчальним розкладом, яка поєднує

зручність, автоматизацію та інноваційні технології для підвищення ефективності освітнього процесу.

Запропонована інтелектуальна система управління навчальним розкладом має широкий спектр можливих напрямків використання та може бути адаптована до різних типів освітніх і навчальних процесів. Її універсальність зумовлена гнучкою архітектурою, модульністю, динамічним формуванням календарних подій та можливістю інтеграції з зовнішніми сервісами. Основною сферою застосування є заклади вищої та передвищої освіти — університети, академії, інститути, коледжі, а також приватні навчальні центри. При цьому система не обмежується традиційним академічним середовищем: вона може бути ефективно адаптована для корпоративного навчання, програм підвищення кваліфікації, курсів інтенсивного навчання, спортивних або творчих шкіл.

Можливі напрямки застосування

Першим і найбільш очевидним напрямком застосування є централізоване управління академічним розкладом у ЗВО. Враховуючи складність організації навчального процесу — наявність десятків спеціальностей, сотень груп, різних форматів занять (лекції, практичні, лабораторні), різноманіття аудиторій та індивідуальних графіків викладачів — автоматизована система стає ключовою умовою ефективного адміністрування. Система дозволяє створювати як тижневі шаблони, так і нестандартні події, враховувати зміни у складі груп, обмеження аудиторного фонду, заміни викладачів та інші фактори.

Другим напрямом є інформаційна підтримка студентів і викладачів у реальному часі. Інтеграція з Telegram Bot API забезпечує автоматичні розсилки актуального розкладу, повідомлення про зміни, скасування або перенесення занять. Такий підхід мінімізує ризики дезінформації та зменшує залежність від статичних джерел — друкованих листків або неактуальних публікацій на сайтах.

Третім напрямком застосування є організація подій та навчальних модулів у приватних навчальних центрах, де часто використовуються тижневі програми,

потоки слухачів та гнучкі графіки занять. Система дозволяє формувати розклад тренінгів, семінарів, консультацій, персональних занять, а також забезпечувати клієнтам персоналізовані сповіщення.

Четвертим напрямком є використання системи в корпоративному секторі. Компанії, що мають внутрішні програми навчання співробітників, курси з підвищення кваліфікації та технічні тренінги, можуть застосовувати систему як централізовану платформу планування, з можливістю інтеграції з HRM-системами та корпоративними месенджерами.

Окремим перспективним напрямком є застосування системи на платформах дистанційного навчання, де необхідно поєднувати різні часові пояси, формати вебінарів та змішаного навчання. Завдяки модульності та Onion-архітектурі система може бути інтегрована з LMS (Learning Management System), забезпечуючи синхронізацію подій.

Основні вигоди, які отримують користувачі

Інтелектуальна система управління розкладом створює значний практичний ефект для всіх груп користувачів — студентів, викладачів, адміністраторів та керівництва навчальних установ.

Студенти отримують доступ до актуального розкладу у зручному форматі, без необхідності вручну перевіряти зміни або переглядати складні таблиці. Автоматичні сповіщення через Telegram забезпечують миттєву комунікацію щодо перенесених або скасованих занять. Це особливо важливо в умовах динамічних змін — від заміни викладача до зміни аудиторії. Система зменшує невизначеність, полегшує планування навчального дня, а також сприяє підвищенню відвідуваності.

Викладачі отримують зручний доступ до власного навантаження, актуального тижневого графіка, змін у розкладі та інформації про аудиторії. Вони можуть швидше реагувати на зміни, планувати робочий час та уникати накладок між заняттями. Система також зменшує кількість адміністративних повідомлень, оскільки автоматизація сповіщень бере на себе комунікаційні функції.

Адміністратори факультетів та деканатів отримують найбільші вигоди: централізоване управління дозволяє швидко формувати розклад, дублювати шаблони, робити глобальні зміни, контролювати навантаження аудиторій та груп. Автоматизація значно зменшує рутинну роботу, яка традиційно виконується вручну — формування Excel-таблиць, перевірка конфліктів, поширення оновленої інформації.

Керівництво освітньої установи отримує інструмент, що підвищує організаційну культуру, оптимізує використання ресурсів та мінімізує помилки. Цілісна система спрощує аналітику, покращує управління навчальним процесом і забезпечує технологічний імідж закладу.

Відмінність від існуючих аналогів

На ринку освітніх технологій існує низка рішень, орієнтованих на створення та публікацію розкладів, проте більшість із них має обмежений функціонал, не забезпечує адаптивності або не дозволяє гнучко взаємодіяти зі студентами. Основні конкурентні переваги розробленого стартап-проєкту полягають у поєднанні сучасної архітектури, інтерактивності, високого рівня автоматизації та персоналізації.

Першою принциповою відмінністю є повна підтримка динамічних змін розкладу. Більшість традиційних систем працює з фіксованими розкладами, де зміни вносяться вручну у таблиці або вебсторінки. Запропонована система автоматично генерує актуальний розклад для кожної групи, дозволяє редагувати окремі заняття без зміни всього тижневого шаблону та забезпечує миттєве поширення оновлень.

Другою ключовою відмінністю є автоматизовані сповіщення через Telegram-бота. Аналоги рідко підтримують інтеграцію з популярними месенджерами або обмежуються електронною поштою. Telegram-бот забезпечує сучасний, швидкий та персоналізований спосіб комунікації.

Третьою перевагою є застосування Onion-архітектури, що забезпечує чітку структурованість, модульність та можливість масштабування. На відміну від монолітних рішень, система легко адаптується для нових типів навчальних закладів, дозволяє додавати модулі, інтегрувати сторонні сервіси, змінювати бізнес-логіку без порушення загальної структури.

Четвертим конкурентним фактором є сучасний адміністративний інтерфейс, побудований на React та Material UI. Більшість аналогів використовують застарілі або перевантажені інтерфейси, що ускладнюють роботу адміністративного персоналу. У цій системі акцент зроблено на зручність, інтуїтивність та мінімізацію кількості дій для виконання повсякденних завдань.

П’ятою суттєвою відмінністю є можливість гнучкого розширення функціоналу — від додавання різних типів подій до інтеграції з LMS, CRM або хмарними платформами. Завдяки використанню .NET 9, EF Core та MSSQL, система може ефективно працювати як у невеликих навчальних закладах, так і в установах із тисячами студентів.

5.2. Аналіз технологічних можливостей реалізації проєкту

Реалізація інтелектуальної системи управління навчальним розкладом потребує комплексного аналізу технологічної здійсненності, що включає оцінку доступних програмних інструментів, архітектурних підходів, методів обробки даних та можливостей інтеграції із зовнішніми сервісами. У межах даного проєкту розробка відбувається на основі сучасних технологій, які забезпечують масштабованість, надійність, продуктивність і простоту подальшого розвитку. Тому важливо визначити, які саме технології доступні, наскільки вони відповідають вимогам системи та які альтернативи можна розглядати у випадку подальшого розширення.

Першим елементом технологічної здійсненності є вибір серверної платформи. В якості основи використано .NET 9[1], що надає сучасні можливості

для створення високопродуктивних вебзастосунків. Оновлена версія платформи оптимізована для багатопоточних обчислень, масштабування, використання мінімальних API, реагування на високі навантаження та роботи у хмарних середовищах. Використання .NET 9 [1] забезпечує швидку обробку запитів, низьку затримку та підтримку великої кількості одночасних користувачів, що є критично важливим для системи, яка може використовуватися тисячами студентів і викладачів упродовж дня.

Одним із ключових компонентів реалізації є Onion-архітектура, яка забезпечує чітке розділення відповідальностей між бізнес-логікою, інфраструктурою та зовнішніми інтерфейсами. Такий підхід гарантує, що зміни у технологічних деталях (API, БД, клієнтська частина) не впливають на внутрішню логіку системи. Це дозволяє швидко впроваджувати нові модулі, змінювати джерела даних або інтегрувати додаткові сервіси. Onion-архітектура є не лише доступною, але й рекомендованою для систем, які мають розвиватися протягом тривалого часу.

Наступним важливим технологічним компонентом є система керування базами даних. Вибір MSSQL Server обумовлений його стабільністю, підтримкою складних зв’язків між сутностями, наявністю механізмів транзакційності, високою продуктивністю і можливістю оптимізації великих обсягів даних. Система розкладів містить сутності груп, дисциплін, аудиторій, викладачів, тижневих шаблонів, подій, тому використання реляційної моделі є цілком виправданим.

Проміжним інструментом між серверною логікою та базою даних виступає Entity Framework Core [2], який забезпечує об’єктно-реляційне відображення (ORM). EF Core дозволяє працювати з даними через моделі, автоматично виконуючи SQL-запити, створюючи міграції та оптимізуючи доступ до даних. Дана технологія є доступною, добре документованою і повністю підтримує MSSQL, що робить її оптимальним вибором.

Для автоматичного надсилання студентам щоденних повідомлень із розкладом використовується технологія Hangfire, яка дозволяє виконувати фонові задачі, планувати регулярні операції та керувати чергою задач. Hangfire підтримує роботу як у локальному режимі, так і у хмарному середовищі, забезпечуючи високу надійність виконання. Окрім сповіщень, Hangfire можна використовувати для автоматичної генерації тижневих шаблонів, перевірки конфліктів у розкладі та виконання періодичних сервісних операцій.

Для інтерактивної комунікації зі студентами впроваджено Telegram Bot API, який є безкоштовним, стабільним та широко застосовуваним. Telegram-бот формує персоналізовані повідомлення на основі розкладу, що зберігається в системі, підтримує запити користувачів, дозволяє швидко отримати інформацію без входу в браузер чи перегляду складних сайтів.

Фронтенд частина системи реалізована за допомогою React — сучасного інструменту для створення інтерактивних вебінтерфейсів. Ця технологія забезпечує високу продуктивність, компонентну структуру, можливість створення динамічних таблиць, форм, модальних вікон і visual-календарів. Бібліотека Material UI використовується для забезпечення стилізованих компонентів, які відповідають сучасним вимогам UX/UI.

Крім того, React легко інтегрується з REST API, що дозволяє адміністраторам і викладачам працювати із системою у реальному часі — переглядати розклад, оновлювати його, створювати нові події та керувати групами. Ці технології є відкритими, поширеними й повністю доступними для автора проєкту, що підтверджує технологічну здійсненність реалізації.

Для розгортання та ізоляції сервісів використовується Docker, який дозволяє запускати серверну частину, базу даних і супутні сервіси у вигляді контейнерів. Це забезпечує мобільність, швидке масштабування, контроль середовища виконання та можливість деплойменту в будь-яку хмарну інфраструктуру. Використання

Docker значно спрощує розгортання системи в навчальних закладах різних масштабів — від коледжів до університетів.

Проведений аналіз технологій демонструє, що всі програмні інструменти, необхідні для реалізації проєкту, вже існують, доступні, добре документовані та повністю сумісні між собою. Більше того, обрані технології використовуються у сучасних високонавантажених системах, що підтверджує їхню ефективність і стабільність.

Завдяки використанню відкритих бібліотек, хмарних рішень і гнучкої архітектури, система може бути розроблена, розгорнута та масштабована без необхідності створення нових технологій з нуля. Таким чином, проєкт має високу технологічну життєздатність, а обраний стек технологій дозволяє повністю реалізувати всі заплановані функції.

5.3 Аналіз ринкових можливостей

Аналіз ринкових можливостей є ключовим елементом успішного планування стартап-проєкту, що дозволяє визначити попит на розроблену систему, оцінити конкурентне середовище, окреслити ризики та перспективи впровадження. Для інтелектуальної системи управління навчальним розкладом особливу увагу приділено характеристикам ринку освітніх технологій (EdTech), його тенденціям, потребам цільової аудиторії та рівню готовності закладів освіти до цифрової трансформації.

Ринок EdTech активно зростає впродовж останніх років, що пов’язано з переходом навчальних закладів на цифрові платформи, поширенням дистанційного навчання та необхідністю підвищення ефективності адміністративних процесів. У світі та Україні спостерігається тенденція до автоматизації внутрішніх процесів у ЗВО та коледжах: облік студентів, електронні журнали, розклади, комунікація зі студентами, управління ресурсами.

Пандемія COVID-19 значно прискорила цей процес, продемонструвавши, що традиційні інструменти не здатні забезпечити оперативну адаптацію до змін. Університети активно інвестують у системи управління навчальним процесом, проте більшість таких систем орієнтована на LMS або загальні адміністративні модулі, а підтримка гнучких розкладів часто реалізована недостатньо.

Важливо, що ринок розкладів має стабільний та постійний попит, оскільки навчальні установи працюють у циклічному режимі та потребують системного планування незалежно від формату навчання (очно, дистанційно, змішано).

Попит формують три основні групи клієнтів:

1. Заклади вищої освіти

Вони потребують:

- централізованої системи розкладів,
- інструментів для швидкого реагування на зміни,
- можливості автоматизації сповіщень,
- засобів оптимізації навантаження аудиторій.

У багатьох університетах розклад досі формується вручну в Excel, що призводить до помилок, конфліктів і дублювань. Тому інтерес до автоматизації стабільно високий.

2. Коледжі, ліцеї та приватні школи

Такі установи часто працюють за складними ротаційними графіками, де групи мають різну тривалість і структуру тижня. Гнучка система з тижневими шаблонами і сповіщеннями вирішує їхні ключові проблеми.

3. Освітні центри та корпоративні навчальні програми

Вони потребують:

- планування тренінгів,
- розкладів потоків,
- можливості індивідуального сповіщення,
- онлайн-доступу до графіків.

У цій сфері особливо цінуються швидка адаптація системи та простота інтеграції.

Для освітніх установ критично важливими є:

- гнучкість — можливість швидко змінювати розклад;
- точність — уникнення аудиторних і викладацьких конфліктів;
- інтерактивність — сповіщення студентів без додаткових дій;
- зручність — інтуїтивний інтерфейс для адміністраторів;
- надійність — стабільна робота навіть у пікові години;
- масштабованість — підтримка великої кількості груп і потоків;
- модульність — можливість з часом додавати нові функції;
- інтеграція з Telegram, Viber, WhatsApp, Google Calendar, LMS або корпоративними системами.

Усі ці вимоги враховані в технологічній та функціональній моделі створеної системи.

На ринку існують кілька груп рішень:

1. Прості таблиці та документи (Excel, Google Sheets).

Переваги - доступність та простота.

Недоліки - немає автоматизації, немає перевірки конфліктів, ручні оновлення, не масштабується.

2. Великі університетські інформаційні системи типу Moodle-плагінів, університетських ERP).

Переваги — багатофункціональність.

Недоліки — складність, відсутність гнучкої роботи з динамічним розкладом, відсутність сповіщень, висока вартість.

3. Комерційні сервіси з вузьким функціоналом (електронні розклади в готових LMS).

Переваги — швидке розгортання.

Недоліки — не адаптуються до нестандартних графіків, не підтримують складні тижні, немає Telegram-інтеграції.

Інтелектуальна система управління розкладом має чітку ринкову нішу — автоматизовані та динамічні розклади з інтерактивною комунікацією.

Її головні конкурентні переваги:

- динамічна модель розкладу, побудована на тижневих шаблонах та окремих подіях;
- Onion-архітектура, яка дозволяє масштабувати систему на десятки та сотні груп;
- Telegram-бот зі щоденними сповіщеннями;
- інтеграція Hangfire для автоматизації завдань;
- сучасний React-інтерфейс для адміністраторів;
- швидке редагування розкладу без потреби оновлювати всю сітку;
- мобільність завдяки Docker;
- легка адаптація до різних типів навчальних процесів.

Таким чином, продукт може зайняти нішу між простими ручними рішеннями та громіздкими університетськими системами.

Фактори, що сприяють ринковому впровадженню

- Зростаюча цифровізація освіти.
- Потреба у швидкому реагуванні на зміни розкладу.
- Поширеність Telegram, WhatsApp серед студентів.
- Потреба у зменшенні адміністративного навантаження.
- Перехід установ від паперових розкладів до цифрових.
- Можливість інтегрувати продукт у наявну інфраструктуру.
- Низький поріг входу для користувачів.

Фактори, що можуть перешкоджати впровадженню

- бюджетні обмеження окремих закладів освіти;
- закритість або інертність адміністративних структур до нових технологій;

- конкуренція з існуючими університетськими платформами;
- потреба у технічному адмініструванні при першому впровадженні;
- низька цифрова грамотність частини персоналу.

Однак більшість цих ризиків можна знизити шляхом навчання, демоверсій та гнучкого ціноутворення.

Проведений аналіз свідчить, що попит на систему є стабільним і довгостроковим. Оскільки розклад — одна з основних характеристик освітнього процесу, інструменти для його автоматизації мають сталу актуальність. Завдяки модульності та можливості інтеграції, система може масштабуватися в такі напрямки:

- додавання push-сповіщень у мобільний застосунок;
- інтеграція з електронними журналами;
- автоматичне формування навчального навантаження;
- планування іспитів і модулів;
- інтеграція з LMS і CRM.

Ринок EdTech продовжує зростати, тому можливість комерціалізації проєкту оцінюється як висока.

5.4. Розроблення ринкової стратегії проєкту

Розроблення ринкової стратегії є ключовим етапом планування запуску стартап-проєкту, оскільки визначає підхід до охоплення цільових груп споживачів та способи виходу на ринок. На основі попередньо проведеного аналізу попиту, ринкового середовища та конкурентних пропозицій сформовано оптимальну стратегію ринкового позиціонування та впровадження інтелектуальної системи управління навчальним розкладом.

Завдяки модульності та універсальності системи, потенційними користувачами можуть бути кілька сегментів ринку:

1. Заклади вищої освіти (університети, академії, інститути).

Ця група має найбільший попит на автоматизовані розклади через значний обсяг даних, необхідність оперативних змін та логістичні труднощі.

2. Коледжі, ліцеї та приватні школи.

Для них важливою є гнучкість календарних планів, змінні тижневі розклади та простота адміністрування.

3. Освітні центри та школи додаткової освіти.

Потребують інструментів для планування тренінгів, потоків і коротких курсів.

4. Корпоративні програми навчання.

Цей сегмент цінує швидку адаптацію та інтеграції з внутрішніми системами.

Аналіз показує, що найбільшу ринкову вигоду забезпечує саме перший та другий сегменти, тому вони визначені як основні цільові групи.

5.4.1. Стратегія охоплення ринку

З урахуванням різноманітності потенційних користувачів та можливості адаптації системи, обрано стратегію диференційованого маркетингу. Це означає, що для кожного сегмента розробляється окрема програма ринкового просування та позиціонування продукту.

- Для закладів вищої освіти акцент робиться на автоматизації, масштабованості, можливості централізованого управління десятками груп та інтеграції з наявною інфраструктурою.
- Для коледжів та ліцеїв основними перевагами є простота використання, швидкість змін, підтримка ротаційних графіків та сповіщення студентів.
- Для приватних навчальних центрів підкреслюється гнучкість формування розкладів, легка інтеграція з Telegram, ефективність роботи з короткими курсами.
- Для корпоративних клієнтів головний акцент — можливість індивідуального налаштування та масштабування.

Стратегія диференційованого підходу дозволяє максимально точно відповідати потребам різних груп, не обмежуючи потенціал розвитку продукту.

5.4.2. Позиціонування продукту

Ринкове позиціонування системи базується на таких ключових характеристиках:

- інтелектуальність і динамічність,
- інтегровані інструменти комунікації (Telegram-бот),
- автоматизація процесів (Hangfire),
- сучасний інтерфейс на React,
- масштабованість та модульність,
- низький поріг входу для користувача.

Продукт позиціонується як оптимальне рішення для навчальних установ, які прагнуть перейти від статичних таблиць і застарілих систем до автоматизованого та інтерактивного управління.

5.4.3. Шляхи виходу на ринок

Для ефективного впровадження проєкту передбачено такі інструменти:

1. Пілотні впровадження у ЗВО та коледжах.

Дозволяють отримати зворотний зв’язок, покращити функціонал і створити рекомендації.

2. Демонстраційні версії.

Надають можливість тестування системи без витрат, формуючи довіру.

3. Співпраця з ІТ-відділами навчальних установ.

Забезпечує швидке розгортання та покращує адаптацію продукту.

4. Прямі продажі та консультаційні презентації.

Ефективні для великих навчальних закладів.

5. Просування через цифрові канали, включно з соціальними мережами, тематичними форуми, професійними спільнотами викладачів.

5.5. Маркетингова програма стартап-проєкту

Маркетингова програма визначає комплекс заходів, спрямованих на просування інтелектуальної системи управління навчальним розкладом на ринку освітніх технологій. Вона формує концепцію товару, цінову політику, підходи до збуту та просування, а також містить інструменти взаємодії з потенційними споживачами. Основою маркетингової програми є попередньо проведений аналіз ринку, визначення цільових груп, конкурентного середовища та ринкової стратегії охоплення.

5.5.1. Концепція продукту

Розроблений стартап позиціонується як інтелектуальна, динамічна та інтерактивна система управління навчальним розкладом, яка поєднує сучасні технології (Onion-архітектура, .NET 9, React, Telegram Bot API) та автоматизацію адміністративних процесів (планувальник Hangfire). Концепція продукту ґрунтується на таких характеристиках:

- автоматичне формування та оновлення розкладів;
- динамічне внесення змін із миттєвим поширенням користувачам;
- інтегровані сповіщення через Telegram-бота;
- сучасний інтерфейс для адміністраторів і викладачів;
- масштабованість для різних типів навчальних закладів;
- легка інтеграція з іншими сервісами.

Ці функціональні можливості формують цінність продукту як інноваційного інструменту для навчальних установ.

5.5.2. Цінова політика

Оскільки продукт орієнтований на різні типи установ — від невеликих коледжів до університетів, — доцільним є застосування гнучкої багаторівневої моделі ціноутворення:

1. Базовий план — для невеликих навчальних центрів.

Містить основні функції: формування розкладів, управління групами, Telegram-бот.

2. Стандартний план — для коледжів та ліцеїв.

Включає розширений функціонал, роботу з тижневими шаблонами, автоматичні фонові задачі.

3. Преміум-план — для університетів та великих установ.

Включає всі модулі, пріоритетну підтримку, індивідуальне налаштування та інтеграції.

4. Індивідуальна цінова пропозиція — для корпоративного сектору.

Вартість визначається на основі вимог до адаптації.

Такий підхід дозволяє зробити продукт доступним для широкого кола клієнтів, зберігаючи гнучкість монетизації.

5.5.3. Стратегія збуту

Продаж та впровадження продукту здійснюється через кілька основних каналів:

1. Прямі продажі навчальним закладам, з попередньою презентацією можливостей системи.

2. Пілотні впровадження, що дозволяють клієнтам безкоштовно протестувати продукт протягом певного періоду.

3. Партнерські програми з компаніями, що займаються автоматизацією освітніх процесів.

4. Розгортання у хмарній інфраструктурі, що знижує технічний поріг входу для клієнтів.

5. Дистрибуція через цифрові платформи, де клієнти можуть ознайомитися з демоверсією та документацією.

Багатоканальний підхід дозволяє швидко масштабувати продажі та виходити на установи різних типів.

5.5.4. Стратегія просування

Для популяризації продукту застосовується комплекс інструментів:

- Цифровий маркетинг: таргетована реклама, SEO, промосторінка продукту.
- Соціальні мережі: Telegram, Facebook, LinkedIn — для демонстрації переваг системи.
- Професійні освітні форуми та конференції, де представники викладацької та адміністративної спільноти можуть ознайомитися з функціоналом.
- Публікації в тематичних спільнотах, орієнтованих на EdTech та управління навчальним процесом.
- Відеопрезентації та інструкції, що демонструють роботу системи.
- Email-кампанії з інформацією про оновлення, нові модулі та можливості.

Просування має бути системним та постійним, оскільки навчальні установи ухвалюють рішення про впровадження переважно напередодні семестру або навчального року.

Розроблена маркетингова програма забезпечує системний підхід до просування проєкту, поєднуючи гнучку цінову політику, багатоканальну стратегію збуту, ефективні інструменти комунікації та сучасні методи цифрового маркетингу. Це створює сприятливі умови для успішного виходу продукту на ринок, формування стабільної клієнтської бази та подальшої комерціалізації системи в освітній сфері.

Висновок

У ході розробки виконано повний цикл дослідження, проєктування та реалізації інтелектуальної системи управління навчальним розкладом на основі Onion-архітектури. Запропоноване рішення забезпечує автоматизацію основних процесів взаємодії між адміністраторами, викладачами та студентами в межах організації навчального процесу.

З наукової точки зору, в роботі було здійснено формалізацію предметної області у вигляді математичних моделей, що дозволило точно визначити взаємозв’язки між сутностями та сформулювати обмеження у вигляді логічних предикатів. Розроблено алгоритми побудови розкладу з урахуванням циклічності, перевірки конфліктів, динамічного формування запитів, а також механізми автоматичного сповіщення користувачів.

З практичної точки зору, створено повнофункціональний програмний комплекс, який включає:

- фронтенд, реалізований засобами React і Material UI;
- серверну частину на основі .NET 9 з використанням Onion-архітектури, Hangfire, EF Core;
- інтегрований Telegram-бот для студентів;
- базу даних на основі MSSQL;
- автоматичні задачі розсилки через Hangfire.

Досягнуті якісні результати включають: адаптивність інтерфейсу, розмежування прав доступу, можливість роботи з великими обсягами даних без втрати продуктивності, та підтримку повторюваних шаблонів розкладу. Кількісні показники підтверджують стабільність системи, зокрема, ефективність обробки розкладу для великої кількості груп, швидке оновлення розкладу та надійне щоденне сповіщення через Telegram.

Завдання, поставлені у технічному завданні, були повністю виконані. Розроблена система задовольняє вимоги щодо функціональності, безпеки, масштабованості та зручності використання.

Результати роботи можуть бути використані для подальшої розробки комплексних освітніх платформ, розширення функціоналу розкладу, інтеграції з навчальними платформами (Google Calendar, WhatsApp) або застосування у реальних закладах вищої освіти.

Отже, дана робота має як теоретичну цінність, так і практичне значення, відкриваючи перспективи для подальших наукових досліджень і впроваджень у сфері цифровізації освіти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Інтернет-ресурси

1. Microsoft .NET Documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/ (дата звернення: 03.02.2025).
2. Entity Framework Core Documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 06.02.2025).
3. ASP.NET Core Documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/core/ (дата звернення: 10.03.2025).
4. Hangfire Documentation. URL: https://docs.hangfire.io/ (дата звернення: 10.03.2025).
5. React Documentation. Meta. URL: https://react.dev/ (дата звернення: 15.03.2025).
6. Material UI – React components for faster web development. URL: https://mui.com/ (дата звернення: 17.03.2025).
7. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2003. 533 p.
8. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley, 2004. 560 p.
9. Esposito D., Saltarello A. Microsoft .NET: Architecting Applications for the Enterprise. 2nd ed. Redmond : Microsoft Press, 2014. 368 p.
10. Code-maze. Onion Architecture in ASP.NET Core. URL: https://code-maze.com/onion-architecture-in-aspnetcore/ (дата звернення: 10.04.2025).
11. Medium. Building Scalable Applications with Onion Architecture in ASP.NET Core. URL https://semihtekin.medium.com/building-scalable-applications-with-onion-architecture-in-asp-net-core-8-cadb5c05fc2b (дата звернення: 11.04.2025).
12. Medium. From Concept to Code: Building Background Jobs in ASP.NET Core with Hangfire. URL: https://medium.com/@yayasaafan/from-concept-to-code-

building-background-jobs-in-asp-net-core-with-hangfire-d4a9d97b9004 (дата звернення: 20.05.2025).

13. Medium. Building a Telegram Bot with .NET, Chat GPT API. URL: https://medium.com/@sogue/building-a-telegram-bot-with-net-chat-gpt-api-d47dd312eb79 (дата звернення: 21.05.2025).

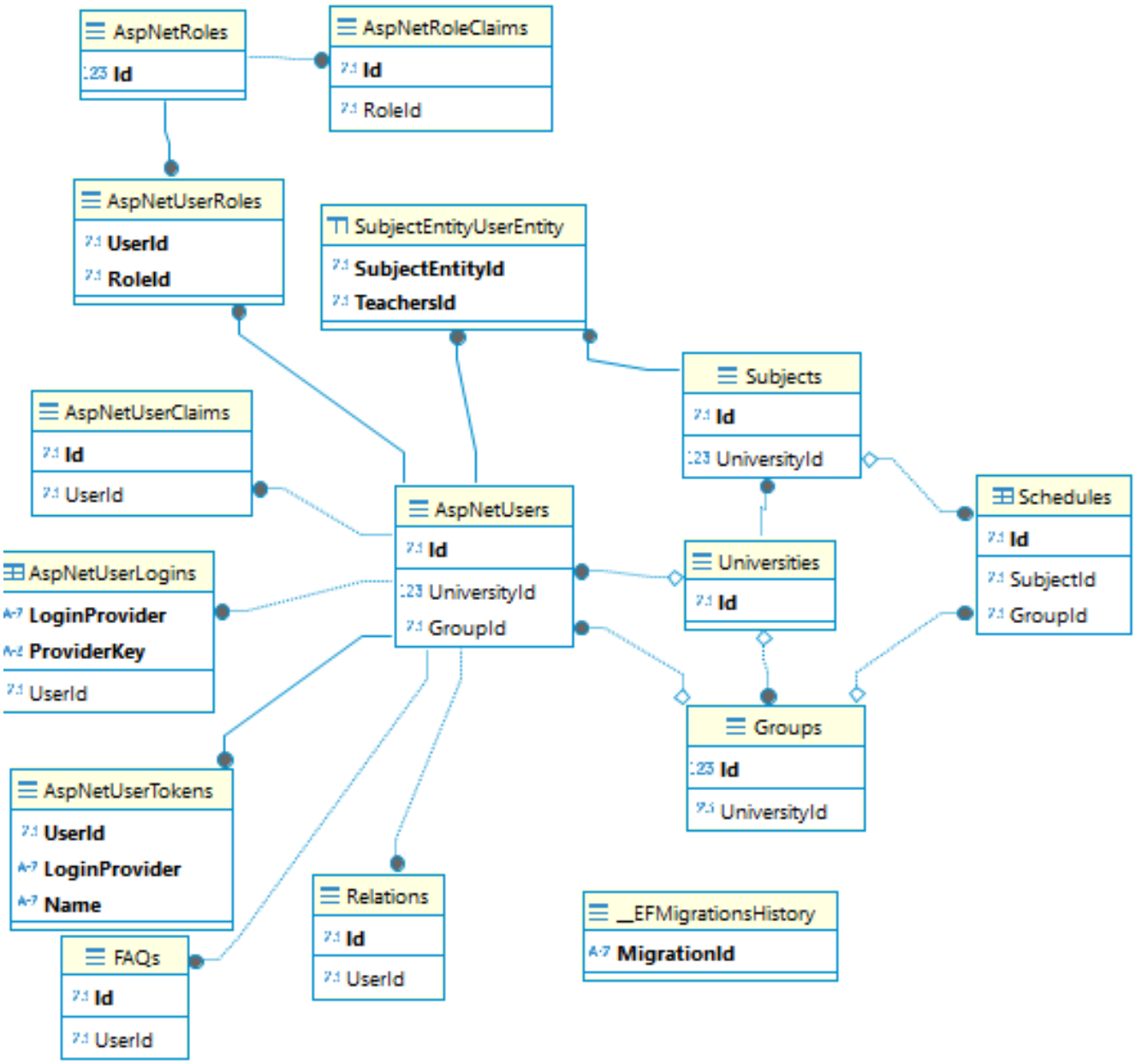
14. Docker, develop with containers URL: https://docs.docker.com/get-started/introduction/get-docker-desktop/

15. Visual Studio documentation. Learn how to use Visual Studio to develop applications, services, and tools URL: https://learn.microsoft.com/en-us/visualstudio/windows/?view=visualstudio

ДОДАТКИ

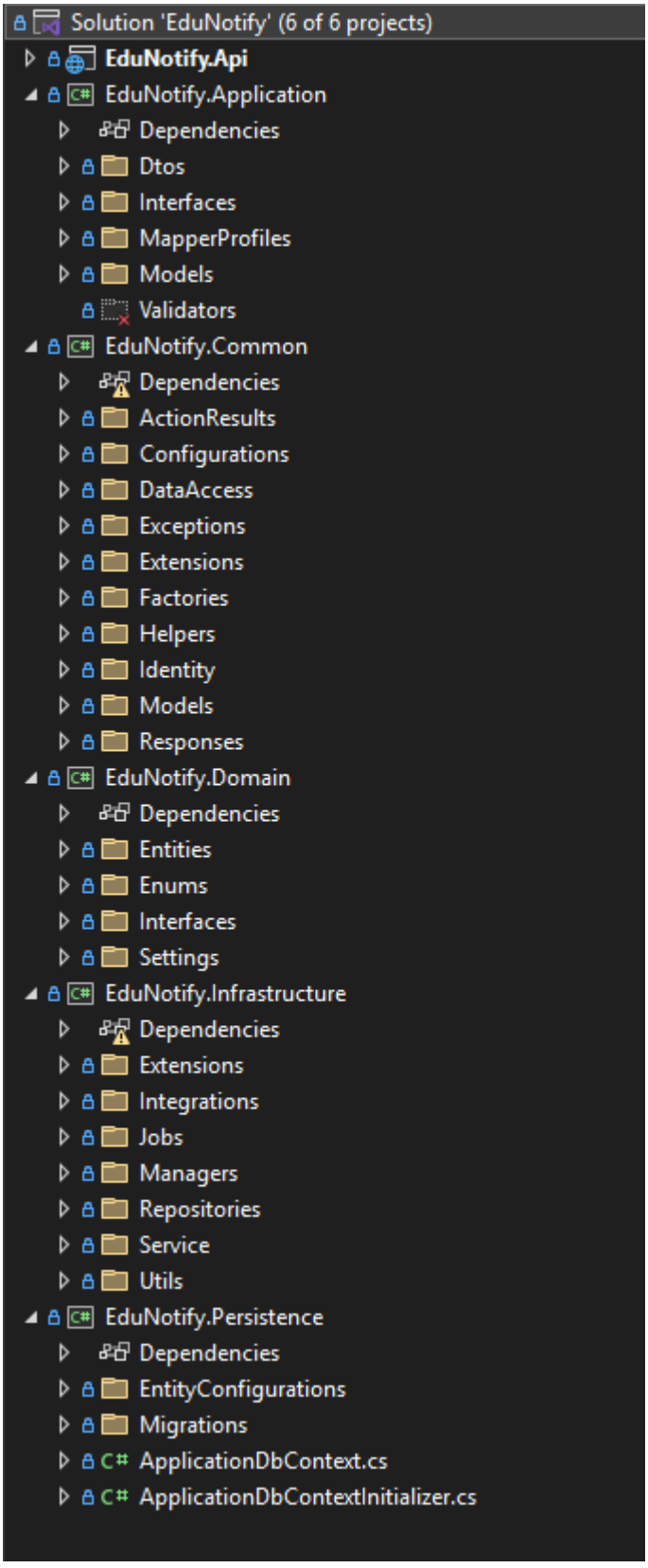
ДОДАТОК А

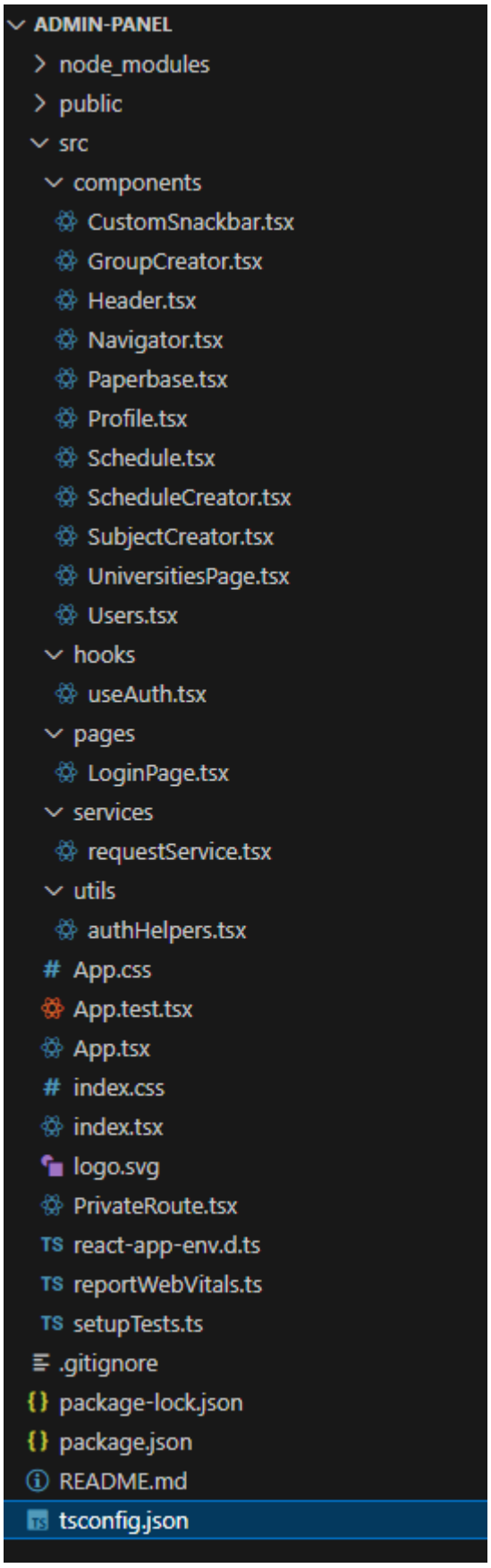
Структура бази даних системи



ДОДАТОК В

Архітектура програмної системи




ADMIN-PANEL
node_modules
public
src
components
CustomSnackbar.tsx
GroupCreator.tsx
Header.tsx
Navigator.tsx
Paperbase.tsx
Profile.tsx
Schedule.tsx
ScheduleCreator.tsx
SubjectCreator.tsx
UniversitiesPage.tsx
Users.tsx
hooks
useAuth.tsx
pages
LoginPage.tsx
services
requestService.tsx
utils
authHelpers.tsx
App.css
App.test.tsx
App.tsx
index.css
index.tsx
logo.svg
PrivateRoute.tsx
react-app-env.d.ts
reportWebVitals.ts
setupTests.ts
.gitignore
package-lock.json
package.json
README.md
tsconfig.json

ДОДАТОК С

Program.cs

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("LocalConnection");
var appSettings = builder.Configuration.GetSection("JwtTokenSettings").Get<TokenSettings>() ?? default!;
builder.Services.Configure<TokenSettings>(builder.Configuration.GetSection("JwtTokenSettings"));

builder.Services.AddDbContext<ApplicationDbContext>(options => options.UseSqlServer(connectionString));
builder.Services.AddProblemDetails();
builder.Services.AddRouting(options => options.LowercaseUrls = true);

builder.Services.AddControllers().AddJsonOptions(opt =>
{
   opt.JsonSerializerOptions.Converters.Add(new JsonStringEnumConverter());
});

builder.Services.Configure<SecurityStampValidatorOptions>(o => {
   o.ValidationInterval = TimeSpan.Zero;
});

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddServices();
builder.Services.AddRepositories();
builder.Services.AddIdentity();
builder.Services.AddAuthentication(appSettings);
builder.Services.AddSwagger();

var paths = Directory.GetFiles(AppDomain.CurrentDomain.BaseDirectory, "*Application.dll").ToList();
Assembly[] assemblies = paths.Select(path => Assembly.Load(AssemblyName.GetAssemblyName(path))).ToArray();
builder.Services.AddAutoMapper(assemblies);

builder.Services.AddScoped<ApplicationDbContextInitializer>();

builder.Services.AddOpenApi();

builder.Services.AddMvc().ConfigureApiBehaviorOptions(options =>
{
   options.InvalidModelStateResponseFactory = c =>
   {
      var errors = string.Join(' ', c.ModelState.Values.Where(v => v.Errors.Count > 0)
         .SelectMany(v => v.Errors)
         .Select(v => v.ErrorMessage));

      return new BadRequestObjectResult(ResponseFactory.InvalidData(errors));
   };
});

builder.Services.AddHangfire(connectionString!);
builder.Services.AddFluentValidationAutoValidation();

builder.Services.AddCors(options =>
{
   options.AddPolicy("AllowReactDev", policy =>
   {
```

```
        policy.WithOrigins("http://localhost:3000")
              .AllowAnyMethod()
              .AllowAnyHeader();
    });
});

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
    using var scope = app.Services.CreateScope();
    var initializer = scope.ServiceProvider.GetRequiredService<ApplicationDbContextInitializer>();
    await initializer.InitializeAsync();
    await initializer.SeedAsync();
}
var hangfireAdmin = builder.Configuration.GetSection("HangfireAdmin").Get<HangfireCredential>();
app.UseHangfireDashboard("/hangfire", new DashboardOptions
{
    Authorization = new[]
    {
        new HangfireCustomBasicAuthenticationFilter()
        {
            User = hangfireAdmin!.Name,
            Pass = hangfireAdmin.Password
        }
    },
    IgnoreAntiforgeryToken = true
});

app.UseCors("AllowReactDev");
app.UseHttpsRedirection();
app.UseMiddleware<ErrorHandlerMiddleware>();

app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();
app.MapHangfireDashboard();

app.Run();
```

ServiceCollectionExtensions.cs

```
public static class ServiceCollectionExtensions
 {
     public static void AddSwagger(this IServiceCollection services)
     {
         services.AddSwaggerGen(swagger =>
         {
             //This is to generate the Default UI of Swagger Documentation
             swagger.SwaggerDoc("v1", new OpenApiInfo
             {
                 Version = "v1",
                 Title = "EduNotify API",
                 Description = ".NET 9 Web API"
             });
             // To Enable authorization using Swagger (JWT)
             swagger.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme()
```

```
            {
                Name = "Authorization",
                Type = SecuritySchemeType.ApiKey,
                Scheme = "Bearer",
                BearerFormat = "JWT",
                In = ParameterLocation.Header,
                Description = "JWT Authorization header using the Bearer scheme. \r\n\r\n Enter 'Bearer' [space] and then your token in the text input below.\r\n\r\nExample: \"Bearer 12345abcdef\"",
            });
            swagger.AddSecurityRequirement(new OpenApiSecurityRequirement
            {
                {
                    new OpenApiSecurityScheme
                    {
                        Reference = new OpenApiReference
                        {
                            Type = ReferenceType.SecurityScheme,
                            Id = "Bearer"
                        }
                    },
                    new string[] {}
                }
            });
        });
    }

    public static void AddIdentity(this IServiceCollection services)
    {
        services
            .AddIdentity<UserEntity, RoleEntity>(options =>
            {
                options.SignIn.RequireConfirmedAccount = false;
                options.User.RequireUniqueEmail = false;
                options.Password.RequireDigit = false;
                options.Password.RequiredLength = 6;
                options.Password.RequireNonAlphanumeric = false;
                options.Password.RequireUppercase = false;
            })
            .AddRoles<RoleEntity>()
            .AddSignInManager()
            .AddEntityFrameworkStores<ApplicationDbContext>()
            .AddTokenProvider<DataProtectorTokenProvider<UserEntity>>("REFRESHTOKENPROVIDER");
    }

    public static void AddAuthentication(this IServiceCollection services, TokenSettings appSettings)
    {
        services.Configure<DataProtectionTokenProviderOptions>(options =>
        {
            options.TokenLifespan = TimeSpan.FromSeconds(appSettings.TokenExpireSeconds);
        });

        services.AddAuthentication(options => {
            options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
            options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
            options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
        })
        .AddJwtBearer(options =>
        {
            options.IncludeErrorDetails = true;
```

```
            options.TokenValidationParameters = new TokenValidationParameters()
            {
                ClockSkew = TimeSpan.Zero,
                ValidateIssuer = true,
                ValidateAudience = true,
                ValidateLifetime = true,
                ValidateIssuerSigningKey = true,
                ValidIssuer = appSettings.Issuer,
                ValidAudience = appSettings.Audience,
                IssuerSigningKey = new SymmetricSecurityKey(
                    Encoding.UTF8.GetBytes(appSettings.SecurityKey)
                ),
            };
        });
    }
}
```

AuthController.cs

```
[Route("[controller]")]
public class AuthController(
    AuthService authService)
    : BaseController
{
    [HttpPost("login")]
    public async Task<IActionResult> Login([FromBody] AuthRequest request)
    {
        return MapResponse(await authService.LoginAsync(request));
    }

    [HttpPost("refresh")]
    public async Task<IActionResult> RefreshToken([FromBody] UserRefreshTokenRequest request)
    {
        return MapResponse(await authService.RefreshTokenAsync(request));
    }

    [HttpPost("logout")]
    [Authorize]
    public async Task<IActionResult> Logout()
    {
        return MapResponse(await authService.LogoutAsync(User));
    }
}
```

GroupController.cs

```
[Route("[controller]")]
public class GroupController(GroupService groupService) : BaseController
{
    [HttpPost]
    [Authorize(Roles = IdentityRoles.Teacher)]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    public async Task<IActionResult> CreateGroup([FromBody] CreateGroupRequest request)
    {
        var result = await groupService.CreateGroupAsync(request.Name);
        return MapResponse(result);
    }
```

```
    [HttpGet("{universityId}")]
    [Authorize(Roles = IdentityRoles.Teacher)]
    [ProducesResponseType(StatusCodes.Status200OK)]
    public async Task<IActionResult> GetAll([FromRoute] long universityId)
    {
        var result = await groupService.GetAllGroupsByUniversityIdAsync(universityId);
        return MapResponse(result);
    }

    [HttpGet]
    [Authorize(Roles = IdentityRoles.Teacher)]
    [ProducesResponseType(StatusCodes.Status200OK)]
    public async Task<IActionResult> GetUserGroups()
    {
        var result = await groupService.GetUserGroupsAsync();
        return MapResponse(result);
    }


    [HttpDelete("{id}")]
    [Authorize(Roles = IdentityRoles.Teacher)]
    [ProducesResponseType(StatusCodes.Status200OK)]
    public async Task<IActionResult> DeleteById([FromRoute] long id)
    {
        var result = await groupService.DeleteGroupAsync(id);
        return MapResponse(result);
    }
}
```

ScheduleController.cs

```
[Route("[controller]")]
 [Authorize(Roles = IdentityRoles.AdminOrTeacher)]
 public class ScheduleController(ScheduleService scheduleService) : BaseController
 {
    [HttpPost("list")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    public async Task<IActionResult> GetByDateRange([FromBody] ScheduleFilterRequest request)
    {
        var result = await scheduleService.GetSchedulesAsync(request.GroupId, request.From, request.To,
request.SubjectsIds);
        return MapResponse(result);
    }

    [HttpPost]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    public async Task<IActionResult> CreateSchedules([FromBody] CreateScheduleEntriesRequest request)
    {
        var result = await scheduleService.CreateSchedulesAsync(request);
        return MapResponse(result);
    }

    [HttpPost("pair")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    public async Task<IActionResult> CreateSchedule([FromBody] CreateScheduleRequest request)
    {
        var result = await scheduleService.CreateScheduleAsync(request);
        return MapResponse(result);
```

```
    }

    [HttpPut("{id}")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    public async Task<IActionResult> UpdateSchedule([FromRoute] long id, [FromBody] UpdateScheduleRequest request)
    {
        var result = await scheduleService.UpdateScheduleAsync(id, request);
        return MapResponse(result);
    }

    [HttpPut("{id}/status/{status}")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    public async Task<IActionResult> UpdateScheduleById([FromRoute] long id, [FromRoute] ScheduleStatus status)
    {
        var result = await scheduleService.UpdateScheduleStatusAsync(id, status);
        return MapResponse(result);
    }

    [HttpDelete("group/{groupId}")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    public async Task<IActionResult> DeleteSchedule([FromRoute] long groupId, [FromRoute] long subjectId)
    {
        var result = await scheduleService.DeleteByGroupIdAsync(groupId);
        return MapResponse(result);
    }

    [HttpDelete("{id}")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    public async Task<IActionResult> DeleteSchedule([FromRoute] long id)
    {
        var result = await scheduleService.DeleteAsync(id);
        return MapResponse(result);
    }
}
```

UniversityController.cs

```
[Route("[controller]")]
 [Authorize]
 public class UniversityController(
    IMapper mapper,
    UniversityService universityService)
    : BaseController
{
    [HttpPost]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    public async Task<IActionResult> RegisterAsync([FromBody] CreateUniversityRequest request)
    {
        var result = await universityService.CreateAsync(mapper.Map<UniversityDto>(request));
        return MapResponse(result);
    }

    [HttpGet]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    public async Task<IActionResult> GetAll()
    {
        var result = await universityService.GetAllAsync();
        return MapResponse(result);
```

```
    }

    [HttpDelete("{id}")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    public async Task<IActionResult> DeleteById([FromRoute] long id)
    {
        var result = await universityService.DeleteAsync(id);
        return MapResponse(result);
    }
}
```

UsersController.cs

```
[Route("[controller]")]
public class UsersController(
    UserService userService, ICurrentUser currentUser)
    : BaseController
{
    [Authorize(Roles = IdentityRoles.Admin)]
    [HttpGet("{id}")]
    [ProducesResponseType(StatusCodes.Status200OK)]
    public async Task<IActionResult> GetById( [FromRoute] long id)
    {
        return MapResponse(await userService.GetByIdAsync(id));
    }

    [Authorize(Roles = IdentityRoles.Admin)]
    [HttpGet]
    [ProducesResponseType(StatusCodes.Status200OK)]
    public async Task<IActionResult> GetAll()
    {
        return MapResponse(await userService.GetAllTeachers());
    }

    [Authorize]
    [HttpGet("profile")]
    [ProducesResponseType(StatusCodes.Status200OK)]
    public async Task<IActionResult> GetProfile()
    {
        var id = currentUser.IdentityId;
        return MapResponse(await userService.GetByIdAsync(id));
    }

    [Authorize(Roles = IdentityRoles.Admin)]
    [HttpPut]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    public async Task<IActionResult> UpdateUser(UpdateUserRequest request)
    {
        var id = currentUser.IdentityId;
        return MapResponse(await userService.UpdateAsync(id, request));
    }

    [Authorize(Roles = IdentityRoles.Admin)]
    [HttpPost("register/teacher")]
    [ProducesResponseType(StatusCodes.Status204NoContent)]
    public async Task<IActionResult> RegisterTeacher(RegistrationRequest request)
    {
        return MapResponse(await userService.RegisterTeacherAsync(request));
```

```
    }

    [Authorize(Roles = IdentityRoles.Admin)]
    [HttpDelete("{id}")]
    [ProducesResponseType(StatusCodes.Status200OK)]
    public async Task<IActionResult> DeleteById([FromRoute] long id)
    {
        return MapResponse(await userService.DeleteAsync(id));
    }
}
```

AuthService.cs

```
public class AuthService(
    TokenService tokenService,
    SignInManager<UserEntity> signInManager,
    IOptions<TokenSettings> settings,
    UserManager userManager)
{
    /// <summary>
    /// Attempts to log in a user using either email or phone number and password.
    /// </summary>
    /// <param name="request">The login request containing login identifier and password.</param>
    /// <returns>A response with authentication tokens or an error message.</returns>
    public async Task<IResponse<AuthResponse>> LoginAsync(AuthRequest request)
    {
        UserEntity? user;

        if (new EmailAddressAttribute().IsValid(request.Login))
        {
            user = await userManager.FindByEmailAsync(request.Login!);
        }
        else if (IsPhoneNumber(request.Login!))
        {
            user = await userManager.FindByPhoneNumberAsync(request.Login!);
        }
        else
        {
            return ResponseFactory.InvalidData<AuthResponse>("❌ Невірний формат логіну або паролю 🤦‍♂️");
        }

        if (user == null)
        {
            return ResponseFactory.NotFound<AuthResponse>("☹ Користувача не знайдено! Спробуйте ще раз...");
        }

        var result = await signInManager.CheckPasswordSignInAsync(user, request.Password!, true);
        if (!result.Succeeded)
        {
            return ResponseFactory.InvalidData<AuthResponse>("🔒 Неправильні дані для входу! 😖");
        }

        return ResponseFactory.Success(await tokenService.CreateToken(user));
    }

    /// <summary>
    /// Refreshes the access token using a valid refresh token.
```

```
    /// </summary>
    /// <param name="request">The refresh token request.</param>
    /// <returns>A response with a new authentication token or an error message.</returns>
    public async Task<IResponse<AuthResponse>> RefreshTokenAsync(UserRefreshTokenRequest request)
    {
        var principal = TokenUtil.GetPrincipalFromExpiredToken(settings.Value, request.AccessToken);

        if (principal == null || principal.FindFirst(ClaimTypes.Email)?.Value == null)
        {
            return ResponseFactory.InvalidData<AuthResponse>("⊘ Невірний токен доступу. Оновлення заборонено! 🔐");
        }

        var user = await userManager.FindByEmailAsync(principal.FindFirst(ClaimTypes.Email)?.Value ?? "");
        if (user == null)
        {
            return ResponseFactory.InvalidData<AuthResponse>("🕵️‍♂️ Користувача не знайдено. Перевірте токен! 🚫");
        }

        if (!await userManager.VerifyUserTokenAsync(user, "REFRESHTOKENPROVIDER", "RefreshToken",
request.RefreshToken))
        {
            return ResponseFactory.InvalidData<AuthResponse>("🔄 Невірний refresh токен. Спробуйте перезайти ☹️");
        }

        var accessToken = await tokenService.CreateToken(user);
        return ResponseFactory.Success(accessToken);
    }

    /// <summary>
    /// Logs out the current authenticated user.
    /// </summary>
    /// <param name="user">The current user principal.</param>
    /// <returns>A response indicating success or failure.</returns>
    public async Task<IResponse> LogoutAsync(ClaimsPrincipal user)
    {
        if (user.Identity?.IsAuthenticated ?? false)
        {
            await signInManager.SignOutAsync();

            var email = user.Claims.First(x => x.Type == ClaimTypes.Email).Value;
            var appUser = await userManager.FindByEmailAsync(email);
            if (appUser == null)
            {
                return ResponseFactory.NotFound<AuthResponse>("🚫 Користувача не знайдено під час виходу ❌");
            }

            return ResponseFactory.Success(userManager.UpdateSecurityStampAsync(appUser));
        }
        else
        {
            return ResponseFactory.InvalidData<AuthResponse>("⛔ Ви не авторизовані. Вихід недоступний! 🔓");
        }
    }

    /// <summary>
    /// Validates whether a given string matches the Ukrainian phone number format.
    /// </summary>
    /// <param name="input">The input string to validate.</param>
```

```
    /// <returns>True if it is a valid phone number; otherwise, false.</returns>
    private bool IsPhoneNumber(string input)
    {
        return Regex.IsMatch(input, @"^\+380\d{9}$");
    }
}
```

ScheduleService.cs

```
public class ScheduleService(
    IScheduleRepository scheduleRepository,
    IGroupRepository groupRepository,
    ISubjectRepository subjectRepository,
    IUniversityRepository universityRepository,
    ICurrentUser currentUser,
    UserManager userManager,
    TelegramBotClient botClient)
{
    /// <summary>
    /// Creates a schedule for a group based on a weekly template.
    /// </summary>
    public async Task<IResponse> CreateSchedulesAsync(CreateScheduleEntriesRequest request)
    {
        var user = await userManager.FindByIdAsync(currentUser.IdentityId.ToString());
        if (user is null || user.UniversityId == null)
        {
            return ResponseFactory.InvalidData<List<ScheduleResponseDto>>("😳💸 Неможливо знайти користувача або університет!");
        }

        if (await scheduleRepository.ExistAsync(x => x.GroupId == request.GroupId))
        {
            return ResponseFactory.InvalidData("📅 Розклад для цієї групи вже існує!");
        }

        if (!await universityRepository.ExistAsync(x => x.Id == user.UniversityId))
        {
            return ResponseFactory.NotFound("🏚️ Університет не знайдено!");
        }

        if (!await groupRepository.ExistAsync(x => x.Id == request.GroupId))
        {
            return ResponseFactory.NotFound("👨‍👩‍👧‍👦 Групу не знайдено!");
        }

        var subjectIds = await subjectRepository.GetSubjectsIdsAsync(user.UniversityId.Value);
        var totalDays = (request.EndDay.DayNumber - request.StartDay.DayNumber) + 1;
        var numberOfWeeks = request.WeeklySchedules.Count;

        var entries = new List<SchedulesEntity>();

        for (int i = 0; i < totalDays; i++)
        {
            var currentDate = request.StartDay.AddDays(i);
            var weekIndex = (i / 7) % numberOfWeeks;
            var dailySchedule = request.WeeklySchedules[weekIndex].DayliSchedules
                .FirstOrDefault(d => d.Day == currentDate.DayOfWeek);
```

```
            if (dailySchedule?.Subjects != null)
            {
                foreach (var subject in dailySchedule.Subjects)
                {
                    if (!subjectIds.Contains(subject.SubjectId))
                    {
                        return ResponseFactory.InvalidData($"🏫 Предмет з ID {subject.SubjectId} не знайдено!");
                    }

                    var pairInfo = ClassTimes.Times.First(x => x.Key == subject.PairNumber);

                    entries.Add(new SchedulesEntity
                    {
                        GroupId = request.GroupId,
                        Date = currentDate,
                        PairNumber = pairInfo.Key,
                        PairDate = pairInfo.Value,
                        SubjectId = subject.SubjectId,
                        Room = subject.Room,
                        Type = subject.Type,
                    });
                }
            }
        }

        await scheduleRepository.InsertRangeAsync(entries);
        await scheduleRepository.UnitOfWork.SaveChangesAsync();

        return ResponseFactory.Success();
    }

    /// <summary>
    /// Creates a single schedule pair (lesson) for a group.
    /// </summary>
    public async Task<IResponse> CreateScheduleAsync(CreateScheduleRequest request)
    {
        if (await scheduleRepository.ExistAsync(
            x => x.Date == request.Date && x.PairNumber == request.PairNumber && x.GroupId == request.GroupId &&
x.Status == ScheduleStatus.Active))
        {
            return ResponseFactory.InvalidData("⛔ Пара в цей час вже існує!");
        }

        var schedule = new SchedulesEntity
        {
            GroupId = request.GroupId,
            SubjectId = request.SubjectId,
            Date = request.Date,
            PairNumber = request.PairNumber,
            PairDate = ClassTimes.Times.First(x => x.Key == request.PairNumber).Value,
            Info = request.info,
            Room = request.Room,
            Type = request.Type,
        };

        await scheduleRepository.InsertAsync(schedule);
        await scheduleRepository.UnitOfWork.SaveChangesAsync();
```

```
        var now = DateTime.Now;
        DateOnly dateOnly = DateOnly.FromDateTime(now);

        var pairTime = ClassTimes.Times[request.PairNumber];
        var currentTime = now.TimeOfDay;

        if (schedule.GroupId != null)
        {
            SendUserUpdateScheduleStatus(schedule, schedule.GroupId!.Value, ChangeType.Added);
        }

        return ResponseFactory.Success();
    }

    /// <summary>
    /// Updates schedule pair's information and status.
    /// </summary>
    public async Task<IResponse> UpdateScheduleAsync(long id, UpdateScheduleRequest request)
    {
        var pair = await scheduleRepository.FirstOrDefaultAsync(x => x.Id == id && x.Status == ScheduleStatus.Active);
        if (pair == null)
        {
            return ResponseFactory.InvalidData("✕ Пару не знайдено!");
        }

        pair.Status = request.Status;
        pair.Info = request.info;
        pair.Room = request.Room;
        pair.Type = request.Type;

        scheduleRepository.Update(pair);
        await scheduleRepository.UnitOfWork.SaveChangesAsync();

        return ResponseFactory.Success();
    }

    /// <summary>
    /// Updates only the status of a schedule pair.
    /// </summary>
    public async Task<IResponse> UpdateScheduleStatusAsync(long id, ScheduleStatus status)
    {
        var schedule = await scheduleRepository.FirstOrDefaultAsync(x => x.Id == id && x.Status ==
ScheduleStatus.Active);
        if (schedule == null)
        {
            return ResponseFactory.InvalidData("❗ Пару не знайдено або вона неактивна");
        }

        schedule.Status = status;

        scheduleRepository.Update(schedule);
        await scheduleRepository.UnitOfWork.SaveChangesAsync();

        if (schedule.GroupId != null && status == ScheduleStatus.Canceled)
        {
            SendUserUpdateScheduleStatus(schedule, schedule.GroupId!.Value, ChangeType.Canceled);
        }
```

```
        return ResponseFactory.Success();
    }

    /// <summary>
    /// Retrieves the schedule for a group between two dates.
    /// </summary>
    public async Task<IResponse<List<ScheduleResponseDto>>> GetSchedulesAsync(long? groupId, DateOnly from,
DateOnly to, List<long> subjectsIds)
    {
        if (to < from)
            return ResponseFactory.InvalidData<List<ScheduleResponseDto>>("🗓 Некоректний діапазон дат!");

        var user = await userManager.FindByIdAsync(currentUser.IdentityId.ToString());
        if (user is null || user.UniversityId == null)
        {
            return ResponseFactory.InvalidData<List<ScheduleResponseDto>>("😕 Дані користувача відсутні!");
        }

        var scheduleEntities = await scheduleRepository.GetByDateRangeAsync(user.UniversityId!.Value, groupId, from, to,
subjectsIds);

        var grouped = scheduleEntities
            .GroupBy(x => x.Date)
            .Select(g => new ScheduleResponseDto
            {
                Date = g.Key,
                Entries = g.OrderBy(e => e.PairNumber).Select(e => new ScheduleEntryDto
                {
                    Id = e.Id,
                    PairNumber = e.PairNumber,
                    PairTime = e.PairDate,
                    SubjectId = e.SubjectId!.Value,
                    SubjectName = e.Subject?.Name ?? "Unknown",
                    Room = e.Room,
                    GroupId = e.GroupId!.Value,
                    Type = e.Type,
                    Info = e.Info,
                    GroupName = e.Group!.Name,
                    Teacher = e.Subject?.Teachers?
                        .Select(x =>
                            $"{(x.LastName ?? "").Trim()} {(x.FirstName ?? "").Trim()}".Trim()
                        )
                        .Where(name => !string.IsNullOrWhiteSpace(name))
                        .ToList() ?? new List<string>()
                }).ToList()
            })
            .OrderBy(dto => dto.Date)
            .ToList();

        return ResponseFactory.Success(grouped);
    }

    /// <summary>
    /// Deletes a schedule entry by its ID.
    /// </summary>
    public async Task<IResponse> DeleteAsync(long id)
    {
        var schedule = await scheduleRepository.FirstOrDefaultAsync(x => x.Id == id);
```

```
        if (schedule == null)
        {
            return ResponseFactory.NotFound("🗑 Пару не знайдено для видалення!");
        }

        scheduleRepository.Remove(schedule);
        await scheduleRepository.UnitOfWork.SaveChangesAsync();

        if (schedule.GroupId != null)
        {
            SendUserUpdateScheduleStatus(schedule, schedule.GroupId!.Value, ChangeType.Removed);
        }

        return ResponseFactory.Success();
    }

    /// <summary>
    /// Deletes all schedules associated with a specific group.
    /// </summary>
    public async Task<IResponse> DeleteByGroupIdAsync(long groupId)
    {
        var schedules = await scheduleRepository.GetAsync(x => x.GroupId == groupId);

        if (schedules == null || !schedules.Any())
        {
            return ResponseFactory.NotFound("🗀 Розклади для групи не знайдено!");
        }

        scheduleRepository.RemoveRange(schedules);
        await scheduleRepository.UnitOfWork.SaveChangesAsync();

        return ResponseFactory.Success();
    }

    private void SendUserUpdateScheduleStatus(SchedulesEntity schedule, long groupId, ChangeType changeType)
    {
        var now = DateTime.Now;
        DateOnly dateOnly = DateOnly.FromDateTime(now);

        var pairTime = ClassTimes.Times[schedule.PairNumber];
        var currentTime = now.TimeOfDay;


        if (dateOnly == schedule.Date && currentTime < pairTime)
        {
            BackgroundJob.Enqueue<IScheduleChangeStatusJob>(job => job.HandleAsync(groupId, schedule.Id,
changeType));
        }
    }
}
```

RecurringJobsService.cs

```
public class RecurringJobsService : BackgroundService
 {
    private readonly IServiceProvider _serviceProvider;
```

```
    public RecurringJobsService(IServiceProvider serviceProvider)
    {
        _serviceProvider = serviceProvider;
    }

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        using var scope = _serviceProvider.CreateScope();
        var universityRepository = scope.ServiceProvider.GetRequiredService<IUniversityRepository>();
        var universities = await universityRepository.GetUniversitets();

        foreach (var university in universities)
        {
            RecurringJob.AddOrUpdate<IScheduleNotifierJob>(
                $"daily_notify_student_job_{university.Id}",
                job => job.HandleAsync(university.Id),
                "30 7 * * 1-5"
            );
        }
    }
}
```

TelegramBotService.cs

```
public class TelegramBotService
{
    private readonly TelegramBotClient _botClient;
    private readonly UserManager _userManager;
    private readonly UserService _userService;
    private readonly IUniversityRepository _universityRepository;
    private readonly IGroupRepository _groupRepository;
    private readonly IScheduleRepository _scheduleRepository;

    public TelegramBotService(
        UserManager userManager,
        UserService userService,
        TelegramBotClient botClient,
        IUniversityRepository universityRepository,
        IGroupRepository groupRepository,
        IScheduleRepository scheduleRepository
        )
    {
        _botClient = botClient;
        _userManager = userManager;
        _userService = userService;
        _universityRepository = universityRepository;
        _groupRepository = groupRepository;
        _scheduleRepository = scheduleRepository;
    }

    /// <summary>
    /// Handles incoming Telegram updates, such as text messages or contacts.
    /// </summary>
    /// <param name="update">The update received from Telegram.</param>
    public async Task HandleUpdateAsync(Update update)
    {
        // Check if the update is a message with text
        if (update.Type == UpdateType.Message && update.Message?.Text != null)
```

```
{
    var message = update.Message;
    var chatId = message.Chat.Id;
    var rawText = message.Text.Trim().ToLower();
    string pattern = @"(\p{Cs}|\p{So}|\p{Sk}|[\uFE00-\uFE0F])+(\s)?";
    var text = Regex.Replace(rawText, pattern, "");
    var dateTime = DateTime.Now;

    switch (text)
    {
        case "/start":
            // Handle the /start command
            await HandleStartCommand(chatId);
            break;

        case "налаштування":
            // Open settings menu
            await OpenSettings(chatId);
            break;

        case "головне меню":
            // Return to the main menu
            await MainMenu(chatId);
            break;

        case "обрати університет":
            // Show list of available universities
            await OpenListOfUniversitets(chatId);
            break;

        case string s when s.StartsWith("університет:"):
            // Select a specific university
            await SelectUniversity(chatId, text);
            break;

        case "обрати групу":
            // Show list of available groups
            await OpenListOfGroups(chatId);
            break;

        case string s when s.StartsWith("група:"):
            // Select a specific group
            await SelectGroup(chatId, message.Text);
            break;

        case "розклад":
            // Open the schedule menu
            await OpenScheduleMenu(chatId);
            break;

        case "розклад на сьогодні":
            // Send today’s schedule
            DateOnly today = DateOnly.FromDateTime(dateTime);
            await SendSchedules(chatId, today, today);
            break;

        case "розклад на тиждень":
            // Send schedule for the upcoming week
            DateOnly from = DateOnly.FromDateTime(dateTime);
```

```
            DateOnly to = DateOnly.FromDateTime(dateTime.AddDays(7));
            await SendSchedules(chatId, from, to);
            break;

        case "допомога":
            // Send help message with list of commands
            await _botClient.SendMessage(
                chatId: chatId,
                text:
                "🆘 *Help*\n\n" +
                "Here are the available commands:\n\n" +
                "▶ /start — start working with the bot\n" +
                "⚙ Налаштування — open settings menu\n" +
                "🏫 Обрати університет — list of universities\n" +
                "👥 Обрати групу — list of groups\n" +
                "🗓 Розклад — open schedule menu\n" +
                "📅 Розклад на сьогодні — shows today’s schedule\n" +
                "🆔 Розклад на тиждень — shows weekly schedule\n" +
                "ℹ Допомога — this help menu\n\n" +
                "_Якщо у вас виникли проблеми, зверніться до адміністратора_",
                parseMode: ParseMode.Markdown);
            break;

        default:
            // Handle unknown commands
            await _botClient.SendMessage(
                chatId: chatId,
                text: "❓ Unknown command. Type 'допомога' to see available options.");
            break;
    }
  }
  // Check if the message contains a contact (e.g., phone number)
  else if (update.Type == UpdateType.Message && update.Message?.Contact != null)
  {
      var message = update.Message;
      await HandleContactAsync(message);
      await OpenSettings(message.Chat.Id);
  }
}

#region Private

/// <summary>
/// Handles the /start command. If the user is known, shows the main menu; otherwise, requests contact info.
/// </summary>
private async Task HandleStartCommand(long chatId)
{
  // Check if the user already exists in the system
  if (await _userManager.ExistAsync(x => x.TelegramChatId == chatId))
  {
      await MainMenu(chatId);
  }
  else
  {
      // Ask the user to send their phone number to subscribe
      await _botClient.SendMessage(
          chatId: chatId,
```

```
            text: "👋 Привіт! Щоб підписатися на сповіщення, надішли свій номер телефону 👇👇👇",
            replyMarkup: new ReplyKeyboardMarkup(new[]
            {
                new KeyboardButton("📱 Надіслати номер телефону")
                {
                    RequestContact = true
                }
            })
            {
                ResizeKeyboard = true,
                OneTimeKeyboard = true
            });
    }
}

/// <summary>
/// Shows the main menu with primary options.
/// </summary>
private async Task MainMenu(long chatId)
{
    var keyboard = new ReplyKeyboardMarkup(new[]
    {
        new KeyboardButton("⚙ Налаштування"),
        new KeyboardButton("📅 Розклад"),
        new KeyboardButton("🆘 Допомога")
    })
    {
        ResizeKeyboard = true
    };

    await _botClient.SendMessage(chatId, "🔎 Оберіть команду 👇👇👇", replyMarkup: keyboard);
}

/// <summary>
/// Opens the settings menu to allow selecting university or group.
/// </summary>
private async Task OpenSettings(long chatId)
{
    var keyboard = new ReplyKeyboardMarkup(new[]
    {
        new KeyboardButton("🏫 Обрати університет"),
        new KeyboardButton("👥 Обрати групу"),
        new KeyboardButton("🏠 Головне меню")
    })
    {
        ResizeKeyboard = true
    };

    await _botClient.SendMessage(chatId, "⚙ Оберіть налаштування 👇👇👇", replyMarkup: keyboard);
}

/// <summary>
/// Shows the list of universities to select from.
/// </summary>
private async Task OpenListOfUniversitets(long chatId)
{
    // Fetch available universities from repository
```

```
        var universities = await _universityRepository.GetUniversitets();

        // Convert universities into button layout (2 per row)
        var keyboardButtons = universities
            .Select(u => new KeyboardButton($"🏛 Університет: {u.Name}"))
            .Chunk(2)
            .Select(row => row.ToArray())
            .ToArray();

        var replyKeyboard = new ReplyKeyboardMarkup(keyboardButtons)
        {
            ResizeKeyboard = true,
            OneTimeKeyboard = true
        };

        await _botClient.SendMessage(chatId, "🏛 Оберіть університет 👇👇👇", replyMarkup: replyKeyboard);
    }

    /// <summary>
    /// Handles university selection and saves it for the user.
    /// </summary>
    private async Task SelectUniversity(long chatId, string text)
    {
        // Extract the university name from the message
        var universityName = text.Replace("університет: ", "").Trim();

        // Try to find the university in the repository
        var university = await _universityRepository.FirstOrDefaultAsync(x => x.Name == universityName);

        if (university != null)
        {
            // Find the user by Telegram chat ID
            var user = await _userManager.FindByTelegramChatIdAsync(chatId);

            if (user != null)
            {
                // Update user's selected university
                user.UniversityId = university.Id;
                user.GroupId = null; // Clear group selection
                await _userManager.UpdateAsync(user);

                await _botClient.SendMessage(chatId, $"✅ Ви обрали університет: *{university.Name}*", parseMode:
ParseMode.Markdown);
                await OpenListOfGroups(chatId); // Prompt to select group next
            }
        }
        else
        {
            // University not found
            await _botClient.SendMessage(chatId, "❌ Університет не знайдено.");
        }
    }

    /// <summary>
    /// Opens a list of available groups for the user based on their university
    /// </summary>
    private async Task OpenListOfGroups(long chatId)
    {
```

```
        var user = await _userManager.FindByTelegramChatIdAsync(chatId);
        if (user != null && user.UniversityId != null)
        {
            var groups = await _groupRepository.GetGroupsNames(user.UniversityId.Value);
            if (groups is { Count: 0 })
            {
                await _botClient.SendMessage(chatId, "❌ У цього університету немає груп для вибору. Будь ласка, оберіть інший університет або зверніться до адміністрації.");
                await MainMenu(chatId);
                return;
            }

            // Create keyboard buttons for each group (2 in each row)
            var keyboardButtons = groups
                .Select(g => new KeyboardButton($"Група: {g}")) // 🏷 Group label
                .Chunk(2)
                .Select(row => row.ToArray())
                .ToArray();

            var replyKeyboard = new ReplyKeyboardMarkup(keyboardButtons)
            {
                ResizeKeyboard = true,
                OneTimeKeyboard = true
            };

            // Send group selection prompt with keyboard
            await _botClient.SendMessage(chatId, "📋 Оберіть групу 👇👇👇", replyMarkup: replyKeyboard);
        }
        else
        {
            await _botClient.SendMessage(chatId, "❌ Виникла помилка при отриманні списку груп.");
            return;
        }

        if (user == null)
        {
            await _botClient.SendMessage(chatId, "❌ Користувача не знайдено. Спробуйте ще раз.");
            return;
        }
        else if(user.UniversityId == null)
        {
            await _botClient.SendMessage(chatId, "ℹ️ Для отримання списку груп потрібно обрати університет");
            await OpenListOfUniversitets(chatId);
            return;
        }
        else
        {
            var groups = await _groupRepository.GetGroupsNames(user.UniversityId.Value);

            // Create keyboard buttons for each group (2 in each row)
            var keyboardButtons = groups
                .Select(g => new KeyboardButton($"Група: {g}")) // 🏷 Group label
                .Chunk(2)
                .Select(row => row.ToArray())
                .ToArray();

            var replyKeyboard = new ReplyKeyboardMarkup(keyboardButtons)
            {
```

```
            ResizeKeyboard = true,
            OneTimeKeyboard = true
        };
    }
}

/// <summary>
/// Handles user's group selection and updates their profile
/// </summary>
private async Task SelectGroup(long chatId, string text)
{
    var groupName = text.Replace("Група: ", "").Trim();
    var group = await _groupRepository.FirstOrDefaultAsync(x => x.Name == groupName);

    if (group != null)
    {
        var user = await _userManager.FindByTelegramChatIdAsync(chatId);

        if (user == null)
        {
            await _botClient.SendMessage(chatId, "❌ Користувача не знайдено. Спробуйте ще раз.");
            return;
        }
        else
        {
            user.GroupId = group.Id;
            await _userManager.UpdateAsync(user);

            // Confirm group selection
            await _botClient.SendMessage(chatId, $"✅ Ви обрали групу: {group.Name} 🎓");
            await MainMenu(chatId);
        }
    }
    else
    {
        await _botClient.SendMessage(chatId, "❌ Групу не знайдено. Спробуйте ще раз.");
        return;
    }
}

/// <summary>
/// Opens the menu for selecting a schedule type (today or the week)
/// </summary>
private async Task OpenScheduleMenu(long chatId)
{
    var keyboard = new ReplyKeyboardMarkup(new[]
    {
        new KeyboardButton("📅 Розклад на сьогодні"),
        new KeyboardButton("🗓 Розклад на тиждень"),
        new KeyboardButton("🏠 Головне меню")
    })
    {
        ResizeKeyboard = true
    };

    // Prompt user to select the schedule type
    await _botClient.SendMessage(chatId, "ℹ️ Оберіть тип розкладу 👇👇👇", replyMarkup: keyboard);
}
```

```
    private async Task SendSchedules(long chatId, DateOnly from, DateOnly to)
    {
        var user = await _userManager.FindByTelegramChatIdAsync(chatId);
        if (user is null)
        {
            await _botClient.SendMessage(chatId, "❌ Користувача не знайдено. Спробуйте ще раз.");
            return;
        }
        else if(user.UniversityId == null)
        {
            await _botClient.SendMessage(chatId, "ℹ️ Для отримання розкладу потрібно обрати університет");
            await OpenListOfUniversitets(chatId);
            return;
        }
        else if(user.GroupId == null)
        {
            await _botClient.SendMessage(chatId, "ℹ️ Для отримання розкладу потрібно обрати групу");
            await OpenListOfGroups(chatId);
            return;
        }
        else
        {
            var schedules = await _scheduleRepository.GetByDateRangeAsync(user.UniversityId.Value ,user.GroupId!.Value,
from, to, []);

            // Group schedules by date
            var groupedByDate = schedules
                .GroupBy(s => s.Date)
                .Select(group => group.ToList())
                .ToList();

            if (groupedByDate == null || !groupedByDate.Any())
            {
                await _botClient.SendMessage(chatId, "🔕 Пари у вказаний період відсутні.");
                return;
            }

            var sb = new StringBuilder();

            foreach (var schedule in groupedByDate!)
            {
                var message = ScheduleExtensions.BuildScheduleMessage(schedule);
                sb.AppendLine(message);
                sb.AppendLine();
            }

            var fullMessage = sb.ToString().Trim();

            await _botClient.SendMessage(chatId, fullMessage, parseMode: ParseMode.Markdown);
        }
    }

    /// <summary>
    /// Handles contact sharing from Telegram and registers or updates the user
    /// </summary>
    private async Task HandleContactAsync(Message message)
    {
```

```
        var phone = message.Contact!.PhoneNumber;
        var chatId = message.Chat.Id;

        var user = _userManager.Users.FirstOrDefault(u => u.PhoneNumber == phone);

        try
        {
            if (user != null)
            {
                // Link existing user to Telegram
                user.TelegramChatId = chatId;
                user.IsTelegramSubscribed = true;

                await _userManager.UpdateAsync(user);
            }
            else
            {
                // Register a new Telegram user
                await _userService.RegisterTelegramUserAsync(phone, chatId, true);
            }

            await _botClient.SendMessage(chatId, "✅ Ви успішно надали свій номер телефону.");
        }
        catch (Exception)
        {
            await _botClient.SendMessage(chatId, "⚠ Виникла неочікувана помилка. Спробуйте пізніше.");
        }
    }

    #endregion
}
```

ApplicationDbContext.cs

```
public class ApplicationDbContext : IdentityDbContext<UserEntity, RoleEntity, long>, IUnitOfWork
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
    : base(options)
    {
    }
    public override DbSet<UserEntity> Users { get; set; }
    public override DbSet<RoleEntity> Roles { get; set; }
    public DbSet<UniversityEntity> Universities { get; set; }
    public DbSet<GroupEntity> Groups { get; set; }
    public DbSet<SubjectEntity> Subjects { get; set; }
    public DbSet<SchedulesEntity> Schedules { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.ApplyConfiguration(new UserEntityConfiguration());
        modelBuilder.ApplyConfiguration(new UniversityEntityConfiguration());
        modelBuilder.ApplyConfiguration(new GroupEntityConfiguration());
        modelBuilder.ApplyConfiguration(new SubjectsEntityConfiguration());
        modelBuilder.ApplyConfiguration(new SchedulesEntityConfiguration());
    }
```

```
    #region Transaction Management

    private IDbContextTransaction? _currentTransaction;

    public bool HasActiveTransaction => _currentTransaction != null;

    public async Task<IDbContextTransaction> BeginTransactionAsync()
    {
        if (_currentTransaction != null)
            throw new InvalidOperationException("There is already an active transaction.");

        _currentTransaction = await Database.BeginTransactionAsync();
        return _currentTransaction;
    }

    public async Task CommitTransactionAsync(IDbContextTransaction transaction)
    {
        if (_currentTransaction == null || transaction != _currentTransaction)
            throw new InvalidOperationException("Transaction is not active or does not match the current transaction.");

        try
        {
            await SaveChangesAsync();
            await transaction.CommitAsync();
        }
        finally
        {
            await DisposeTransactionAsync();
        }
    }

    public async Task RollbackTransactionAsync()
    {
        if (_currentTransaction == null)
            throw new InvalidOperationException("No active transaction to rollback.");

        try
        {
            await _currentTransaction.RollbackAsync();
        }
        finally
        {
            await DisposeTransactionAsync();
        }
    }

    public IDbContextTransaction? GetCurrentTransaction() => _currentTransaction;

    private async Task DisposeTransactionAsync()
    {
        if (_currentTransaction != null)
        {
            await _currentTransaction.DisposeAsync();
            _currentTransaction = null;
        }
    }

    #endregion
}
```