Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

ЗВІТ
з комп’ютерного практикуму з дисципліни
«Проектування і розроблення інформаційних систем та технологій»
на тему

«Типова інформаційна система ВНЗ з реалізацією компоненту складання розкладів»

Виконав:
студент групи ІС-41мп
Коваль Вадим Юрійович

Перевірив:
професор кафедри ІСТ
Теленик Сергій Федорович

Київ - 2024

ЗМІСТ

ВСТУП

Управління вищим навчальним закладом (ВНЗ) – це задача не з простих, особливо, коли це управління має бути якісним та комфортним для всіх сторін таких закладів, особливо для викладачів та здобувачів освіти.

Сучасний розвиток інформаційних систем та технологій може забезпечити розробку та безперебійну роботу такої інформаційної системи, що допомогла б керівництву закладів вищої освіти працювати більш ефективно. Точніше, така система могла б пришвидшити, спростити та автоматизувати усі основні функції на всіх основних етапах керування ВНЗ.

Даний проект є неабияк актуальним сьогодні, адже навіть в еру інформаційних технологій керівництво вищих навчальних закладів і досі стикається з багатьма типовими проблемами, що можуть сповільнити процес управління, основними з яких є наступні:

- величезний обсяг «паперової» роботи;
- вирішення занадто великої кількості проблем шляхом комунікації з посередником/третьою особою (зазвичай, представлена деканатом);
- нелогічний розподіл цілої інформаційної системи на підсистеми/компоненти за функціональною складовою;
- застарілі технології та підходи розробки програмного забезпечення, відсутність підтримки системи, проблеми продуктивності системи тощо.

Інформаційна система вищих навчальних закладів значно зменшила б кількість «паперової» роботи шляхом надійного збереження даних на хмарних провайдерах, вирішила б проблему зайвої комунікації, автоматизуючи більшість функцій компонент та обробляючи можливі помилкові ситуації під час виконання типових функцій керування ВНЗ.

В даній роботі ми розглянемо основні засади створення системи ВНЗ, визначимо цілі розробки та обґрунтуємо вибір архітектури типової системи.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Даний розділ є фундаментальним перед початком розробки типової інформаційної системи ВНЗ з реалізацією компоненту складання розкладів. В ньому описано мету і проблеми, зроблено проектні припущення та обсяг продукту, охарактеризовано користувачів, описано організацію.

1.1 Мета

Метою даного проекту є спрощення процесів закладів вищої освіти, зокрема процесу керування закладом освіти, шляхом створення та впровадження типової інформаційної системи ВНЗ.

Проект має на меті створення інфраструктури для незалежної комфортної роботи з інформацією для керівництва, викладацького складу та студентства, що в свою чергу дозволить ефективно керувати, працювати й навчатись у ВНЗ.

Насправді, дана мета має досить багато аспектів, зокрема:

- *Спрощення процесу викладання та навчання у ВНЗ*: інформаційна система не лише прискорить роботу керуючих закладами вищої освіти, а й загалом зробить простішим життя як для викладачів, так і для студентів, забезпечуючи їм доступ до окремих автоматизованих компонентів системи;
- *Збільшення ефективності роботи вищих навчальних закладів*: рішення буденних задач для керівництва ВНЗ вже реалізовані у вигляді інформаційної системи і більшість з них можна виконати у декілька кліків або автоматично;
- *Доступність інформації ВНЗ*: інформаційна система дає можливість вводити та отримувати необхідну інформацію в будь-який зручний для кожного користувача час, важлива інформація є доступною коли вона потрібна, причому без зайвих комунікацій (особливо актуально у неробочі години закладу освіти).

1.2 Опис проблеми, проектні припущення, обсяг продукту

Основними **проблемами**, з якими можуть стикатись користувачі типових інформаційних систем ВНЗ, є наступні:

- *Різнорідні дані*: ВНЗ часто використовують декілька систем (іноді навіть занадто багато) для бухгалтерії, студентського обліку, управління навчальним процесом тощо, що призводить до складнощів при інтеграції та синхронізації даних між усіма окремими компонентами системи для єдиного ВНЗ;
- *Старі системи*: багато закладів вищої освіти використовують застарілі системи та технології, що важко чи неможливо інтегрувати в нові рішення;
- *Завантаженість системи*: під час пікових періодів (екзаменаційна сесія, запис на курси тощо), системи можуть бути перевантаженими, що призводить до повільної роботи або навіть технічних збоїв;
- *Неінтуїтивний інтерфейс*: часто інформаційні системи ВНЗ мають складний інтерфейс, що ускладнює роботу студентів, викладачів та адміністративного (керуючого) персоналу.

Можна виділити й проблеми, що власне й **вимагають створення системи створення розкладів** вищих навчальних закладів, зокрема:

- Колізії при заповненні розкладу: пара в одного викладача у декількох груп (у випадку практики) одразу, група має одночасно пари з вибіркових предметів, що знаходились в різних каталогах тощо;
- Незручність розкладів для викладачів, якщо викладач хоче звільнити деякі дні від пар для власної наукової діяльності;
- Неврахування обмежень на сховища в корпусах ВНЗ в умовах війни в межах кожної з пар розкладу в певному корпусі;
- Неповне заповнення розкладу відповідно до факультету, спеціальності та сформованого на семестр навчального плану групи;
- Помилкове дублювання пар в розкладі для однієї групи.

Це далеко не всі можливі проблеми, але навіть такого невеликого списку достатньо, щоб зрозуміти, що існує реальна потреба у створенні типової інформаційної системи ВНЗ, що могла б забезпечити синхронізацію даних, надійність користування ресурсами і при цьому була б зручною в користуванні.

При створенні системи **проектні припущення** дозволяють визначити основні принципів й умови, на яких буде базуватись розробка системи. Найбільш важливими та критичними є наступні моменти:

- Припускається, що основні користувачі системи – це студенти, викладачі, адміністратори та керівництво ВНЗ. Кожна категорія має різний рівень доступу до інформації;
- Припускається, що у системі передбачено можливість інтеграції з вже існуючими рішеннями ВНЗ за потреби, якщо такі рішення існують;
- Припускається, що система підтримує механізми захисту інформації, тобто дані, що зберігаються в базах даних, які використовує система, надійно захищені відповідно до чинного законодавства про захист персональних даних;
- Припускається, що інфраструктура системи передбачає можливість її розгортання як на власних серверах ВНЗ, так і на хмарних середовищах.

На початкових етапах досить складно описати обсяг майбутнього програмного продукту в повній мірі, але можна скласти перелік компонентів, функціональність яких із великою ймовірністю повинна бути наявною в типовій інформаційні системі ВНЗ.

SaaS-сервіс – це хмарний сервіс, що можна використовувати онлайн, таке рішення працює на основі підписки. Серед переваг рішення: невисока вартість послуг, простота налаштування та можливість використання командою, висока швидкість впровадження в роботу. Зазвичай, така система при реалізації у вигляді SaaS (Software as a Service) рішення, тобто у вигляді веб-системи без встановлення додаткового ПЗ, має наступний **обсяг продукту**:

- *Віртуальний деканат та сервіс заяв*: оформлення заяв про академічну відпустку, переведення, відрахування тощо;

- *Особові дані*: зберігання та управління персональними даними для студентів (ПІБ, контактна інформація, паспортні дані тощо);
- *Плата*: модуль для управління оплатою за навчання, студентський гуртожиток та інші послуги, що надає ВНЗ;
- *Календар*: спільний ресурс з розкладом семестрів, іспитів і дедлайнами;
- *Поштова книга*: інструмент для внутрішньої комунікації між студентами, викладачами та адміністрацією;
- *Вибір студента*: компонент для організації процесу вибору вибіркових дисциплін, запису на гуртки та організації;
- *Складання розкладу*: автоматизоване складання та управління розкладом занять, враховуючи ресурси ВНЗ;
- *CMS*: внутрішній форум з новинами, анонсами та інформацією;
- *Фінансові звіти*: генерація звітів щодо фінансових операцій ВНЗ;
- *Господарчий відділ*: модуль для управління майном, ресурсами та інвентарем, а також організації ремонтів та закупівель.

Перед впровадженням системи як SaaS-рішення, необхідно пам’ятати про вимоги до надійності в контексті такого підходу, що стосуються здатності сервісу функціонувати стабільно, без помилок та збоїв протягом заданого часу. Висока доступність сервісу є критично важливою для SaaS-продуктів і зазвичай метою є досягнення принципу «трьох дев’яток», тобто 99.9% або більше часу безвідмовної роботи сервісу, реалізованого як Software as a Service.

Надійність – це ключовий параметр збереження довіри користувачів та забезпечення безперервності бізнесу. SaaS-система повинна мати мінімум дві копії (instance): гарячу та робочу.

Причому кожна з копій має своя копію провайдера баз даних. Наявність декількох копій сервісу повинне гарантувати, що у разі збою система матиме здатність швидко відновитись та швидко переключитись на резервні сервери (failover systems) у разі збоїв.

1.3 Характеристика користувачів

Як вже зазначалось у проектних припущеннях, типова інформаційна система ВНЗ передбачає взаємодію багатьох користувачів, причому ці користувачі поділені за ролями. Кожна роль відповідає своєму унікальному набору прав та доступів до інформації й функціональності системи.

1. Студент

Характеристика: здобувачі освіти, можуть бути бакалаврами, магістрами чи аспірантами. Студенти є переважаючою більшістю серед всіх користувачів системи. Дані користувачі використовують типову систему ВНЗ для власних навчальних та адміністративних цілей.

Основні обов'язки та права: реєстрація на курси, перегляд розкладу занять та календаря навчання, перевірка інформації для оплати за навчання, обрання вибіркових дисциплін, запис на гуртки.

2. Викладач

Характеристика: професорсько-викладацький склад, асистенти, старші викладачі та інші науково-педагогічні працівники. Дані користувачі використовують типову систему ВНЗ для управління навчальним процесом, оцінювання студентів, комунікації зі студентами.

Основні обов'язки та права: управління навчальними курсами та матеріалами, ведення електронного журналу оцінок та відвідуваності, комунікація зі студентами, відслідковування власного навчального навантаження та планування робочих годин.

3. Технічний персонал (системний адміністратор, IT-спеціаліст тощо)

Характеристика: відповідальний за підтримку працездатності інформаційної системи та за її своєчасні оновлення, захист даних і безперебійне функціонування всіх складових інформаційної системи.

Основні обов'язки та права: має доступ до всіх частин застосунку для коректного моніторингу проблем та їх оперативного вирішення.

4. Адміністративний персонал

Характеристика: найрізноманітніша група користувачів, в залежності від обраного розподілу ролей, тобто якщо кожна з підролей має доступ лише до своєї частини функціоналу, може бути поділена на деканів, керівників факультетів, ректорів й проректорів, фінансових працівників (бухгалтерія, фінансовий відділ), інспектори кадрового та господарчого відділів тощо.

Основні обов'язки та права: залежить від обраної ролі та системи розподілу ролей. Попередньо передбачено повний доступ для керівничих посад та обмежений доступ для працівників окремих відділів ВНЗ.

1.4 Опис організації

Важливо розуміти структуру організації, що бере відповідальність за розробку даного проекту. Хоча й кожен із підрозділів організації має свої особливості та обов'язки, мета співпраці даних підрозділів єдина – створення такої інформаційної системи ВНЗ, що забезпечила б комфортне керування, викладання та навчання у ВНЗ та спростила б всі необхідні процеси.

Окрім цього, важливо розуміти й структуру самого вищого навчального закладу, для якого система розробляється. Університет – це багатопрофільний ВНЗ, що забезпечує підготовку фахівців у різних галузях знань, підтримує науково-дослідну роботу та сприяє розвитку інноваційній в сфері освіти та науки. Він забезпечує підготовку студентів через академічні факультети.

1.4.1 Структура та підрозділи

Організація з розробки типової системи ВНЗ має такі **підрозділи**:

- *Відділ управління проектом*, що включає менеджерів проекту та аналітиків, за необхідності включає членів команди, які керують технічними завданнями для відділу розробки (наприклад, скрам-майстри);

- *Відділ проектування та розробки*, що включає розробників і проектувальників архітектури системи, баз даних тощо;
- *Відділ тестування*, що включає тестувальників.

За необхідності в межах організації можуть бути виокремлені наступні **додаткові відділи**:

- *Відділ впровадження*: налаштування системи та інтеграцій;
- *Відділ безпеки*: загальна кібербезпека та механізми захисту;
- *Відділ навчання*: тренінги використання системи для персоналу ВНЗ.

Організаційна структура вищого навчального закладу має такі **підрозділи**:

- *Академічні підрозділи:*

– *Факультети та інститути:* кожен факультет чи інститут відповідає за конкретні напрямки підготовки фахівців. Факультети поділяються на кафедри, що забезпечують викладання спеціальних дисциплін;

– *Кафедри:* кафедри є основними навчальними та науковими одиницями факультетів. Вони займаються викладацькою роботою, науковими дослідженнями, а також підготовкою навчальних матеріалів.

- *Адміністративні підрозділи:*

– *Ректорат:* вищий орган управління університету, до якого входять ректор, проректори та інші керівники. Ректорат відповідає за загальне стратегічне управління університетом, прийняття рішень щодо розвитку, фінансів, кадрової політики;

– *Проректори:* проректори керують різними напрямками діяльності університету, навчальною, науково-дослідною, міжнародною діяльністю, студентськими питаннями, адміністративно-господарською роботою.

- *Сервісні підрозділи:*

– *Відділ кадрів:* займається питаннями кадрового забезпечення, набору персоналу, формування та ведення особових справ співробітників, оцінювання їхньої діяльності;

– *Бухгалтерія:* відповідає за фінансове забезпечення університету, ведення бухгалтерського обліку, контроль за витратами та доходами закладу;

– *ІТ-відділ:* відповідає за технічне забезпечення університету, підтримку інформаційних систем та мереж, обслуговування комп'ютерної техніки.

- *Наукові підрозділи:*

– *Науково-дослідні інститути та центри:* займаються науковими дослідженнями у різних галузях знань. Підрозділи активно співпрацюють з факультетами для залучення студентів до наукової роботи;

– *Аспірантура та докторантура:* відповідають за підготовку наукових кадрів вищої кваліфікації. Координують діяльність аспірантів та докторантів, забезпечують науковий супровід їхньої діяльності.

- *Підрозділи зі студентських питань*

– *Деканати:* кожен факультет має свій деканат, який займається питаннями організації навчального процесу, ведення документації студентів, формування розкладу та контролю академічних успіхів студентів;

– *Студентські ради та об'єднання:* студентські організації представляють інтереси студентів у взаємовідносинах з адміністрацією, організовують культурні та спортивні заходи, сприяють формуванню студентського самоврядування.

1.4.2 Зацікавлені сторони та користувачі системи

Кожна зі сторін так чи інакше зацікавлена у розробці чи/та користуванні інформаційною системою ВНЗ. Деякі з них безпосередньо впливають на розробку, а інші – опосередковано. Основними **зацікавленими сторонами** є:

- *Адміністрація ВНЗ*: зазвичай виступають замовниками системи, саме вони визначають основні функції та стратегію впровадження системи, ухвалюють рішення щодо строків виконання робіт та бюджету;

- *ІТ-відділ ВНЗ*: впроваджують та обслуговують безпосередньо систему. Співпрацюють з розробниками та адміністрацією для стабільної роботи системи;
- *Викладачі*: користувачі, що взаємодіють зі студентами через електронні ресурси, іноді формують самостійно навчальні плани та розклади тощо;
- *Студенти*: кінцеві користувачі, що використовують інформаційну систему для отримання інформації про навчання, перегляду розкладів тощо;
- *Міністерство освіти та науки*: може виступати як регулятор чи надавати методичні рекомендації про використання систем у ВНЗ;
- *ІТ-компанії, що є партнерами ВНЗ*: компанії, що можуть бути залучені до розробки та підтримки інформаційної системи ВНЗ. Вони надають технічну експертизі, створюють та інтегрують нові технології, допомагають підтримувати систему в актуальному стані. Також ці компанії можуть мати інтерес у співпраці з ВНЗ для пошуку нових талантів серед студентів, а також для тестування або впровадження інноваційних технологій:
- *Компанії-роботодавці*: корпорації та компанії, що зацікавлені у студентах ВНЗ як потенційних співробітниках. Вони можуть використовувати систему для даних про випускників, сприяти впровадженню спеціалізованих навчальних програм для підготовки фахівців відповідно до потреб ринку;
- *Партнерські освітні установи*: навчальні заклади, що співпрацюють з ВНЗ, та які можуть використовувати систему для обміну студентами, впровадження спільних програм навчання або для наукових досліджень;
- *Академічні дослідницькі організації*: установи, що проводять дослідження в галузі освітніх технологій та які можуть використовувати систему для збору даних та впровадження нових моделей навчання або аналізу великих даних про навчальний процес у вищому навчальному закладі;
- *Фінансові організації та спонсори*: банки, фонди тощо, які підтримують ВНЗ або окремі освітні програми та зацікавлені в тому, щоб мати доступ до аналітичних даних, показників ефективності навчання та фінансових звітів.

2. ЗАГАЛЬНА ТЕХНІЧНА СПЕЦИФІКАЦІЯ

Даний розділ описує всі положення, що стосуються специфікації типової інформаційної системи ВНЗ, зокрема функції продукту, функціональні та нефункціональні вимоги, технічні специфікації.

2.1 Специфікація вимог

Специфікацію вимог можна поділити на основні функції продукту, функціональні та нефункціональні вимоги.

2.1.1 Основні функції продукту

Таблиця 2.1 – Основні функції продукту

Функція	**Функціональність**	**Опис**
Управління розкладом	Створення та редагування розкладу занять	Автоматичне формування розкладу занять для студентів і викладачів з урахуванням доступності викладачів та академічних програм
Електронний журнал успішності	Ведення обліку оцінок студентів	Викладачі можуть вводити оцінки та коментарі до успішності студентів, студенти мають доступ до результатів через особистий кабінет

Продовження таблиці 2.1 – Основні функції продукту

Реєстрація на курси	Автоматизація процесу вибору та реєстрації на курси	Студенти можуть реєструватися на вибрані курси, формувати індивідуальні навчальні плани та отримувати підтвердження про реєстрацію
Модуль управління стипендіями	Адміністрування та розрахунок стипендій	Система автоматизує процес нарахування стипендій, з урахуванням успішності та інших критеріїв
Адміністрування навчальних планів	Управління навчальними програмами та курсами	Викладачі та адміністратори можуть формувати навчальні програми, додавати нові курси, редагувати інформацію про курси та їх тривалість
Управління навчальними матеріалами	Завантаження та доступ до електронних ресурсів	Викладачі можуть публікувати навчальні матеріали, студенти мають до них доступ
Модуль комунікації	Інструменти для взаємодії студентів і викладачів	Можливість надсилати повідомлення, створювати обговорення, проводити опитування, брати участь у форумах в рамках навчальних курсів

Продовження таблиці 2.1 – Основні функції продукту

Модуль відвідуваності	Ведення обліку присутності студентів на заняттях	Викладачі можуть відзначати присутність або відсутність студентів на заняттях, система формує звіти для деканатів та адміністраторів
Фінансовий модуль	Облік оплати за навчання та інші фінансові операції	Система автоматизує процес виставлення рахунків за навчання, стипендії, а також забезпечує студентів інформацією щодо їхніх фінансових даних

2.1.2 Функціональні вимоги

Таблиця 2.2 – Основні функціональні вимоги до продукту

Назва вимоги	**Опис**
Управління навчальними планами та розкладом	Система повинна дозволяти створювати, редагувати та публікувати навчальні плани з підтримкою автоматичного визначення конфліктів у розкладі (перетину часу та викладачів)
Управління доступом до навчальних матеріалів	Викладачі повинні мати можливість завантажувати навчальні матеріали, а студенти повинні мати доступ до даних матеріалів. Система повинна підтримувати файли різних форматів файлів (PDF, DOCX, відео тощо).

Продовження таблиці 2.2 – Основні функціональні вимоги до продукту

Комунікація між користувачами системи	Система повинна надавати можливість надсилання повідомлень для обміну інформацією між викладачами та студентами. При отриманні нового повідомлення система повинна надсилати сповіщення електронною поштою користувачеві.
Управління відвідуваністю	Система повинна дозволяти викладачам фіксувати присутність студентів на заняттях з можливістю автоматичного формування звітів про відвідуваність. Студенти можуть переглядати свою історію відвідуваності.
Електронний журнал успішності студентів	Система повинна надавати можливість ведення електронного журналу успішності, де викладачі можуть вводити оцінки за завдання, іспити та інші академічні активності. Студенти повинні мати доступ до перегляду своїх оцінок у режимі реального часу через особисті кабінети.

2.1.3 Нефункціональні вимоги

Таблиця 2.3 – Основні нефункціональні вимоги до продукту

Назва вимоги	**Опис**
Продуктивність	Система повинна забезпечувати швидкий час відгуку на запити користувачів (менше 6-ти секунд для основних операцій, таких як завантаження оцінок). Система має підтримувати одночасну роботу не менше 500 активних користувачів.

Продовження таблиці 2.3 – Основні нефункціональні вимоги до продукту

Надійність	Система повинна мати високий рівень доступності (не менше 99.9% часу) та мати механізми автоматичного відновлення після збоїв. Максимальний час відновлення після збоїв не повинен перевищувати 1-ї години.
Безпека	Система повинна забезпечувати захист персональних даних та конфіденційної інформації відповідно до стандартів (наприклад, GDPR). Кількість невдалих спроб входу до системи обмежується 5 спробами перед блокуванням акаунту. Час автентифікації користувачів не повинен перевищувати 3 секунд.
Резервне копіювання	Система повинна мати регулярну систему резервного копіювання і механізми для швидкого відновлення даних у разі збою. Частота створення резервних копій – кожні 6 годин.
Масштабованість	При збільшенні навантаження продуктивність має падати не більше ніж на 5% при додаванні кожних 500 користувачів.
Сумісність	Інформаційна система повинна підтримувати мінімум 3 основі браузери: Google Chrome, Firefox, Microsoft Edge причому і на платформі Windows, і на macOS.
Юзабіліті	Для навчання нових користувачів на повне освоєння базових функцій системи має йти не більше 2 годин. Кількість кроків для виконання основних дій без врахування виключних ситуацій не повинна перевищувати 6-ти кроків.

2.2 Вимоги до реалізації компонента складання розкладів

Важливо перед власне програмною реалізацією компонента чітко розуміти різноманіття функцій, що мають бути реалізовані в межах компонента складання розкладів, а також розглянути можливі шляхи автоматизації процесу складання розкладів із врахуванням потреб всіх сторін цього процесу.

2.2.1 Опис функцій

Компонент для складання розкладів буде розроблено як незалежна окрема система, що може бути як інтегрована в більшу інформаційну систему (при додатковій розробці API), так і бути використана окремо будь-яким закладом вищої освіти. Це не все – даний компонент можна й використати як фундамент для розробки повної типової інформаційної системи, що описується.

Таблиця 2.4 – Функціональні вимоги до компонента складання розкладів

Назва вимоги	Опис
Авторизація	Система повинна дати змогу відповідальній особі від ВНЗ зареєструвати власний кабінет, а потім – авторизуватись. Дані для входу мають валідуватись, як-мінімум, на унікальність назви ВНЗ, для якого створюватиметься розклад, а адреса електронної пошти. Він повинен мати змогу вийти із системи.
Управління викладачами	Система повинна дати змогу створити викладача, вказуючи, як-мінімум, ПІБ викладача. Підтримується також редагування та видалення викладача.

Продовження таблиці 2.4 – Функціональні вимоги до компонента складання розкладів

Управління дисциплінами	Система повинна дати змогу створити дисципліну, вказуючи, як-мінімум, назву та опис дисципліни, курс її викладання, тип оцінювання (залік або екзамен), кількість годин на семестр. Окрім цього, система повинна надати користувачеві можливість вказати чи є дисципліна вибірковою і якому каталогу вона належить, а також закріпити за нею можливих викладачів (лекторів та практиків), що можуть викладати дану дисципліну. Підтримується також редагування та видалення дисципліни.
Управління каталогами	Система повинна дати змогу створити каталог для дисциплін, вказавши йому назву, унікальність якої має валідуватись цією ж системою. Підтримується також редагування та видалення каталогу.
Управління спеціальностями	Система повинна дати змогу створити спеціальність, для якої необхідно як-мінімум вказати унікальну назву, шифр та опис. Для кожної спеціальності користувач повинен мати можливість заповнення навчального плану, обираючи дисципліни на кожен із семестрів. а замовчуванням навчальний план розраховано на повні 4 роки навчання (8 семестрів). Підтримується також редагування та видалення спеціальності.

Продовження таблиці 2.4 – Функціональні вимоги до компонента складання розкладів

Управління факультетами	Система повинна дати змогу створити факультет на основі списку спеціальностей, вказавши також ім’я та опис факультету. Підтримується також редагування та видалення факультету.
Управління групами	Система повинна дати змогу навчальну групу, де вказано шифр та поточний курс (за замовчуванням – перший), факультет та спеціальність навчання. Підтримується також редагування та видалення груп. Система повинна дозволяти відповідальній особі автоматично в один клік збільшувати курс для всіх груп вищого навчального закладу. Система повинна показувати для яких груп розклад пустий, для яких заповнений частково, а для яких – повністю.
Доступ до розкладу	Система повинна дати змогу згенерувати унікальне посилання, що веде на всі розклади даного навчального року для груп та викладачів певного ВНЗ. Дозволено до перегляду також і неавторизованим користувачам. Поточний день в розкладі має підсвічуватись іншим кольором для зручності.
Підписка на розклад	Система повинна дати змогу підписатись на розклад певної групи на певний семестр, увівши адресу електронної пошти. Після повного заповнення розкладу (відповідно навчального плану) система має надсилати відповідне повідомлення.

Продовження таблиці 2.4 – Функціональні вимоги до компонента складання розкладів

Управління розкладами	Система повинна дати змогу заповнити пару в розкладі для певної групи, вказавши як-мінімум день та проміжок часу, дисципліну, вид проведення пари (практика чи лекція) та відповідального викладача. Система повинна автоматично показувати які з дисциплін згідно навчального плану ще потрібно додати в розклад. Система повинна комплексно валідувати додавання нової дисципліни в розклад, як-мінімум враховуючи різноманітні колізії, наприклад, зайнятість викладача, аудиторії, студентів тощо. Система повинна надавати функціональність швидкої перестановки пари, що вже додана в розклад, на інший обраний вільний день та час.
Спільний доступ	Система повинна дати можливість запросити до спільного редагування розкладів певного ВНЗ будь-якого іншого користувача за адресою електронної пошти. Запрошення користувач повинен мати змогу скасувати. При прийнятті запрошення у спливаючому вікні, користувачу відкривається доступ до даних та розкладів ще одного ВНЗ. Користувач повинен мати змогу перемикатись між усіма доступними йому закладами освіти. Можливість запрошувати до спільного доступу має лише адміністратор – «перший» користувач ВНЗ, всі інші (запрошені ним) користувачі є просто учасниками.

2.2.2 Математична модель

Типова математична модель для складання розкладу ВНЗ як компонента типової інформаційної системи ВНЗ має наступний вигляд.

Постановка задачі. Необхідно скласти розклад занять для груп студентів, де кожне заняття має бути проведене в конкретній аудиторії або дистанційно у визначений час і викладатися конкретним викладачем. При цьому слід врахувати наявність аудиторій, викладачів та інші обмеження.

Дано:

N – кількість груп студентів;

M – кількість викладачів;

T – кількість часових слотів (періодів в одному тижні);

L – кількість доступних аудиторій;

Ai,j,t – проведення заняття викладачем j з групою i в слот часу t;

Ri,l,t – наявність у групи i заняття в аудиторії l в слот часу t;

Ci,j – кількість занять, що повинен на тиждень провести викладач j для групи i;

Dj – доступність викладача j на тиждень.

Цільова функція. Мінімізувати кількість конфліктів, забезпечити рівномірний розподіл занять та максимальну ефективність використання аудиторій. Розрізняють багато підфункцій, наприклад, метою може бути максимізація ефективного використання часового простору:

$$z = \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{t=1}^{T} A_{i,j,t} * R_{i,l,t} \rightarrow \max$$

Обмеження.

Кожен викладач може проводити не більше одного заняття одночасно:

$$\sum_{i=1}^{N} A_{i,j,t} \leq 1$$

Кожна група студентів може мати не більше одного заняття у кожен момент часу:

$$\sum_{j=1}^{M} A_{i,j,t} \leq 1$$

Одна з аудиторій може бути використана не більше, ніж однією групою в кожен момент часу:

$$\sum_{j=1}^{N} R_{i,l,t} \leq 1$$

Кількість занять для кожної групи і кожного викладача повинна відповідати запланованій кількості занять на тиждень:

$$\sum_{t=1}^{T} A_{i,j,t} = C_{i,j}$$

В детальних моделях зазвичай враховуються й інші обмеження.

2.2.3 Опис методів роботи

Для розв'язання задач оптимізації розкладів існує велика кількість математичних методів, але основними методами є лінійне програмування, комбінаторні методи, генетичні алгоритми.

Також для побудови оптимального розкладу може бути застосовано метод розгалужень та меж чи евристичні методи.

1. Лінійне програмування (LP)

Лінійне програмування є одним із найпоширеніших методів оптимізації. Воно дозволяє знаходити оптимальні рішення для задач, які можна описати системою лінійних рівнянь та нерівностей.

У задачах розкладу, такими змінними можуть бути заняття, викладачі, аудиторії та обмеження на їх використання. За допомогою LP можна мінімізувати витрати чи максимізувати ефективність при дотриманні всіх обмежень.

Задачі, що вирішуються:

- Оптимізація розкладу занять або екзаменів у ВНЗ.
- Пошук оптимального розподілу ресурсів на виробництві.

2. Комбінаторні методи

Комбінаторні методи застосовуються для розв'язання задач, де потрібно знайти найкращий варіант із великої кількості можливих рішень. Для задач розкладу це може бути оптимізація комбінацій занять, викладачів і аудиторій.

Задачі, що вирішуються:

- Складання розкладу екзаменів, де кожен студент має з'явитися на певні іспити, у певний час, у різних аудиторіях.

3. Генетичні алгоритми

Генетичні алгоритми імітують процес природного відбору, де найкращі рішення "схрещуються" для створення нових варіантів, покращуючи кожну наступну "популяцію" рішень. Цей підхід застосовується для оптимізації розкладів, коли традиційні методи є надто складними.

Задачі, що вирішуються:

- Створення розкладу занять, який максимально відповідає потребам викладачів і студентів, враховуючи безліч факторів (наприклад, мінімізація перерв або накладань занять).

4. Метод розгалужень і меж (Branch and Bound)

Цей метод використовується для розв'язання задач, де потрібно знаходити оптимальні рішення через поступове обмеження простору пошуку. Метод полягає в тому, що кожне можливе рішення розбивається на менші частини (розгалуження), а ті, що не задовольняють вимоги, відкидаються (обмеження).

Задачі, що вирішуються:

- Розв'язання задач розкладу пар, де існує багато обмежень і факторів.

7. Евристичні методи

Евристичні методи використовуються для знаходження швидких рішень у випадках, коли точні алгоритми є повільними. Ці методи не завжди гарантують знаходження найкращого рішення, але швидко дають прийнятні результати.

Задачі, що вирішуються:

- Побудова розкладів для великих груп студентів або працівників.

2.3 Технічні специфікації для бізнесу

Технічні специфікації для бізнесу – це фактично інформація, що визначає вимоги, параметри та умови розробки або інтеграції інформаційних систем для бізнес-процесів організації.

2.3.1 Бізнес-цілі

Наступні **бізнес-цілі** відображають те, як інформаційна система ВНЗ може спростити роботу або досягти цілей керівництва чи окремих представників ВНЗ:

- *Оптимізація навчального процесу*: автоматизація курсів, розкладів (як для проведення дисциплін, так і загальних семестрових), управління навчальними матеріалами, проведення екзаменаційних сесій та оцінювання результатів навчання. Це дозволить підвищити ефективність викладачів та адміністрації, спростити адміністрування освітніх програм, зменшити ручні помилки. Передбачено можливість онлайн-навчання, дистанційного складання іспитів та інтеграція із системами дистанційної освіти для зручності;
- *Покращення управління студентськими даними*: зберігання усіх даних про студентів, викладачів та інших співробітників в єдиному місці, що забезпечує точний облік, спрощує ведення документації та полегшує доступ до інформації у будь-який момент. Інтеграція із зовнішніми інформаційними системами (державні реєстри, міжнародні бази даних тощо) забезпечить надійність та актуальність даних;
- *Підвищення ефективності управління ресурсами*: зручне автоматизоване управління технічними ресурсами. Це стосується бронювання аудиторій, управління обладнанням, моніторингу технічного стану ресурсів, планування закупівель і утримання інфраструктури. Оптимізація цього процесу забезпечить ефективне використання ресурсів, зменшення витрат на її підтримку та запобігання перевантаження системи;

- *Покращення комунікації*: створення єдиної системи для забезпечення взаємодії усіх зацікавлених сторін та користувачів системи ВНЗ. Система дозволить відправляти сповіщення, обмінюватись повідомленням, проводити онлайн-консультації, організовувати відеоконференції, що, в свою чергу, сприятиме налагодженню швидкого та безперервного зв'язку;
- *Безпека даних*: забезпечення кібербезпеки в інформаційному просторі системи і захисту персональних даних всіх користувачів у відповідності до стандартів та чинних законів. Реалізація багаторівневої автентифікації, шифрування даних, регулярне резервне копіювання та моніторинг системи на наявність потенційних загроз є ключовими аспектами забезпечення безпеки;
- *Підтримка наукових досліджень*: інформаційна система повинна надавати сховище наукових робіт, доступ до електронних бібліотек, мати інтеграцію з міжнародними науковими базами даних та системами цитування;
- *Підтримка стратегічного розвитку ВНЗ*: масштабованість інформаційної системи мають дозволити закладу адаптуватись до нових викликів, впроваджувати інноваційні освітні технології чи навчальні програми і як результат залишатись конкурентоспроможним на освітньому ринку.

2.3.2 Аналіз обмежень та аналіз ризиків

Важливо розуміти ймовірні **обмеження**, що зазвичай мають вплив на розробку та впровадження типових інформаційних систем для ВНЗ:

- *Правові обмеження*: необхідно дотримуватись чинного законодавства щодо захисту персональних даних;
- *Кваліфікаційні обмеження персоналу*: викладачі та адміністративний персонал можуть не мати необхідних навичок для користування системою;
- *Інфраструктурні обмеження*: існуючі або нові апаратні ресурси ВНЗ можуть не підтримувати версії платформ та бібліотек, що використовувались;

- *Фінансові обмеження*: бюджет, що виділений ВНЗ на розробку проекту, може бути значно обмежений, що потребує пошуку доступних альтернатив;
- *Часові обмеження*: розробка інформаційної системи ВНЗ може бути терміновим рішенням, через що планування розробки може бути організоване у досить зжаті строки та відповідно матиме жорсткі дедлайни.

Окрім обмежень, необхідно робити аналіз **ризиків**, щоб ідентифікувати потенційні загрози і мати шляхи їх вирішення:

- *Ризик збоїв та помилок*: можливість втрати особових даних студентів та користувачів системи без можливості відновлення через технічні помилки. Мінімальним шляхом вирішення є забезпечення створення резервних копій баз даних, які зберігають всю основну інформацію;
- *Ризик безпеки*: можливі кібератаки або витоки даних можуть призвести до розкриття або розшифрування конфіденційної інформації користувачів. Мінімальним шляхом вирішення є забезпечення кодування даних, шифрування каналів спілкування у системі комунікації між користувачами та відслідковування вразливостей, що є потенційними причинами атак;
- *Ризик впровадження*: складність інтеграції даної інформаційної системи з іншими існуючими системами та платіжними системами. Мінімальним шляхом вирішення є використання нових стабільних та сумісних версій ПЗ.

2.3.3 Продукти та послуги

Наступні **продукти та послуги** можуть стати в пригоді у процесі реалізації інформаційної системи ВНЗ для спрощення роботи закладу освіти:

- *Автоматизована система управління студентами (SIS)*: програми для управління студентськими розкладами, їх відвідуваністю та успішністю;
- *Система управління навчальним процесом (LMS)*: програмні платформи для організації дистанційного навчання;

- *Системи управління ресурсами (ERP)*: інтеграційні платформи, що мають на меті управління матеріальними ресурсами, персоналом й фінансами;
- *Платформи для комунікацій*: інтегровані рішення для спілкування через внутрішні й зовнішні повідомлення, електронну пошту та відеоконференції;
- *Системи забезпечення безпеки даних*: програмне забезпечення, що шифрує канали зв'язку, кодує збережену інформацію, контролює доступ до інформації та захищає систему від кібератак, працюючи як антивіруси чи VPN-утиліти, блокуючи доступи по IP-адресам, які не входять у список допустимих адрес під'єднання до ресурсів (так звані, white-lists).

Основною послугою ВНЗ є можливість для студентів отримати освіту, а для викладачів – можливість брати участь в науковій діяльності.

Створювана інформаційна система може поліпшити дані послуги, зокрема, завдяки автоматизації більшості важливих та типових внутрішніх процесів:

- Автоматизоване миттєве оцінювання робіт, що зменшує час очікування результатів студентами та збільшує для викладачів час на наукову діяльність;
- Автоматизовані платежі та відстеження платежів, завдяки чому можна провадити онлайн-запити та стипендію та автоматичні нагадування про платежі, наприклад, за навчання на контрактній формі або за гуртожиток;
- Автоматизація швидкого пошуку книг з інтерфейсу бібліотечної системи, що дозволяє студентам швидше знаходити потрібні методичні матеріали, а викладачам – готувати якісні наукові напрацювання;
- Автоматизація різноманітної звітності та аналітики (семестровий контроль, звіт успішності, фінансові звіти тощо), що дозволяє приймати адміністрації ВНЗ більш обґрунтовані рішення.

3. СЦЕНАРІЇ ВИКОРИСТАННЯ

В попередніх розділах було визначено, що основними діючими особами типової інформаційної системи ВНЗ є студенти, викладачі та адміністрація (сюди можуть входити і керуючі особи, і ІТ-спеціалісти ВНЗ тощо). В даному розділі описуються сценарії використання для основних підсистем типової системи.

3.1 Підсистема контролю відвідуваності та успішності студентства

Чи не найбільш важлива складова будь-якої інформаційної системи ВНЗ – функціональність, за допомогою якої можна ставити відмітки щодо присутності студентів на заняттях та виставляти оцінки. В межах даного компонента може бути автоматизований процес розрахунку середнього та рейтингового балів всіх студентів та автоматичне створення списків успішності. На рисунку 3.1 зображено діаграму використання для описаної вище підсистеми.

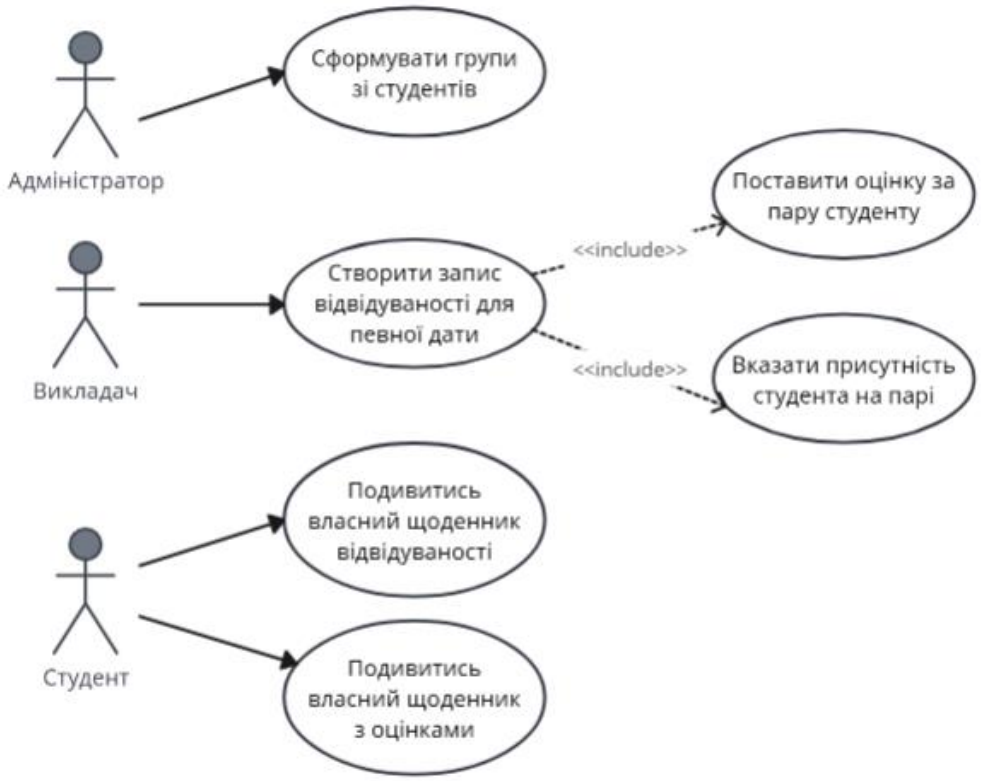

Рисунок 3.1 – Діаграма використання підсистеми відвідуваності/успішності

Деталізований опис ключового сценарію додавання запису відвідуваності підсистеми відвідуваності/успішності описано в таблиці 3.1.

Таблиця 3.1 – UC01 – Створення запису відвідуваності

Ідентифікатор і назва	**UC01 – Створення запису відвідуваності**
Основна діюча особа	Викладач
Опис	Викладач створює запис відвідуваності свого предмету для певної дати та певного студента групи
Тригер	Викладач вказує про наміри додати запис, натиснувши на відповідну кнопку «Add».
Попередні умови	PRE-1. Викладач увійшов в систему, увівши правильні дані. PRE-2. Викладач обрав необхідну дисципліну з переліку тих, що він викладає. PRE-3. Викладач обрав групу, якій дана дисципліна викладається. PRE-4. Викладач для студента (рядок таблиці відвідуваності) заповнив запис про відвідуваність на певну дату (стовпець), вказавши оцінку та проставивши позначку про відвідуваність («н» якщо не присутній).
Вихідні умови	POST-1. Запис про відвідуваність додано в щоденник відвідуваності. Він відображається для викладача та студента.

Продовження таблиці 3.1 – UC01 – Створення запису відвідуваності

Винятки	1E. Викладач не може створити запис, бо перелік дисциплін пустий через те, що у системі не прикріплено за викладачем жодного предмету. 2E. На третьому етапі система не пропонує жодної групи (не прикріплена група до дисципліни) чи група не містить студентів (не прикріплені студенти до групи). 3E. На четвертому етапі викладач не має змоги створити запис відвідуваності на потрібну дату, бо завдяки інтеграції з розкладом система знає, що на дату у групи немає такої пари. Якщо не проставлено позначку, не можна поставити оцінку.

3.2 Підсистема складання розкладу

Складання розкладу, здавалося б, саме та функція, що потребує лише людського ресурсу та часу і не може бути спрощена в типовій інформаційній системі ВНЗ. Насправді ж, в такому компоненті може бути автоматизовано обробку великої кількості можливих помилок з боку персоналу, що безпосередньо складає розклад.

Завдяки цьому не потрібно переживати за можливі колізії в побудованому розкладі. На рисунку 3.2 зображено діаграму використання для описаної вище підсистеми.

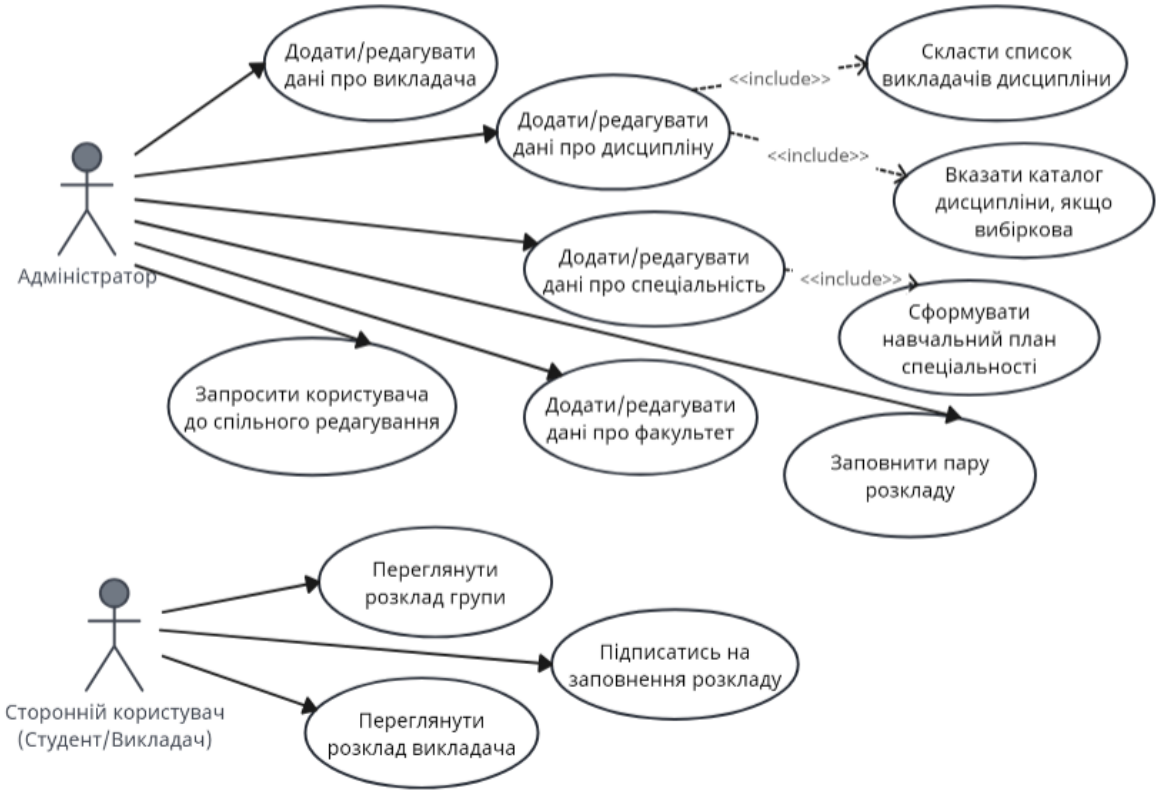

Рисунок 3.2 – Діаграма використання підсистеми складання розкладу

Деталізований опис ключового сценарію додавання дисципліни (пари) в розклад групи підсистеми складання розкладу описано в таблиці 3.2.

Таблиця 3.2 – UC02 – Заповнення пари в розкладі

Ідентифікатор і назва	**UC02 – Заповнення пари в розкладі**
Основна діюча особа	Адміністратор
Опис	Адміністратор після заповнення повної інформації для навчального закладу складає розклад для певної групи і заповнює певну комірку (пару) на певний день в навчальному розкладі.
Тригер	Оператор вказує про наміри додати дисципліну в розклад, натиснувши на відповідну кнопку «Add».

Продовження таблиці 3.2 – UC02 – Заповнення пари в розкладі

Попередні умови	PRE-1. Адміністратор увійшов в систему, увівши правильні дані. PRE-2. Адміністратор обрав необхідний навчальний заклад з переліку тих, до яких він має доступ. PRE-3. Адміністратор заповнив базу даних ВНЗ, а саме список викладачів, дисциплін (із вказаними викладачами) та їх каталогів (для вибіркових дисциплін, за необхідності), спеціальностей (на основі навчального плану) та факультетів. PRE-4. Адміністратор створив навчальні групи, вказавши їх факультет та спеціальність.
Вихідні умови	POST-1. Дисципліну успішно додано в розклад групи. POST-2. Сповіщення щодо заповнення розкладу надіслано при умові, що після додавання даної дисципліни сформовано повний розклад на поточний семестр.
Розвиток варіанту використання	1. Вибір дисципліни за назвою 2. Вибір типу дисципліни (лекція/практика) 3. Вибір викладача для даної пари

Продовження таблиці 3.2 – UC02 – Заповнення пари в розкладі

Винятки	1E. На першому етапі система не пропонує жодних дисциплін до вибору, якщо всі необхідні за навчальним планом для групи дисципліни мають і лекційні, і практичні завдання на семестр. 2E. На другому етапі система пропонує лише один із типів дисципліни, якщо пара іншого одного з типів для дисципліни вже наявна в розкладі групи. 3E. На третьому етапі система пропонує список можливих викладачів, але всі з них недоступні до вибору, бо для кожного з цих викладачів є колізія – кожен з них має іншу пару в даний день тижня на даному часовому проміжку, враховуючи тип дисципліни, що спричиняє колізію.

3.3 Підсистема вибору дисциплін

Вибір дисциплін – це невід’ємний процес сучасного процесу здобування вищої освіти, адже тепер студент самостійно може обирати дисципліни, що більш цікаві для вивчення ним особисто. Правильно організований вибір дисциплін забезпечує мінімізацію помилок в компоненті складання розкладів, адже при складанні розкладів можливі перетини дисциплін в межах однакового каталогу вибіркових дисциплін і ручні помилки при створенні каталогів

адміністраторами можуть на це вплинути. На рисунку 3.3 зображено діаграму використання для описаної вище підсистеми.

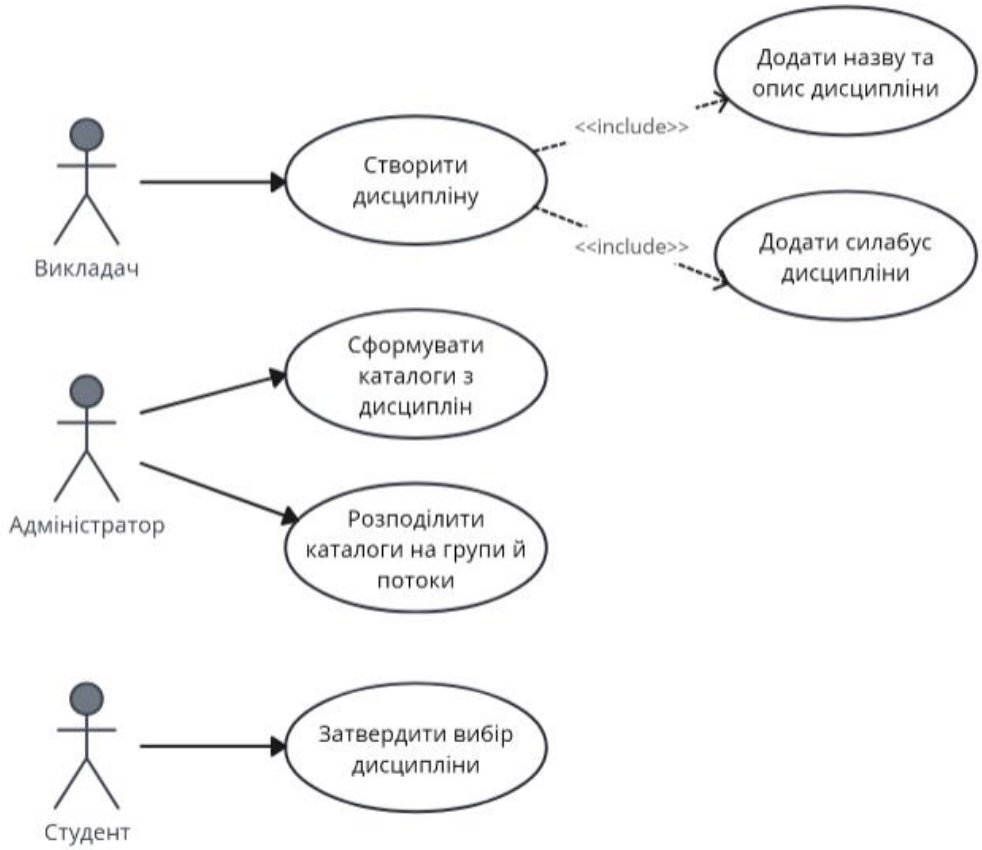

Рисунок 3.3 – Діаграма використання підсистеми вибору дисциплін

Деталізований опис ключового сценарію затвердження вибору дисципліни підсистеми вибору дисциплін описано в таблиці 3.3.

Таблиця 3.3 – UC03 – Затвердження вибору дисципліни

Ідентифікатор і назва	**UC03 – Затвердження вибору дисципліни**
Основна діюча особа	Студент
Опис	Студент затверджує вибір певної вибіркової дисципліни
Тригер	Студент вказує про наміри затвердити вибір, натиснувши на відповідну кнопку «Finish».

Продовження таблиці 3.3 – UC03 – Затвердження вибору дисципліни

Попередні умови	PRE-1. Студент увійшов в систему, увівши правильні дані. PRE-2. Студент обрав групу, в межах якої робитиме вибір (актуально при навчанні на магістратурі після бакалаврату, наприклад). PRE-3. Студент обрав каталог вибіркових дисциплін. PRE-4. Студент обрав одну з дисциплін в межах обраного раніше каталога та натиснув на кнопку затвердження вибору.
Вихідні умови	POST-1. Студента успішно додано в список студентів вибіркової дисципліни. Студент бачить оновлений статус «Обрано».
Винятки	1E. Студент не може обрати групу, бо перелік його груп пустий через те, що він не закріплений за жодною групою. 2E. На третьому етапі система не пропонує жодного каталогу дисциплін, бо жоден з каталогів не прикріплено за групою. 3E. На четвертому етапі студент не може затвердити вибір даної дисципліни через одну із можливих причин:

Продовження таблиці 3.3 – UC03 – Затвердження вибору дисципліни

	• Не наступила дата початку вибору дисциплін, тому вибір ще закрито для всіх; • Поточна дата та час – пізніше за встановлену дату кінця, тому вибір закрито; • На дисципліну вже записано максимальну кількість студентів.

3.4 Підсистема оцінювання викладачів – опитування

Для забезпечення якісної освіти ВНЗ повинен враховувати думку здобувачів освіти, тому наявність компоненти-опитування допоможе керівництву ВНЗ отримувати статистику про викладачів з точки зору студентів. На рисунку 3.4 зображено діаграму використання для описаної вище підсистеми.

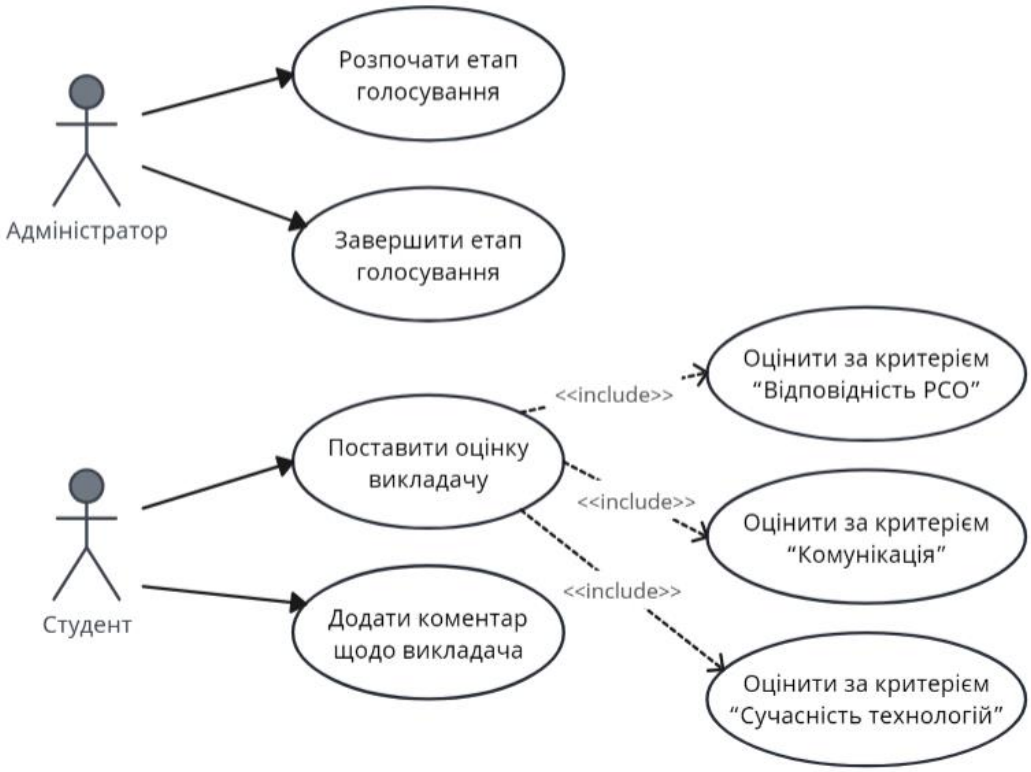

Рисунок 3.4 – Діаграма використання підсистеми оцінювання викладачів

4. АРХІТЕКТУРА ПРОДУКТУ

В даному розділі містяться огляди найбільш відомих та класичних видів сучасних архітектурних рішень для створення інформаційної системи у вигляді веб-застосунку. Окрім цього, зроблено порівняння та аналіз цих архітектур, де обрано найбільш оптимальний вид архітектури на основі таких параметрів як відмовостійкість, масштабованість і організація коду системи.

4.1 Монолітна архітектура

Монолітна архітектура — це традиційна модель розробки програмного забезпечення, також відома як архітектура веб-розробки, де все програмне забезпечення розробляється як єдиний фрагмент коду, що проходить через традиційну водоспадну модель. Всі компоненти архітектури взаємозалежні та взаємопов'язані, і кожен компонент необхідний для запуску програми. Щоб змінити або оновити конкретну функцію, потрібно змінити весь код, який буде переписано та скомпільовано.

Схема монолітної архітектури зображена на рисунку 4.1.

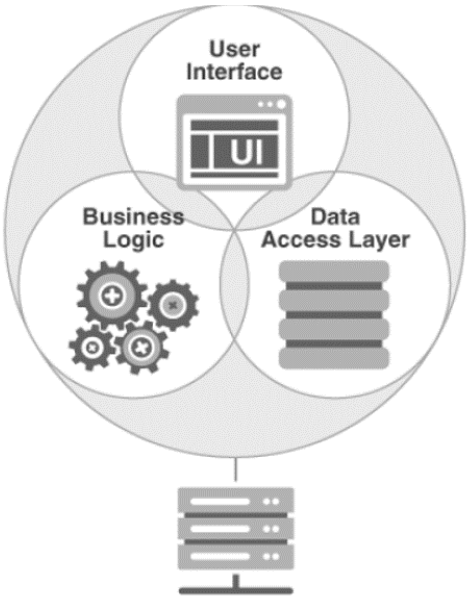

Рисунок 4.1 – Типова монолітна архітектура

Оскільки монолітна архітектура розглядає весь код як єдину програму, створення нового проекту, застосування фреймворків, скриптів та шаблонів, а також тестування стають простими та легкими. Розгортання також просте. Однак у міру того, як код стає більшим, їм стає важко керувати або вносити оновлення; навіть для малої зміни потрібно пройти весь процес архітектури веб-розробки.

Оскільки кожен елемент є взаємозалежним, масштабувати програму непросто. Більше того, це ненадійно, оскільки єдина точка відмови може вивести програму з ладу. Монолітна архітектура буде доречною, якщо потрібно створити легкий за вагою додаток з обмеженим бюджетом, а команда розробників працює з одного місця, а не віддалено.

Загалом, вважається, що це застаріле архітектурне рішення, але таке рішення має свої переваги, і воно дуже добре підходить при побудові деяких проектів. Основними **перевагами** монолітної архітектури є:

- *Простота розгортання.* Монолітну архітектуру можна швидко і відносно просто розгортати, оскільки зазвичай є єдина точка входу;
- *Розробка.* Розробка монолітної архітектури зазвичай відбувається швидко, оскільки всі компоненти та модулі знаходяться в одній кодовій базі і завжди під рукою;
- *Налагодження.* Налагодження монолітної архітектури спрощено за рахунок того, що все поряд, і є можливість відстежити всі ланки виконання коду.

Проте, монолітна архітектура має і свої **недоліки**:

- *Масштабування.* Монолітні архітектури масштабуються лише повністю, тобто якщо навантаження зростає на один модуль, не можна масштабувати тільки цей модуль, потрібно масштабувати весь моноліт;
- *Надійність.* Якщо моноліт виходить з ладу, то виходить весь цілком;
- *Зміна та оновлення технологій.* У моноліті це майже неможливо;
- *Недостатня еластичність.* Монолітні архітектури негнучкі, тобто зміна одного модуля в моноліті майже завжди впливатиме на інший модуль.

4.2 Мікросервісна архітектура

Архітектура мікросервісів вирішує кілька проблем, що виникають у монолітній архітектурі. Це архітектурне рішення, яке базується на розподілі модулів на окремі системи, які спілкуються між собою за допомогою повідомлень. Вся система - це набір маленьких систем, пов'язаних між собою.

Схема мікросервісної архітектури зображена на рисунку 4.2.

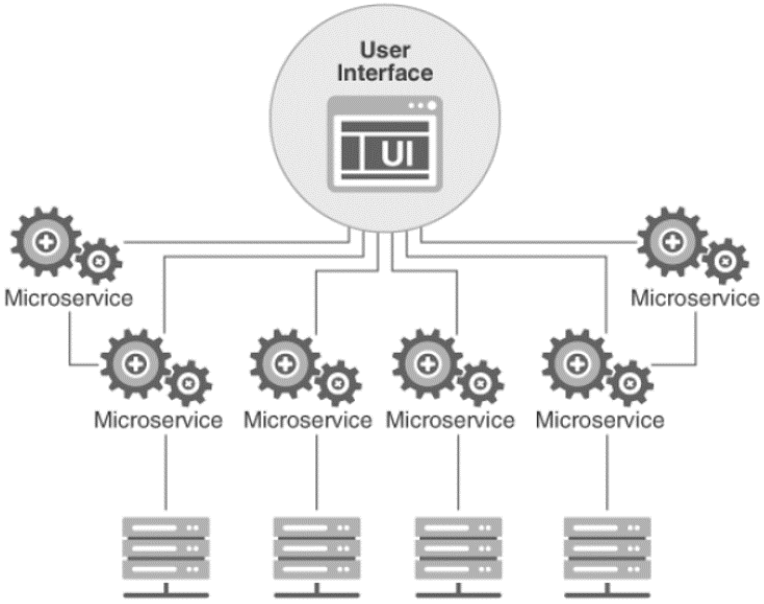

Рисунок 4.2 – Типова мікросервісна архітектура

Кожен мікросервіс містить власну базу даних та використовує певну бізнес-логіку, тобто можна легко розробляти та розгортати незалежні сервіси.

Оскільки архітектура мікросервісів слабко пов'язана, вона забезпечує гнучкість для оновлення/модифікації та масштабування незалежних сервісів. Розробка стає простою та ефективною, а безперервні доповнення стають можливими. Розробники можуть швидко адаптуватись до інновацій.

Мікросервісна архітектура — доречний вибір для високомасштабованих та складних програм. Проте, розгортання кількох служб з екземплярами середовища виконання є складним завданням. Зі зростанням кількості сервісів зростає і складність керування ними. Крім того, програми мікросервісів спільно

використовують бази даних розділів. Це означає, що потрібно забезпечити узгодженість між кількома базами даних, на які впливає транзакція. Це дуже модна і популярна архітектура, але має свої недоліки, не підходить для маленьких і середніх проектів.

Переваги мікросервісної архітектури:

- *Гнучкість*. Мікросервісна архітектура гнучка завдяки тому, що кожен сервіс є самостійною системою, тому зміни в ній можуть вплинути тільки на неї;
- *Масштабування.* На відміну від монолітної архітектури, можна масштабувати лише певну частину системи, оскільки вона є самостійною;
- *Гнучкість*. Кожна з підсистем може бути реалізована будь-якою мовою;
- *Надійність*. Вихід з ладу однієї з підсистем не ламає всю систему.

Недоліки мікросервісної архітектури:

- *Процес розробки.* Мікросервісну архітектуру непросто розробляти, оскільки потрібно виробляти кілька підсистем і налагодити взаємодію між ними;
- *Комунікація команд.* Якщо одна команда розробляє одну підсистему, то потрібно мати комунікацію, щоб налагодити взаємодію між підсистемами;
- *Налагодження.* Мікросервісну архітектуру складно налагоджувати, оскільки потрібно знайти, який сервіс зламався і чому;
- *Розгортання.* Початкове розгортання є непростим, і додавання нових сервісів вимагає налаштування ключових частин проекту

4.3 Безсерверна архітектура

Безсерверна архітектура – це альтернатива мікросервісам, яка автоматизує все розгортання завдяки хмарним технологіям. З назви можна подумати, що тут відсутня серверна частина, але це не так, тут відсутня робота із сервером як із середовищем.

Схема безсерверної архітектури зображена на рисунку 4.3.

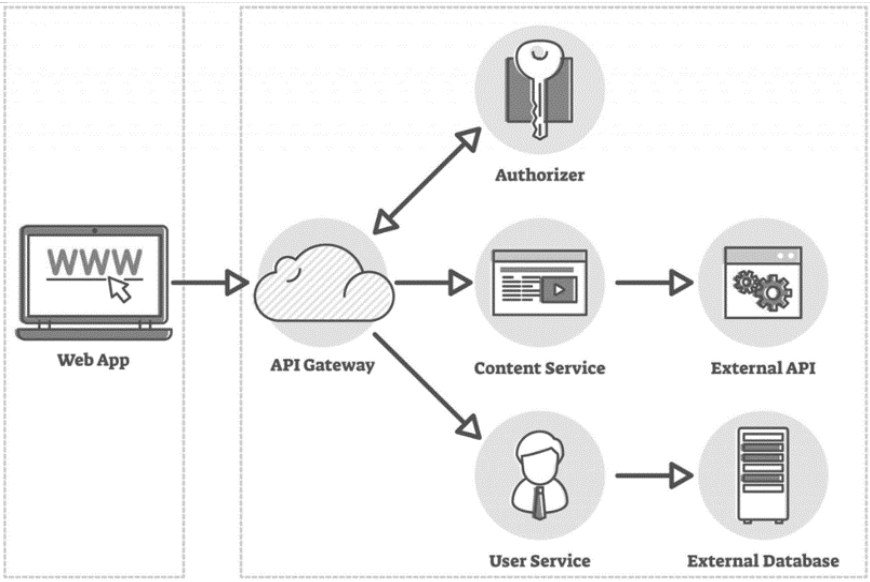

Рисунок 4.3 – Типова безсерверна архітектура

Безсерверна архітектура – це модель розробки програмних додатків, яка фокусується на розробці замість розгортання та взаємодії між сервісами. У цій структурі забезпечення базової інфраструктури керується постачальником хмарних послуг. Відбувається оплата лише за інфраструктуру, коли вона використовується, а не за час простою чи простору, що не використовується.

Безсерверні обчислення зменшують витрати, оскільки ресурси використовуються лише під час виконання програми. Завдання масштабування виконуються хмарним провайдером. Бекенд-код спрощується, що зменшує зусилля розробки, витрати і прискорює виведення результатів обробки. Обробка мультимедіа, потокова передача в реальному часі, чат-бот, повідомлення давачів IoT тощо - ось деякі з варіантів використання безсерверних обчислень.

Основними **перевагами** безсерверної архітектури є наступні:

- *Гнучкість.* Гнучка за рахунок відокремленості модулів;
- *Абстракція від операційної системи.* Хмара сама вирішує, яка ОС потрібна та як її потрібно налаштувати;
- *Легкий поріг входу.* Зазвичай серверлес це дуже простий, відокремлений код, тому розібратися з проектом нескладно;

- *Надійність.* При правильній побудові проекту надійність така сама, як і у мікросервісній архітектурі.

Безсерверна архітектура має наступні **недоліки**:

- *Гнучкість.* Масштабування обмежене хмарою;
- *Налагодження.* Ускладнене взаємодією між компонентами;
- *Прив'язка до постачальника послуг (Vendor Lock).* Кожна хмара працює за власними правилами, тому переїзд з однієї хмари на іншу майже неможливий;
- *Каскадна відмова (Cascade Failur).* Це відмова в системі взаємопов'язаних частин, у якій відмова однієї або кількох частин призводить до відмов інших частин, що прогресивно зростає в результаті позитивного зворотного зв'язку. При неправильній побудові проекту компоненти можуть сильно впливати інші компоненти, що призводить до падіння всього проекту.

4.4 Керована подіями архітектура

Архітектура, що базується на керованих подіях – це підхід до розробки програмного забезпечення, в якому система складається з компонентів, які спілкуються один з одним через обмін подіями. Кожен компонент в системі може бути виробником або споживачем подій. Події є повідомленнями про стан системи або події, які сталися в системі, і вони використовуються для сповіщення і координації між компонентами. Основні **принципи архітектури**:

- *Розділення функціональності.* Кожен компонент в системі відповідає за виконання певної функціональності та реагує на конкретні типи подій. Це дозволяє зберігати систему модульною та спрощує розробку і тестування;
- *Незалежність компонентів.* Компоненти в архітектурі, керованій подіями, можуть розвиватися та змінюватися окремо один від одного, оскільки вони спілкуються через стандартизовані події. Це сприяє гнучкості системи;

- *Асинхронна комунікація.* Події передаються між компонентами асинхронно. Це означає, що вони можуть працювати незалежно один від одного та не очікувати безпосередньої відповіді від інших компонентів;
- *Масштабованість.* Архітектура, керована подіями, дозволяє легко масштабувати окремі компоненти або типи подій в системі, щоб відповісти на збільшену навантаження.

Схема керованої подіями архітектури зображена на рисунку 4.4.

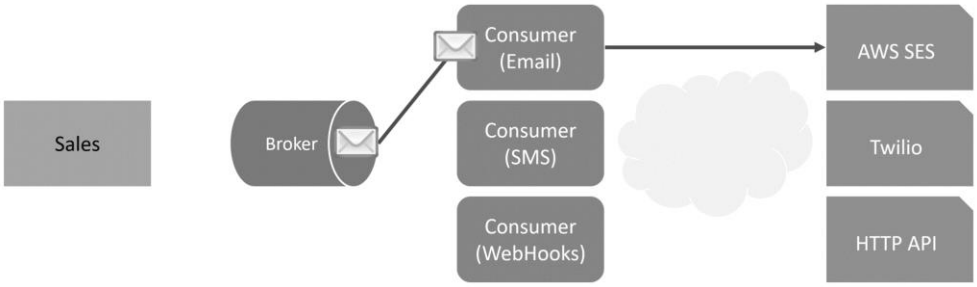

Рисунок 4.4 – Типова керована подіями архітектура

Переваги використання архітектури, керованої подіями, включають:

- *Гнучкість.* Система може легко адаптуватися до змін у вимогах або навантаженні, оскільки компоненти взаємодіють через стандартизовані події;
- *Розділення обов'язків.* Кожен компонент відповідає тільки за власні функції, що сприяє підтримці системи та уникненню залежностей;
- *Широкий вибір технологій.* Компоненти можуть бути реалізовані з використанням різних технологій і мов програмування.

Недоліки архітектури, керованої подіями, включають:

- *Складність аналізу та відлагодження*. Спостереження та аналіз потоку подій може бути складним завданням, особливо в великих системах;
- *Потенційна втрата подій.* При некоректному налаштуванні системи може виникнути ризик втрати подій;
- *Зростання витрат на мережу*. Якщо система велика та має інтенсивний обмін подіями, це може призвести до зростання навантаження на мережу.

4.5 Обрана архітектура системи

Опираючись на всі описані вище переваги на недоліки кожної з розглянутих архітектур, найбільш ефективним та сучасним рішенням є таке, що комбінує керовану подіями та мікросервісну архітектуру.

Такий підхід до проєктування завдяки розподілу компонент на окремі мікросервіси допоможе мінімізувати проблему відмови цілої системи, а налаштування «спілкування» між мікросервісами не по HTTP, а за допомогою проміжних message bus як компонентів типової керованої подіями архітектури, зробить комунікацію мікросервісів швидшою, ефективнішою, а головне – зберігатиме події для подальшого використання і гарантує відновлення системи після відмови.

Трьома **основними рівнями архітектури** є:

- Бізнес-рівень, що пропонує зовнішнім клієнтам продукти та послуги, що реалізуються через інший рівень з бізнес-процесами. На бізнес-рівні включено зацікавлені сторони (користувачів), ключові бізнес-процеси для типової інформаційної системи ВНЗ;
- Прикладний рівень, що підтримує бізнес-рівень через додатки, які в свою чергу реалізовані програмними компонентами. Даний рівень включає веб-додаток (клієнт) та інші програмні компоненти;
- Технологічний рівень, який містить інфраструктурні послуги, що необхідні для запуску програм, що виконуються комп’ютерами, пристроями зв’язку та системним програмним забезпеченням. На даному рівні описано серверні рішення, бази даних, системи зберігання та захисту даних, саме тут визначено архітектурні рішення та технології, що використовуються для безперебійної та зручної для користувачів роботи системи.

На рисунку 4.5 зображено схему архітектури типової інформаційної системи вищого навчального закладу, де схематично надано архітектурне рішення з тими основними функціями, що описано в розділі 3.

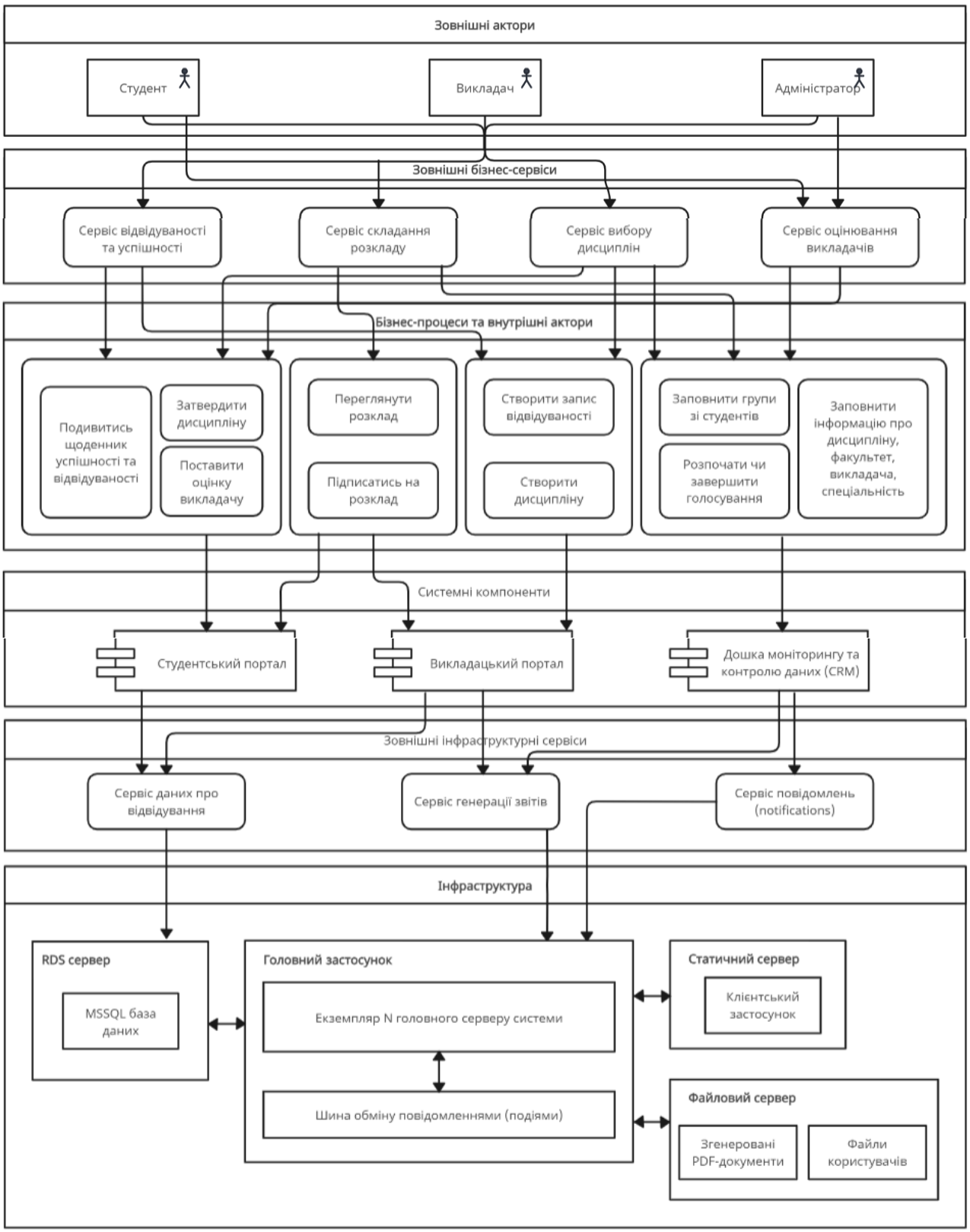

Рисунок 4.5 – Архітектура типової інформаційної системи ВНЗ

5. ТЕХНОЛОГІЇ ТА ЗАСОБИ РЕАЛІЗАЦІЇ СИСТЕМИ

Даний розділ є заключним перед програмною реалізацією інформаційної системи ВНЗ. Для клієнт-серверної архітектури важливо обрати ефективну платформу та зручну мову програмування для використання як для клієнтської, так і для серверної частини системи. В розділі представлено огляд найбільш популярних платформ та мов програмування та засобів збереження даних. На основі поданого аналізу обрано конкретні технології, що застосовуватимуться для розробки типової інформаційної системи вищого навчального закладу.

5.1 Технології розробки серверної частини системи

Згідно статистики 2024-року популярного ресурсу DOU.ua, найбільш популярними мовами програмування станом на сьогодні є такі мови програмування як Python (14,7%), Java (12,7%) та C# (12,7%).

5.1.1 Мова програмування Java

Мова програмування Java – одна з найпопулярніших мов програмування, яку відносять до об’єктноорієнтованих. Вона була розроблена американською компанією «Sun Microsystems» яка спеціалізувалась на виробництві різного програмного і апаратного забезпечення. Спочатку використовувалась виключно для програмування побутових електронних приладів.

Головними цілями при розробці Java, було те, що потрібно створити мову з подібним синтаксисом як в C/C++, але в той же час він повинен бути більш «простим, об’єктно-орієнтованим та звичним». На сьогодні Java являє собою платформу і екосистему, яка об’єднує різноманітні технології, за допомогою яких можна створити від простого десктопного додатку до веб-сервісів і великих

серверних додатків у великих корпораціях. Також на Java створюють програмне забезпечення для різних пристроїв, таких як: персональний комп'ютер, смартфони і навіть побутова техніка. Наприклад, на ОС Android велика кількість програм написано саме на ній.

Однією з важливих особливостей Java є принцип WORA (write once, run anywhere), який був реалізований в мові. Суть полягає в тому, що код написаний на Java, буде працювати на будь-якій платформі з установленою JRE, в той час, коли на інших мовах програмування буде потрібно написати код під кожну платформу. Кросплатформність досягається за допомогою компіляції коду, що написаний на Java в байт-код, який потім виконується на віртуальній машині Java, яка не залежить від платформи і є частиною JRE.

Java забезпечує механізм обміну даними та програмами між декількома комп'ютерами, що дає нам приріст продуктивності та ефективності системи. Також за допомогою технології CORBA, ми можемо обмінюватися різними об'єктами в розподіленому середовищі. Ще однією з особливостей Java є те, що в ній реалізований механізм керування пам'яттю, який називають автоматичний збирач сміття (Garbage Collector). Таким чином, програміст створює об'єкт, а JVM очищує пам'ять за допомогою GC, коли об'єкти перестають бути потрібним. Це може відбуватися у будь-який час, але найкраще, коли програма не використовується, адже цей процес може призводити до короткого зависання.

Java є надійною мовою програмування, що забезпечується двома принципами: ООП і сильна типізація. Java об'єктно-орієнтована мова програмування, а отже вона підтримує основні принципи ООП: абстракція, поліморфізм, наслідування, інкапсуляція. Всі дії і дані групуються в класи об'єктів. Всі ці об'єкти є похідними і успадковують властивості і базову поведінку від головного об'єкта.

Створена ієрархія наслідування збільшує читабельність коду і зменшує кількість невимушених помилок при його написанні. Сильна або строга типізація являє собою один із варіантів роботи з типами даних, що використовуються в

мові програмування. Java відноситься до мов програмування з сильною типізацією, а отже програмісту доводиться виконувати більший обсяг роботи, щоб присвоїти кожній змінні, константі або виразу строго певний тип, але в той же час, дані інтерпретуються однозначно, що зменшує шанс некоректної роботи.

Можна виділити наступні основні **переваги** Java:

- написане програмне забезпечення на одній платформі, може бути запущене на іншій;
- простота синтаксису, що прискорює процес вивчення мови;
- перевіряє код під час компіляції;
- дозволяє написати програму в якій можуть виконуватися декілька операцій паралельно;
- програми на мові Java дешево розробляти та обслуговувати;
- автоматичне керування пам’яттю;
- надійність.

Недоліками можна назвати:

- незважаючи на оптимізації, Java є ресурсномісткою і повільною;
- не забезпечує можливості резервного копіювання;
- написані коди на Java великі за об’ємом, що зменшує читабельність.

5.1.2 Мова програмування C#

Мова C# – це проста, сучасна, з безпечною системою типізації об’єктно-орієнтована мова програмування, яка була розроблена корпорацією Microsoft під керівництвом Андреса Гейлсберга і Скота Вільтамута в 1998 – 2001-х роках. Символ “#”, який міститься в назві можна трактувати, як дві пари плюсів, що в свою чергу натякає на новий етап у розвитку C# порівняно з C++.

Сьогодні мова програмування C# вважається одною з найпотужніших та швидко розвиваючих в IT-галузі. Завдяки зручному Сі-подібному синтаксису,

великій кількості бібліотек і фреймворків, на ній пишуть найрізноманітніші програми: мобільні додатки, комп'ютерні програми, веб-сервіси, веб-сайти, ігри та багато чого іншого.

C# також називають "очищеною версією Java", адже приблизно 70% синтаксичних можливостей він перейняв саме в неї. Ще 10% у C++ і 5% у Visual Basic, а решту 15% це і є унікальність мови – реалізація ідей розробників. Оскільки C# об'єктно-орієнтована мова програмування, то вона підтримує велику кількість корисних функцій: інкапсуляцію, поліморфізм, наслідування, а також перевантаження операторів, статичну типізацію, вказівники на функції-члени класів та багато іншого. А так як C# все ще активно розвивається, то з кожною новою версією розробники додають багато цікавих функцій, наприклад, асинхронні методи, динамічне зв'язування.

В C# як і в Java реалізований збирач сміття, який керує в автоматичному режимі виділенням і звільненням пам'яті для додатку від об'єктів, які не використовуються. Також в мові є інструмент обробки виключень, за допомогою якого програміст може виявляти і обробляти помилки в коді. Ця функція допомагає впоратися з непередбаченими проблемами, які можуть виникнути при використанні програми. C# є мовою програмування з сильною типізацією в якій діє загальна система роботи з типами. Це означає, що кожна константа і зміна має тип. Мова не дозволяє звернення до неініційованих змінних, що виключає можливість виходу за межі певного масиву даних і виконання безконтрольного приведення типів при роботі програми.

Щоб програми на C# працювали необхідним і достатнім є те, щоб була встановлена і налаштована платформа .NET Framework, яка вбудована в пакет Windows, але і існують версії для Linux і MacOS. Написаний код після компіляції перетворюється в проміжну мову, яка в свою чергу під час виконання програми на комп'ютері перетворюється на машинний код.

Це означає що мова C#, як і Java кросплатформна і її виконавчі файли можуть бути запущені без перекомпіляції на тій чи іншій платформі.

Мова програмування C# має наступні **переваги**:

- всі типи даних, які використовуються в мові мають фіксований розмір, що в свою чергу збільшує її “мобільність” і спрощує процес програмування;
- автоматичне виділення і очищення пам’яті від об’єктів, які не використовуються;
- синтаксис мови є простим і зрозумілим;
- активно розвивається і доповнюється функціонал;
- за допомогою платформи Xamarin, можуть бути написані програми для iOS, Android, MacOS і Linux;
- C# об’єднує в собі найкращі ідеї таких мов програмування, як Java, C++, Visual Basic;
- кросплатформеність.

Дана мова програмування має і **недоліки**:

- орієнтація в основному іде на Windows платформу;
- безкоштовна тільки для людей, які навчаються та індивідуальних програмістів, а для великої команди ліцензія обійдеться у велику суму;
- C# сильно залежить від .NET Framework.

5.1.3 Мова програмування Python

Python являє собою потужну об’єктно-орієнтовану мову програмування для загального призначення. Вона була розроблена нідерландським програмістом Гвідо ван Россумом і випущена в 1990 році. До 2018 року він активно брав участь у її розвитку та оптимізації.

На сьогоднішній час, Python легко конкурує з іншими мовами програмування, адже активно розвивається і з кожним роком стає все більш популярнішою мовою. Це обумовлено тим, що це універсальна мова і може використовується для вирішення великої кількісті різноманітних задач. За

допомогою Python можуть бути написані скріпти по автоматизації, ігри, вебресурси, мобільні додатки та інше.

Активно використовується при автоматизації тестування і для створення вбудованих систем для великої кількісті різноманітних пристроїв, таких як банкомати, термінали. Також великої популярності має в галузі математики, адже може використовуватися для обробки великого об'єму даних та виконання складних математичних обрахунків.

Такі компанії як Spotify і Amazon використовують Python для аналізу даних та створення рекомендацій, Disney для створення анімацій, а NASA для проведення наукових обрахунків. Однією з переваг Python є те, що він має простий і логічний синтаксис, в якому було прибрано все зайве. Це робить його дуже подібним до англійської мови, що збільшує читабельність і сприйняття коду. Цей синтаксис дозволяє програмісту написати програму з меншою кількість вихідного коду ніж на інших мовах програмування.

Python – мова програмування з строгою динамічною типізацією, тобто тип змінної буде визначатися в момент надання значення, але при зміні значення може змінюватися і тип даних, що спрощує написання коду новачку і в подальшому дає змогу уникнути безліч помилок і багів. Ще однією перевагою Python є те, що існує велика кількість фреймворків і бібліотек, що спрощують вирішення складних задач.

Також перевагою Python є те, що вона інтерпретована, тобто написаний код буде виконуватися рядок за рядком і якщо буде знайдена помилка – виконання зупиниться і буде повідомлено про неї. Python показує лише одну помилку, тобто першу, яка була виявлена, навіть якщо програма має декілька помилок. Це набагато 21 економить час при розробці і полегшує подальше налагодження програми. Інтерпретатор мови існує для багатьох популярних платформ, що дозволяє написаний код на одній платформі запустити цей же код на іншій.

Оскільки Python підтримує модульність, то написаний код може бути розділений на модулі, які потім можна використовувати заново вже в інших

програмах. Також існує велика кількість колекцій стандартних модулів, які надають різну функціональність: від введення-виведення даних до створення графічного інтерфейсу користувача.

Перевагами Python можна виділити:

- легкий для читання та вивчення;
- Python має бібліотеки які дозволяють інтегрувати з іншими мовами програмування, як C/C++ та Java;
- широке застосування в різних галузях;
- стандарт для написання коду PEP, що забезпечує читабельність коду при переході від одного розробника до іншого;
- активно розвивається і розширюється бібліотеками і фреймворками;
- кросплатформна мова;
- Open Source, що дозволяє будь-кому взяти участь в розвитку і покращенню мови.

Звичайно, що дана мова програмування має і свої **недоліки**:

- через те, що код виконується рядок за рядком це призводить до зменшення продуктивності;
- погана реалізація багатопоточності;
- велике споживання пам’яті і ресурсів;
- залежний від системних бібліотек.

5.1.4 Обрана технологія для серверної частини

C# обрано для розробки системи з кількох ключових причин, які роблять її ідеальним вибором для сучасних програмних рішень.

По-перше, C# працює в екосистемі .NET, що забезпечує багатий набір бібліотек і інструментів для розробки, а також підтримує кросплатформеність, дозволяючи системі працювати на різних операційних системах. Це особливо

важливо, якщо потрібно інтегруватися з допоміжними продуктами Microsoft, такими як Azure або SQL Server.

C# підтримує потужну об'єктно-орієнтовану парадигму програмування, що робить її ідеальною для проектів, де важлива структура, масштабованість і підтримуваність коду. Мова також має автоматичне управління пам'яттю через збирач сміття, що дозволяє уникнути проблем з витоками пам'яті та забезпечує надійну роботу системи.

Безпека є ще одним вагомим аргументом на користь C#. Вона забезпечує численні вбудовані функції безпеки, що дозволяє легко впроваджувати механізми захисту даних, такі як управління правами доступу і криптографія. Це робить C# ідеальним для систем, де безпека є пріоритетом.

Також, C# постійно впроваджує сучасні стандарти програмування, включаючи асинхронність (async/await), що дозволяє будувати продуктивні асинхронні додатки, особливо важливі для веб-систем. Велика спільнота розробників, багаті навчальні ресурси і приклади коду забезпечують легкість навчання нових програмістів і швидке вирішення технічних проблем.

Нарешті, C# добре інтегрується з іншими мовами і технологіями, такими як C++, Python або JavaScript, що важливо для проектів з багатомовною архітектурою або тих, які потребують інтеграції з існуючими системами.

Таким чином, C# обрано за його універсальність, надійність, безпеку, підтримку сучасних стандартів програмування і багатоплатформеність, що забезпечує ефективну розробку, підтримку і масштабування системи.

5.2 Додаткові технології розробки веб-серверу

В процесі сучасної розробки веб-серверу, часто використовуються допоміжні бібліотеки та утиліти для підвищення продуктивності системи, а також для полегшення реалізації й підтримки певних окремих функціональних частин інформаційної системи.

5.2.1 ASP .NET Core

ASP NET – це фреймворк для розробки веб-додатків, створений Microsoft. У світі веб-розробки ASP.NET займає важливе місце, забезпечуючи розробникам інструменти для створення масштабованих і сучасних веб-додатків. Розробники можуть зосереджуватися на бізнес-логіці, а не на деталях інфраструктури, завдяки чому процес розробки стає більш ефективним.

Однією з важливих характеристик є багатошарова архітектура, що включає ASP.NET Web Forms, ASP.NET MVC, ASP.NET Web API і ASP.NET Core. Ця гнучка структура надає розробникам різноманітні можливості, а вибір конкретного підходу залежить від вимог проекту.

Серверні елементи управління в ASP.NET надають інструменти для створення динамічних веб-сторінок, і забезпечують взаємодію з елементами управління на сервері. Порівняно з іншими веб-фреймворками, ASP.NET виділяється гнучкістю і простотою використання. ASP.NET Core вирізняється своєю модульною структурою і підтримкою кросплатформної розробки. На відміну від попередніх версій, його можна використовувати на різних операційних системах, що забезпечує більшу гнучкість при виборі платформи для розгортання додатків.

Програми ASP.NET Core підтримують програмні версії, в якій різні програми, що працюють на одному комп'ютері, можуть орієнтуватися на різні версії ASP.NET Core. Переваги використання ASP.NET Core для сучасних веб-додатків очевидні. Він забезпечує вищу продуктивність завдяки оптимізованій обробці HTTP-запитів і підтримці асинхронного програмування.

Механізм впровадження залежностей вбудований в основу фреймворка, що спрощує тестування і підтримку коду. ASP.NET Core також надає можливість розробляти мікросервісні архітектури та використовувати контейнерні технології, такі як Docker. Це робить його ідеальним вибором для створення масштабованих і легко керованих веб-додатків.

5.2.2 AutoMapper

AutoMapper – це бібліотека для .NET, яка автоматично здійснює об'єктно-об'єктне відображення (mapping). Її основне завдання — спрощення процесу копіювання даних між об'єктами з різними структурами, зокрема при роботі з DTO (Data Transfer Object) і доменними моделями. AutoMapper дозволяє мінімізувати необхідність писати ручний код для перенесення даних між схожими об'єктами.

AutoMapper на основі правил відповідності автоматично відображає властивості одного об'єкта на інший, якщо їх назви співпадають. Можна налаштувати власні правила відображення для складніших випадків або для тих властивостей, які не збігаються за назвами або типами. AutoMapper також дозволяє мапити колекції об'єктів (наприклад, списки чи масиви). Бібліотека може мапити об'єкти, які містять інші об'єкти або складні структури даних.

AutoMapper суттєво зменшує кількість вручну написаного коду для мапінгу, що спрощує підтримку коду та підвищує його зрозумілість. Налаштування і використання AutoMapper досить прості, що дозволяє легко інтегрувати його у проєкти. Замість того, щоб писати численні оператори присвоєння, достатньо сконфігурувати мапінг між двома класами. Автоматизація мапінгу дозволяє уникнути помилок, які можуть виникати при ручному перенесенні даних між об'єктами.

5.2.3 Entity Framework Core

Entity Framework Core (EF Core) — це сучасна об'єктно-реляційна система зіставлення (ORM) для .NET, що дозволяє розробникам працювати з базами даних за допомогою об'єктів C#. EF Core автоматично відображає (мапить) об'єкти програми на реляційні таблиці в базі даних, дозволяючи здійснювати

CRUD-операції (створення, читання, оновлення, видалення) без написання SQL-запитів вручну.

EF Core дозволяє працювати з базою даних через об'єкти .NET, перетворюючи об'єктно-орієнтовану модель на реляційну структуру. Він підтримує роботу з різними типами баз даних, зокрема SQL Server, SQLite, PostgreSQL, MySQL, і багатьма іншими через відповідні провайдери. EF Core підтримує використання мови LINQ для написання запитів до бази даних, що дозволяє писати запити на C#, а не на SQL.

Дозволяє створювати і застосовувати міграції, що дає змогу автоматично оновлювати структуру бази даних відповідно до змін у моделі програми. EF Core відстежує зміни в об'єктах, які завантажуються з бази даних, і автоматично застосовує ці зміни при збереженні.

EF Core підтримує Eager Loading, Lazy Loading, та Explicit Loading для завантаження пов'язаних даних (наприклад, при роботі з Foreign Key), а також генерує SQL-запити автоматично на основі LINQ-запитів або інших операцій над даними, що дозволяє розробникам зосередитися на роботі з об'єктами.

5.2.4 .NET Core MailKit

MailKit — це бібліотека для роботи з електронною поштою у .NET, яка підтримує протоколи IMAP, POP3 та SMTP. MailKit широко використовується для відправки та отримання електронних листів у .NET додатках. Це потужна та безпечна бібліотека, яка підтримує як синхронні, так і асинхронні операції.

Підтримує відправлення електронних листів через протокол SMTP та отримання електронної пошти через протоколи IMAP та POP3.

Підтримує різні методів аутентифікації, зокрема OAuth2. Є можливість додавати до листів вкладення. Підтримує шифрування SSL/TLS для безпечної передачі даних та асинхронні операції для роботи з поштовими серверами.

5.2.5 RabbitMQ

RabbitMQ — це система черг повідомлень (message broker) з відкритим вихідним кодом, яка реалізує протокол AMQP (Advanced Message Queuing Protocol). RabbitMQ використовується для організації асинхронної обробки даних у додатках, що дозволяє їм взаємодіяти через черги повідомлень без прямої взаємодії між собою.

Основні компоненти RabbitMQ:

- Producer (Відправник): Це компонент, який створює і надсилає повідомлення до черги.
- Queue (Черга): Це буфер, у якому зберігаються повідомлення, поки їх не буде забрано споживачем. Кожне повідомлення знаходиться в черзі до моменту доставки одному з споживачів.
- Consumer (Споживач): Компонент, який отримує повідомлення з черги для обробки.
- Exchange (Обмінник): Компонент, який приймає повідомлення від відправника і вирішує, в яку чергу відправити це повідомлення. Існують різні типи обмінників

Відправник (Producer) формує повідомлення і надсилає його в обмінник (Exchange). Обмінник вирішує, до якої черги доставити повідомлення, використовуючи ключі маршрутизації або заголовки. Споживач (Consumer) підписується на певні черги і отримує повідомлення для подальшої обробки.

5.3 Засоби збереження даних

Найпопулярнішою формою баз даних сьогодні є реляційна база даних, хоча розрізняють і багато інших видів баз даних та сховищ даних. Основною мовою для баз даних є мова запитів SQL. Найпопулярнішими серед реляційних баз даних є такі як MySQL, PostgreSQL та MS SQL Server.

5.3.1 Бази даних MySQL

MySQL є найпоширенішою реляційною СКБД серед розробників. Вона є безкоштовною, якщо ми говоримо про базовий набор інструментів, та має більш-менш просту та зрозумілу документацію. Оскільки це реляційна СКБД, то з гарантіями безпеки тут теж все добре. Також MySql ідеально підходить для будь-яких невеликих веб-додатків, оскільки мова PHP найкраще працює з даною СКБД. Недоліками є погана горизонтальна масштабованість, та ліцензування коду приватною компанією Oracle. СКБД має наступні характеристики:

- Можна самостійно змінити вихідний код. Не доведеться нічого платити до того, якщо це не комерційна версія;
- Система підтримує багато моделей кластерних серверів. Пропонується оптимальна швидкість як для обробки великої аналітики, так і для ведення електронної комерції;
- Для розробників є великі ресурси. Підтримується промисловий стандарт. Користувачі досить швидко отримують замовлене програмне забезпечення;
- Система доступу до облікових записів дозволяє гарантувати збереження баз даних та високий клас їх безпеки. Доступне шифрування пароля та перевірка на основі хоста.

5.3.2 Бази даних PostgreSQL

PostgreSQL – це вільна і відкрита система керування базами даних (СКБД) з відкритим вихідним кодом. Є однією з найпопулярніших СКБД, тому її використовують тисячі організацій по всьому світу. PostgreSQL базується на мові SQL. PostgreSQL належить до наступного типу СУБД – об’єктно-реляційного (тобто поєднує в собі переваги реляційних баз даних і об’єктно-орієнтованого програмування).

PostgreSQL підтримує широкий спектр функцій, які роблять її потужною і гнучкою СУБД. Ось деякі з її ключових особливостей:

- PostgreSQL забезпечує транзакції з ACID-властивостями, що гарантує цілісність даних;
- Підтримує автоматично оновлювані подання, що забезпечує можливість користувачам переглядати дані в базі даних у режимі реального часу.
- Підтримує матеріалізовані подання, які зберігають копію даних подання в базі даних;
- Підтримує тригери, які дають змогу виконувати дії під час зміни даних у таблиці;
- Підтримує зовнішні ключі для створення зв’язку між таблицями;
- Підтримує збережені процедури для виконання складних обчислень на стороні сервера.

5.3.3 Бази даних MS SQL Server

MS SQL Server — це реляційна система керування базами даних (РСКБД), розроблена компанією Microsoft. Вона використовується для зберігання, управління і запиту даних в додатках, де критичне значення мають продуктивність, масштабованість та безпека. MS SQL Server забезпечує надійну і продуктивну роботу з даними.

Ключові особливості MS SQL Server:

- MS SQL Server підтримує реляційну модель даних, що дозволяє зберігати інформацію в таблицях і встановлювати зв’язки між ними. Це спрощує організацію даних, запити та аналітику;
- MS SQL Server використовує розширену версію стандартної мови SQL, що називається Transact-SQL (T-SQL). Ця мова дозволяє виконувати складні операції, включаючи транзакції, роботу з курсорами та тригерами;

- MS SQL Server забезпечує багаторівневу безпеку. Він підтримує аутентифікацію на основі ролей, шифрування даних та захист від атак;
- SQL Server підтримує механізми реплікації даних, а також функції високої доступності, що дозволяє гарантувати безперервність роботи бази навіть при аварійних ситуаціях;
- MS SQL Server легко інтегрується з хмарними платформами, такими як Microsoft Azure. За допомогою Azure SQL можна керувати базами даних у хмарі, використовуючи всі переваги масштабованості та безпеки хмарних технологій.

5.3.4 Обраний засіб збереження даних

MS SQL Server був обраний серед можливих баз даних завдяки своїм численним перевагам, що забезпечують надійну роботу, масштабованість та безпеку на рівні підприємств. Він забезпечує високу продуктивність і масштабованість, що дозволяє ефективно працювати з великими обсягами даних та високими навантаженнями.

По-друге, SQL Server пропонує багаторівневу безпеку, включаючи шифрування даних, контроль доступу на основі ролей та захист від SQL-ін'єкцій, що забезпечує безпечне зберігання та управління даними в середовищах з високими вимогами до конфіденційності.

Третім важливим фактором є зручні інструменти для адміністрування та моніторингу. SQL Server Management Studio (SSMS) полегшує управління базами даних, дозволяючи моніторити продуктивність, виконувати запити та автоматизувати рутинні завдання адміністрування.

Нарешті, широка підтримка реляційної моделі та стандарту SQL з розширенням T-SQL робить MS SQL Server потужним інструментом для складних запитів, транзакцій та управління даними в критичних для бізнесу додатках. Усі ці фактори разом роблять MS SQL Server ідеальним вибором для системи, що вимагає високої надійності, безпеки, гнучкості та продуктивності.

5.4 Технології розробки клієнтської частини системи

Згідно статистики 2024-року популярного ресурсу DOU.ua, найбільш популярними технологіями для розробки клієнтської частини інформаційної системи станом на сьогодні є такі фреймворки як React, Angular та Vue.

5.4.1 Фреймворк React

Створений як проект з відкритим кодом і досі використовуваний Facebook, React - це популярний фреймворк JS, який орієнтований на досвід користувачів. Що робить React унікальним, так це те, що він може бути відтворений як на сервері, так і на стороні клієнта. Залежно від вимог до захисту даних, певні компоненти можуть відображатися на сервері, а інші - на клієнті.

React або «ReactJS» являє собою бібліотеку для значного спрощення побудови й маніпулювання DOM елементами. Творцем цієї бібліотеки виступає компанія Facebook. Спочатку її розробляли й використовували лише для внутрішніх потреб (тобто безпосереднього написання своєї соцмережі), й лише згодом зробили публічною, надавши до неї доступ широкому загалу програмістів.

У своєму чистому вигляді React і справді є нічим більшим, аніж звичайнісінькою бібліотекою для написання простих фронтендних додатків (при цьому, навіть не Single Page Application), оскільки не містить ані Routing, ані інших додаткових можливостей. Лише відносно нещодавно отримав офіційну можливість використовувати Context API для реалізації доволі простенького state management. Щоб користуватись React, достатньо додати посилання на бібліотеку у `index.html` сторінку.

Даний фреймворк має наступні **переваги**:

- Багаторазові компоненти React гарантують, що розробникам не доведеться переписувати один і той же код знову і знову;

- Завдяки своїй популярності в мережі Інтернет доступна величезна кількість безкоштовних допомог від однолітків-розробників.

Недоліками React є:

- Посилений фокус React на розробці інтерфейсу може зробити інші аспекти розвитку складними;
- Крива навчання для цієї основи висока, частково через невідповідну проектну документацію.

Ідеально підходить для: тих, хто має досвід розробки та хоче створити вебсайт або мобільний додаток із розширеним інтерфейсом.

5.4.2 Фреймворк Angular

Флагманський JS фреймворк від Google, Angular, розробляється вже досить давно. Хоча це не найпростіший фреймворк для вивчення, крута крива навчання може в кінцевому підсумку вартувати затраченого на вивчення часу. Це чудово підходить для проектів, що потребують команди, які змінюються з часом, оскільки спосіб інкапсуляції компонентів робить його модульним та легким для розуміння новим розробникам.

На відміну від React, Angular, що є продуктом компанії Google, вже є не просто бібліотекою для роботи з DOM, а цілим повноцінним фреймворком, котрий керується принципами MVVM побудови застосунків.

Історично, Angular хоча й виник трішки пізніше за React, фактично є переписаною з «нуля» версією фреймворку AngularJS, котрий існував раніше за нього. На початку серед фронтендщиків навіть виникали деякі непорозуміння, оскільки ніхто не розумів різниці між назвами Angular і AngularJS. Як наслідок, команда розробки фреймворку почала використовувати назви Angular першої версії (для AngularJS) і Angular другої версії (для Angular), аби чіткіше їх розрізняти.Якщо порівняти Angular з React, то його структура є значно

складнішою. Окрім компонентів, тут також з’являються наступні елементи, кожен зі своїм призначенням:

1. пайпи — свого роду допоміжні функції для трансформації візуального вигляду даних у HTML темплейтах. Наприклад: форматування чисел тощо;

2. директиви — класи, котрі використовуються для надання додаткового функціонала іншим елементам застосунку. І також поділяються на кілька видів, в залежності від форми взаємодії з елементами (цей момент я пропущу, аби не вдаватись в подробиці). В основному застосовуються у вигляді користувацьких атрибутів елементів розмітки;

3. компоненти — є елементом для візуального зображення даних. Поєднують в собі стилі, HTML-файл з розміткою, а також JS/TS з логікою;

4. сервіси — для можливості централізованого зберігання даних і утилітарних методів. Можуть використовуватись пайпами, директивами, компонентами й навіть іншими сервісами;

5. модулі — спосіб згрупувати всі вищеперелічені елементи. В більшості випадків, по функціоналу або ж сторінках.

Переваги фреймворку Angular:

- Надзвичайно складні веб-програми можна розробити в enterprise масштабах, і скласти конкуренцію програмам для настільних ПК;
- Оскільки Google гарантує довгострокову підтримку цього проекту з відкритим кодом, розробники можуть бути впевнені, що його не покинуть найближчим часом.

Недоліки фреймворку:

- Angular дуже складний і має одну з найкрутіших кривих навчання;
- Налагодження може бути проблематичним, оскільки йому бракує інструментів калібру деяких його конкурентів.

Ідеально підходить для: досвідчених розробників та інженерів, що роблять корпоративні програми, які потребують максимальної гнучкості та готові витратити час на навчання.

5.4.3 Фреймворк Vue

Vue.js був розроблений як альтернатива Angular та React. Він був створений як мінімалістична версія Angular, але з роками він значно зростав. Спочатку він використовувався для невеликих проектів, що ведуться розробниками, і зараз став повноцінним фреймворком.

Використовуючи традиційні HTML, CSS та JS, розробники можуть створювати компоненти, як і інші популярні фреймворки, такі як React. Що відрізняє Vue.js – це двостороння підтримка прив'язки даних.

Vue є свого роду сумішшю Angular і React. З одного боку, у своєму чистому вигляді — це бібліотека для роботи з DOM, та з іншого боку — це фреймворк, в якого навіть є свій Vue CLI й певна екосистема бібліотек навколо нього.

Vue широко використовує й наполягає на підході Single File Component, щодо написання компонентів, як основному. Його суть полягає в тому, що стилі, розмітка і логіка компонента мають знаходитись в одному файлі. Хоча вона й надає можливість і засоби для розбиття компонента на кілька різних файлів.

Фреймворк має такі **переваги**:

- Відмінна документація і призначена як для початківців, так і для більш досвідчених розробників;
- Широка інструментальна екосистема, яка зросла за ці роки;
- Фреймворк невеликий за розміром і простий у вивченні.

Даний фреймворк має і свої **недоліки**:

- Існує ризик надмірної гнучкості, оскільки занадто багато варіантів може призвести до різних підходів до програмування;
- Враховуючи його вік, Vue отримує менше підтримки, ніж конкуренти;
- Досить низька продуктивність через недостатню оптимізацію рендеру.

Ідеально підходить для: тих, хто має мінімальний досвід веб-розробки, якому потрібно швидко створювати прототипи.

5.4.4 Обрана технологія для клієнтської частини

Для клієнтської частини обрано фреймворк Angular. По-перше, це фреймворк з чіткою архітектурою, що дозволяє структурувати код і легко масштабувати додатки. Завдяки модульності можна створювати незалежні компоненти, що значно спрощує командну роботу.

По-друге, Angular підтримує двостороннє зв’язування даних, що економить час при розробці, оскільки зміни в моделі автоматично відображаються в інтерфейсі і навпаки. Це дуже зручно, особливо при роботі з формами.

В Angular вбудовані інструменти для тестування, які дозволяють легко перевіряти код на різних етапах розробки. Це допомагає підтримувати якість коду і запобігати багам.

Крім того, фреймворк має гарну документацію та активну спільноту, що полегшує процес навчання та отримання допомоги в разі виникнення проблем. Важливо й те, що Angular активно підтримується Google, що підвищує довіру до його стабільності та майбутнього розвитку.

Загалом, Angular підходить для великих і складних проектів, де важлива організація коду та можливість легко масштабувати додаток.

6. РЕАЛІЗАЦІЯ СИСТЕМИ

В даному розділі описано всі ключові положення щодо власне програмної реалізації інформаційної системи ВНЗ, а саме компонента складання розкладів. Із застосуванням засобів та технологій, що описані в розділі 5, описано основні етапи реалізації, серед яких розробка серверної частини (тришарова архітектура, бази даних тощо) та клієнтської частини (комунікація із сервером, маршрутизація тощо).

6.1 Архітектура системи на основі компонентів системи

В межах архітектури системи на базі IT-інфраструктури для інформаційної системи ВНЗ повинно бути визначено основні компоненти системи, підхід до розгортання системи та її елементів та організацію в цілому.

Розгортання (Deployment): система розгорнута в хмарному середовищі AWS (Amazon Web Services), що забезпечує гнучкість, масштабованість та високу доступність. Для безпечного розгортання передбачено використання AWS Secrets Manager — керованого сервісу для безпечного зберігання, управління та доступу до конфіденційної інформації, такої як паролі, ключі API, облікові дані баз даних та інші секрети.

Завдяки ньому можна автоматично змінювати паролі баз даних або інші облікові дані, причому доступ до секретів контролюється через політики AWS IAM, а також всі секрети шифруються за допомогою AWS KMS. Тому вибір є очевидним.

Обмін повідомленнями: для комунікації між клієнтським додатком (побудованим на основі Angular) та мікросервісами використовується брокер повідомлень RabbitMQ. Він забезпечує асинхронний обмін даними між компонентами системи.

Мікросервіси: система складається з наступних мікросервісів, кожен з яких відповідає за конкретну функціональність:

- Faculties (Факультети): взаємодіє зі спеціальностями для управління структурою факультетів;
- Specialties (Спеціальності): обробляє інформацію про спеціальності та взаємодіє з сервісом дисциплін;
- Groups (Групи): зберігає та керує інформацією про академічні групи;
- Disciplines (Дисципліни): взаємодіє з сервісом викладачів для управління навчальними дисциплінами;
- Teachers (Викладачі): керує інформацією про викладачів;
- Groups (Групи): зберігає та керує інформацією про академічні групи.
- Schedules (Розклади): керує розкладом занять, взаємодіючи зі службами груп та викладачів;
- Tokens (Токени): обробляє аутентифікацію користувачів та керування токенами доступу;
- Emailer: надсилає електронні листи за допомогою SmtpClient для взаємодії із зовнішніми поштовими сервісами.

Логування та бази даних: для кожного мікросервісу передбачено власну систему логування для моніторингу та діагностики, а також використовує окрему базу даних на основі RDS (Relational Database Service), що забезпечує незалежність та масштабованість кожного компонента системи.

При розширенні системи, можливі додаткові ресурси (сервіси):

- *AWS Lambda* – безсерверний обчислювальний сервіс, що дозволяє виконувати код у відповідь на події, без необхідності управління серверами;
- *Amazon S3* – сервіс для зберігання великих обсягів неструктурованих даних, таких як мультимедіа, резервні копії, лог-файли;
- *Amazon EC2* — це сервіс від, який надає масштабовані віртуальні сервери для запуску додатків.

На рисунку 6.2 зображено діаграму архітектури типової інформаційної системи ВНЗ на базі IT-інфраструктури та її компонент.

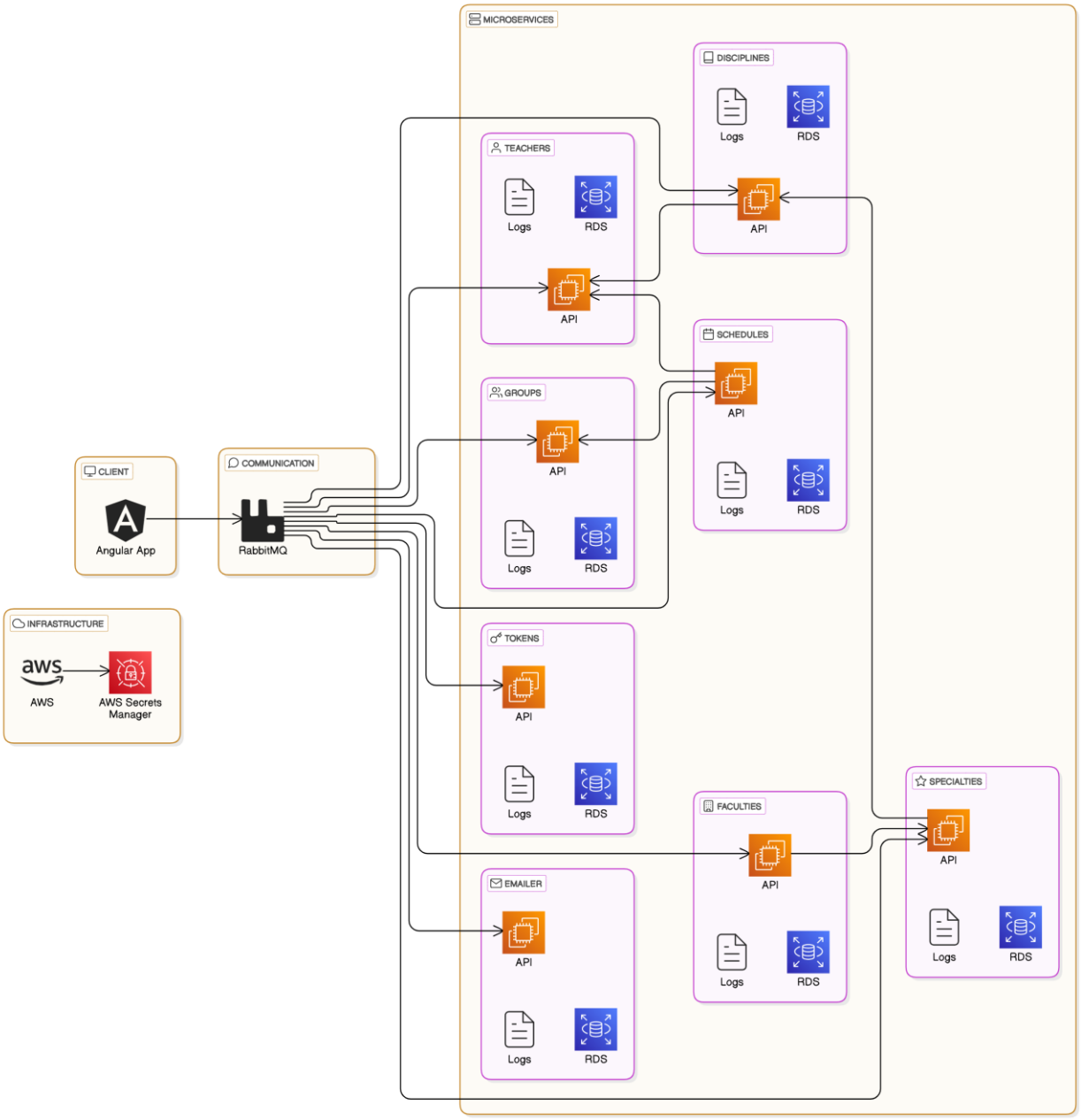

Рисунок 6.2 – Архітектура системи ВНЗ на базі IT-інфраструктури

Переваги такої архітектури:

- Модульність: кожен мікросервіс є незалежним, що полегшує обслуговування та оновлення;
- Масштабованість: можливість незалежного масштабування;

- Надійність: асинхронний обмін повідомленнями знижує залежність між компонентами та підвищує надійність системи;
- Хмарна інфраструктура: AWS надає широкий спектр інструментів для керування, моніторингу та забезпечення безперебійної роботи системи.

Завдяки використанню мікросервісної архітектури та хмарної платформи AWS, кожен компонент можна масштабувати окремо залежно від навантаження.

Наприклад, якщо сервіс Schedules (Розклади) під час навчального року потребує більше ресурсів, його можна збільшити без необхідності масштабувати інші сервіси, що дозволяє оптимізувати витрати на інфраструктуру та забезпечити стабільну роботу системи навіть при високому навантаженні.

6.2 Специфікація обладнання

Специфікація обладнання – це документ, що містить детальний опис технічних параметрів, функціональних можливостей та вимог до апаратного забезпечення, що необхідне для впровадження та експлуатації інформаційної системи. Говорячи про інформаційну систему для ВНЗ, специфікація обладнання має враховувати потреби в серверному та мережевому обладнанні.

1. Серверне обладнання – центральна частина інфраструктури, що забезпечує обробку, зберігання і захист даних.

Сервери обробки даних:

- Процесор: Intel Xeon або AMD EPYC (не менше 8 ядер);
- Оперативна пам'ять: 32–128 ГБ DDR4/DDR5;
- Накопичувач: SSD NVMe від 1 ТБ (для обробки) + HDD від 4 ТБ (для архівування);
- Підтримка віртуалізації (наприклад, VMware, Hyper-V).

Сервери баз даних:

- Процесор: Intel Xeon Gold/Platinum (12+ ядер);

- Оперативна пам'ять: 64–256 ГБ;
- Накопичувач: SSD з високою швидкістю запису (для транзакцій баз);
- RAID-контролери для забезпечення відмовостійкості.

Сервер резервного копіювання:

- Накопичувач: HDD великого об’єму (8–12 ТБ) для резервних копій;
- Підтримка технологій резервного копіювання (напр.,Veeam Backup).

2. Мережеве обладнання – необхідне для підключення користувачів, забезпечення доступу до ресурсів і захисту мережі.

Комутатори (Switches):

- Рівень: L2/L3;
- Швидкість портів: 1 Гбіт/с для користувацьких з'єднань;
- Кількість портів: 24–48, залежно від масштабу мережі.

Маршрутизатори (Routers):

- Пропускна здатність: від 1 Гбіт/с;
- Підтримка VPN, QoS, статичної та динамічної маршрутизації.

Точки доступу Wi-Fi:

- Стандарт: Wi-Fi 6 (802.11ax) для високої пропускної здатності;
- Покриття: 100–150 користувачів на точку доступу.

Мережевий екран (Firewall):

- Захист: від DDoS-атак, фільтрація трафіку, контроль доступу;
- Пропускна здатність: 1–5 Гбіт/с.

6.3 Реалізація серверної частини системи

Ключовими етапами реалізації серверної частини компоненти складання розкладу інформаційної системи ВНЗ є проєктування бази даних та її нормалізація, конфігурація (API та розробницькі налаштування) та безпосередньо розробка системи відповідно до обраного архітектурного стилю.

Для системи складання розкладу ВНЗ) серверна частина виступає як центральний елемент, який відповідає за зберігання, обробку та надання інформації користувачам, а також за інтеграцію з іншими компонентами.

6.3.1. Проєктування бази даних

Правильно спроєктована база даних — це не просто набір таблиць, а продумана структура, яка відображає логіку бізнесу та дозволяє працювати з даними ефективно і без помилок. Важливо при проєктуванні структури сховища дотримуватись нормальних форм та мати нормалізацію хоча б на рівні третьої нормальної форми. Компонента складання розкладів має так, як і цілісна система ВНЗ, мікросервісну архітектуру. А в разі проєктування мікросервісної системи, хорошим тоном є розділення баз даних, тобто кожен мікросервіс має свою базу даних, що дозволяє підтримати основну мету даної архітектури – відмовостійкість. Інакше при проблемах з єдиною базою даних для всіх мікросервісів, ніякої користі від кількості підсистем ми не отримаємо.

В даній компонент кожен мікросервіс відповідає за свою сутність, тобто з'єднується зі свою базою даних, що має таблиці лише щодо відповідної сутності.

База даних мікросервісу, що відповідає за користувачів, навчальні заклади та процес авторизації, містить 5 таблиць, що пов'язують користувачів, їх дані для авторизації (включно з роллю) та навчальні заклади (далі – EI, тобто Education Institution). Врахуємо, що UniversityId представляє те ж саме, що і EIId.

Таблиця 6.1 – Опис таблиць бази даних користувачів та закладів

Назва таблиці	**Назва поля**	**Опис поля**
EIs (Навч. заклади)	Id	Унікальний ідентифікатор
	Name	Назва навчального закладу
	Link	Посилання на розклади

Продовження таблиці 6.1 – Опис таблиць бази даних користувачів та закладів

Roles (Ролі)	Id	Унікальний ідентифікатор
	RoleName	Назва ролі
Credentials (Дані про користувача)	Id	Унікальний ідентифікатор
	UserId	Ідентифікатор користувача
	RoleId	Ідентифікатор ролі
	PasswordHash	Хеш паролю
	PasswordSalt	Кодування паролю
Users (Користувачі)	Id	Унікальний ідентифікатор
	Email	Електронна пошта
	RegistrationTime	Час реєстрації
	CredentatialsId	Ідентифікатор даних про користувача
UserEIs (Навчальні заклади користувачів)	Id	Унікальний ідентифікатор
	UserId	Ідентифікатор користувача
	EIId	Ідентифакатор навч. закладу
	IsAdmin	Користувач є адміністратором – створив навчальний заклад
	IsAccepted	Користувач прийняв запрошення з правами доступу до закладу
	IsAnswered	Користувач відреагував на запрошення з правами доступу до закладу
	UserEmailWhoSendInvite	Електронна пошта користувача, що запросив до навчального закладу

На рисунку 6.2 зображено діаграму бази даних, таблиці якої описані вище.

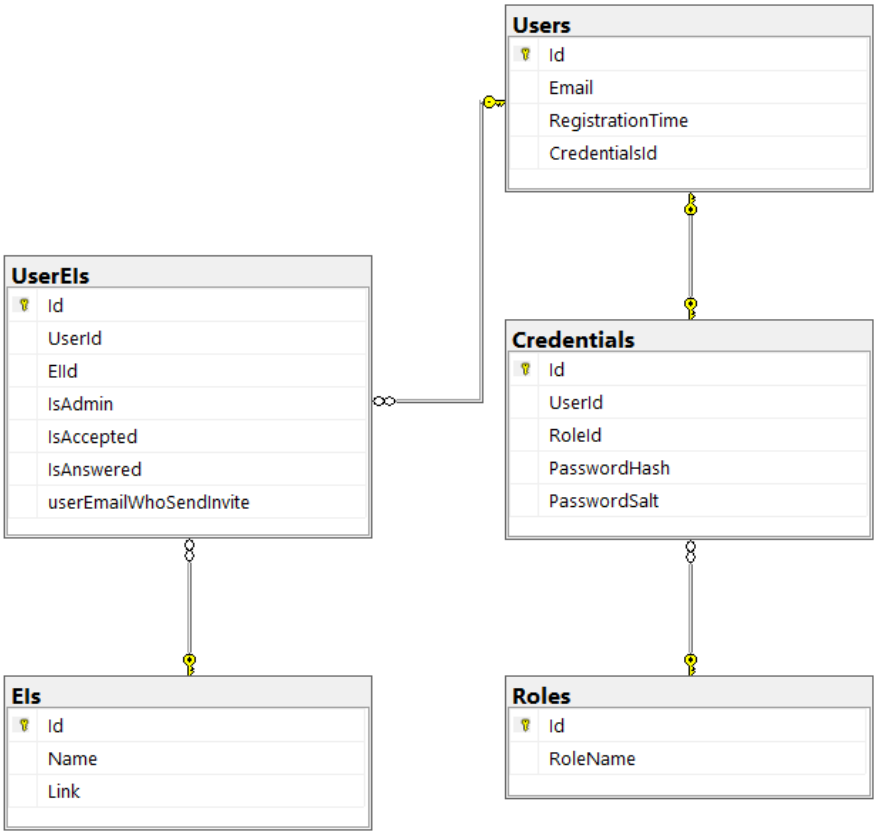

Рисунок 6.2 – Діаграма бази даних з користувачами та навчальними закладами

База даних мікросервісу, що відповідає за факультети, містить єдину таблицю, що описує факультети навчальних закладів.

Таблиця 6.2 – Опис таблиць бази даних факультетів

Назва таблиці	**Назва поля**	**Опис поля**
Faculties (Факультети)	Id	Унікальний ідентифікатор
	Name	Назва факультету
	Description	Опис факультету
	EIId	Ідентифікатор навч. закладу

База даних мікросервісу, що відповідає за спеціальності, містить таблицю зі спеціальностями та асоціативну таблицю для спеціальностей та факультетів.

Таблиця 6.3 – Опис таблиць бази даних спеціальностей

Назва таблиці	**Назва поля**	**Опис поля**
Specialties (Спеціальності)	Id	Унікальний ідентифікатор
	Cipher	Шифр спеціальності
	Name	Назва спеціальності
	Description	Опис спеціальності
	EIId	Ідентифікатор навчального закладу
FacultySpecialties (Спеціальності факультетів)	Id	Унікальний ідентифікатор
	FacultyId	Ідентифікатор факультету
	SpecialtyId	Ідентифікатор спеціальності

База даних мікросервісу, що відповідає за дисципліни, містить 4 таблиці, що пов’язують дисципліни, каталоги дисциплін та їх зв’язок із спеціальностями.

Таблиця 6.4 – Опис таблиць бази даних дисциплін та каталогів

Назва таблиці	**Назва поля**	**Опис поля**
Disciplines (Дисципліни)	Id	Унікальний ідентифікатор
	Name	Назва дисципліни
	Description	Опис дисципліни
	Course	Курс викладання
	CreditType	Тип оцінювання (залік/екзам.)
	Hours	Кількість годин викладання на семестр
	IsSelective	Дисципліна є вибірковою
	EIId	Ідентифікатор навчального закладу

Продовження таблиці 6.4 – Опис таблиць бази даних дисциплін та каталогв

SpecialtyDisciplines (Дисципліни спеціальностей)	Id	Унікальний ідентифікатор
	DisciplineId	Ідентифікатор дисципліни
	SpecialtyId	Ідентифікатор спеціальності
	Semester	Семестер викладання згідно плану спеціальності
Catalogs (Каталоги)	Id	Унікальний ідентифікатор
	Name	Назва каталогу
	EIId	Ідентифікатор навчального закладу
CatalogDisciplines (Дисципліни каталогів)	Id	Унікальний ідентифікатор
	DisciplineId	Ідентифікатор дисципліни
	CatalogId	Ідентифікатор каталогу

На рисунку 6.3 зображено діаграму бази даних, таблиці якої описані вище.

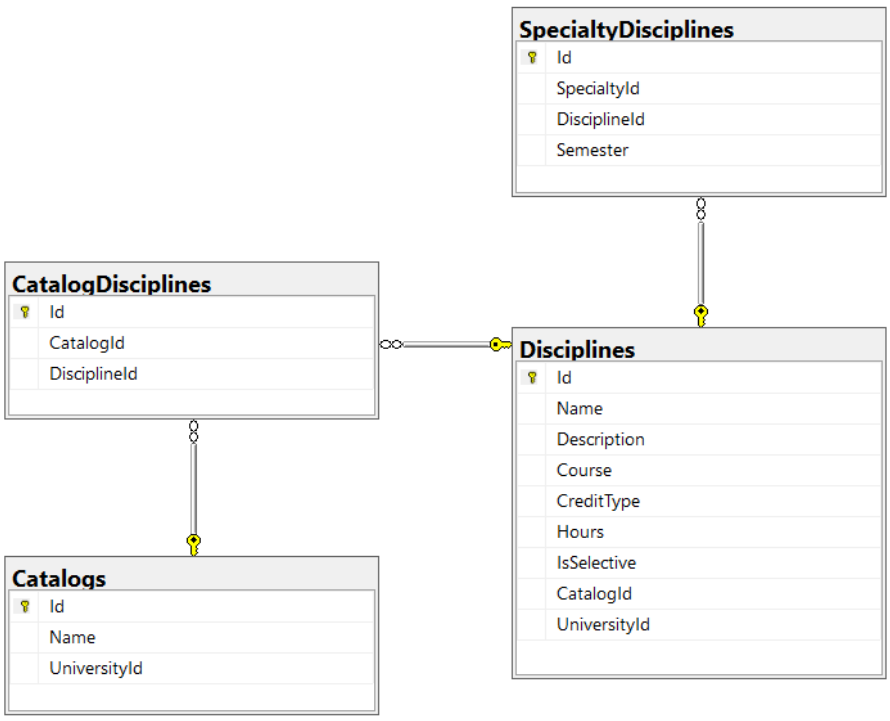

Рисунок 6.3 – Діаграма бази даних з дисциплінами та каталогами

База даних мікросервісу, що відповідає за викладачів, містить таблицю із викладачами та асоціативну таблицю для викладачів та дисциплін.

Таблиця 6.5 – Опис таблиць бази даних викладачів

Назва таблиці	**Назва поля**	**Опис поля**
Teachers (Викладачі)	Id	Унікальний ідентифікатор
	Name	Ім’я викладача
	Surname	Прізвище викладача
	Patronymic	По-батькові викладача
	EIId	Ідентифікатор навчального закладу
DisciplineTeachers (Викладачі дисциплін)	Id	Унікальний ідентифікатор
	DisciplineId	Ідентифікатор дисципліни
	TeacherId	Ідентифікатор викладача
	IsLecturer	Викладач є лектором дисципліни

База даних мікросервісу, що відповідає за групи, містить єдину таблицю, що описує навчальні групи закладів.

Таблиця 6.6 – Опис таблиць бази даних груп

Назва таблиці	**Назва поля**	**Опис поля**
Groups (Групи)	Id	Унікальний ідентифікатор
	Cipher	Шифр (назва) групи
	Course	Поточний курс групи
	FacultyId	Ідентифікатор факультету
	SpecialtyId	Ідентифікатор спеціальності
	EIId	Ідентифікатор навч. закладу

База даних мікросервісу, що відповідає за розклади, містить таблицю із розкладами та асоціативну таблицю для дисциплін розкладів.

Таблиця 6.7 – Опис таблиць бази даних розкладів та їх дисциплін

Назва таблиці	**Назва поля**	**Опис поля**
Schedules (Розклади)	Id	Унікальний ідентифікатор
	GroupId	Ідентифікатор навч. групи
ScheduleDisciplines (Дисципліни розкладів)	Id	Унікальний ідентифікатор
	DisciplineId	Ідентифікатор дисципліни
	TeacherId	Ідентифікатор викладача
	Day	День тижня (слот)
	Semester	Семестр
	Lesson	Номер пари (слот)
	IsLecture	Пара є лекцією
	IsSelective	Дисципліна пари є вибірковим
	ScheduleId	Ідентифікатор розкладу

База даних мікросервісу, що відповідає за електронну пошту (Emailer), містить єдину таблицю, що описує підписки електронною поштою.

Таблиця 6.8 – Опис таблиць бази даних для електронної пошти

Назва таблиці	**Назва поля**	**Опис поля**
EmailSubscriptions (Підписки електронною поштою)	Id	Унікальний ідентифікатор
	Email	Адреса електронної пошти
	Semester	Семестр підписки
	ScheduleId	Ідентифікатор розкладу

6.3.2. Конфігурація серверної частини та розробницькі налаштування

Для зручності розробки системи важливо правильно налаштувати проєкт, зокрема, різні середовища (зазвичай, це Local, Development та Production), профайл-файли маперів (відображень) моделей та основні launch-файли для запуску й розгортання інформаційної системи.

На рисунку 6.4.1 зображена конфігурація для одного з мікросервісів, де основними є налаштування логування, з'єднання з системою управління базами даних, а також під'єднання до посередника RabbitMQ забезпечення обміну повідомленнями-подіями для комунікації між окреамими мікросервісами.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "Auth": {
    "Issuer": "https://localhost",
    "Audience": "https://localhost",
    "Secret": "superSecretKeyGDegegegsdgseeqrwreweeeeee22222442rgsdeewdsfsdf342",
    "TokenLifeTime": "3600"
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "TeacherDb": "Server=localhost;Database=TeachersDb;Trusted_Connection=True;Encrypt=False;"
  }
}
```

Рисунок 6.4.1 – Основна конфігурація окремого мікросервіса

На рисунку 6.4.2 зображена конфігурація мапінгу (відображення) об'єктів.

```
namespace TeacherService
{
    public class AutomapperProfile : Profile
    {
        public AutomapperProfile()
        {
            CreateMap<Teacher, TeacherModel>()
                .ReverseMap();

            CreateMap<DisciplineTeacher, DisciplineTeacherModel>()
                .ReverseMap();
        }
    }
}
```

Рисунок 6.4.2 – Основна конфігурація мапінгу окремого мікросервіса

При розробці систем на базі платформи .NET, точкою входу є файл Program.cs, що і виступає launch-файлом та повинен бути якісно описаний для правильної роботи та зручного розгортання сервісу. В ньому описується аутентифікація, контролери, Swagger, ін’єкції залежностей, Cors-доступи. На рисунках 6.4.3 – 6.4.5 зображено файл Program.cs для сервісу з викладачами. На рисунку 6.4.3 продемострovано конфігурацію аутентифікації та контролерів.

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

var authOptions = builder.Configuration.GetSection("Auth");
builder.Services.Configure<JwtOptions>(authOptions);
var auth = authOptions.Get<JwtOptions>();

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme).AddJwtBearer(options =>
{
    options.RequireHttpsMetadata = false;
    options.TokenValidationParameters = new Microsoft.IdentityModel.Tokens.TokenValidationParameters
    {

        ValidateIssuer = true,
        ValidIssuer = auth?.Issuer,

        ValidateAudience = false,
        ValidAudience = auth?.Audience,

        ValidateLifetime = true,

        IssuerSigningKey = auth?.GetSymmetricSecurityKey(),
        ValidateIssuerSigningKey = true,

    };
});

builder.Services.AddControllers();
// Learn more about configuring Swagger/cdOpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddTransient<ITeacherRepository, TeacherDbRepository>();
```

Рисунок 6.4.3 – Основна конфігурація мапінгу окремого мікросервіса – 1

Конкретніше, тут використано механізм **Dependency Injection** (DI). В ASP.NET Core — це вбудований механізм впровадження залежностей, який дозволяє керувати створенням об'єктів і передачею їх в інші класи або сервіси. DI забезпечує розв’язання проблем залежностей між компонентами, спрощує тестування та поліпшує гнучкість коду.

Основними типами впровадження залежностей (DI) є наступні:

- Transient: об'єкт створюється щоразу, коли викликається. Використовується для легких сервісів, які не зберігають стан;

- Scoped: об'єкт створюється один раз на кожен HTTP-запит. Використовується для сервісів, які мають зберігати дані протягом усього запиту;
- Singleton: об'єкт створюється один раз на весь життєвий цикл програми. Використовується для сервісів, які мають стан, що розділяється між запитами.

На рисунку 6.4.4 продемостровано конфігурацію HTTP-клієнту та відмовостійкості. На цьому фрагменті коду показано конфігурацію HttpClient у файлі Program.cs для ASP.NET Core застосунку, де додано HttpClient для роботи з віддаленим API та зареєстровано клієнт для взаємодії з API, який надає сервіси для роботи з даними викладачів. BaseAddress в становлює базову адресу, яка використовується для запитів до API (тут – необхідний зв’язок з мікросервісом «дисципліни».

Також тут оголошено політику повторних спроб та стійкості. AddTransientHttpErrorPolicy реалізує механізми обробки помилок для запитів, а політика повторних спроб описана через WaitAndRetryAsync, де при виникненні помилки, наприклад, через тайм-аут (TimeoutRejectedException), запит буде повторюватися. Визначано експоненційне збільшення часу між повторними спробами.

Окрім цього, описано політику "Circuit Breaker": якщо кількість помилок перевищує поріг (3 помилки), клієнт тимчасово припиняє відправляти запити протягом 15 секунд, щоб уникнути перевантаження. TimeoutAsync встановлює політику для тайм-аутів при обробці HTTP-відповідей.

```
builder.Services.AddHttpClient<ITeacherService, Business.Service.TeacherService>(a =>
{
    a.BaseAddress = new Uri("https://localhost:7215/api/disciplines/");
})
.AddTransientHttpErrorPolicy(b => b.Or<TimeoutRejectedException>().WaitAndRetryAsync(
    5,
    c => TimeSpan.FromSeconds(Math.Pow(2, c))
))
.AddTransientHttpErrorPolicy(b => b.Or<TimeoutRejectedException>().CircuitBreakerAsync(
    3,
    TimeSpan.FromSeconds(15)
    ))
.AddPolicyHandler(Policy.TimeoutAsync<HttpResponseMessage>(1));

builder.Services.AddTransient<IDisciplineTeacherRepository, DisciplineTeacherDbRepository>();
builder.Services.AddTransient<IDisciplineTeacherService, DisciplineTeacherService>();

builder.Services.AddCors();
```

Рисунок 6.4.4 – Основна конфігурація мапінгу окремого мікросервіса – 1

На рисунку 6.4.5 продемостровано конфігурацію з’єднання з базою даних, мапінгу, HTTP-з’єднань, міграцій тощо.

```
var teacherConnectionString = builder.Configuration.GetConnectionString("TeacherDb");
builder.Services.AddDbContext<TeacherDbContext>(x => x.UseSqlServer(teacherConnectionString));
builder.Services.AddTransient<TeacherDbContext>();

var mapperConfig = new MapperConfiguration(mc =>
{
    mc.AddProfile(new AutomapperProfile());
});

IMapper mapper = mapperConfig.CreateMapper();
builder.Services.AddSingleton(mapper);

var app = builder.Build();
app.ApplyMigrations();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseCors(builder =>
{
    builder
    .AllowAnyOrigin()
    .AllowAnyMethod()
    .AllowAnyHeader();
});

app.UseHttpsRedirection();
app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();
app.Run();
```

Рисунок 6.4.5 – Основна конфігурація мапінгу окремого мікросервіса – 3

Окрім всього перерахованого, для зручності створено додаткове налаштування для зручного автоматичного застосування міграцій бази даних.

```
public static class MigrationExtensions
{
    public static void ApplyMigrations(this WebApplication app)
    {
        using var scope = app.Services.CreateScope();
        var dbContext = scope.ServiceProvider.GetRequiredService<TeacherDbContext>();

        if (dbContext is not null)
        {
            try
            {
                dbContext.Database.Migrate();
            }
            catch (Exception ex)
            {
                app.Logger.LogError($"Could not apply migrations. {ex.Message}");
                throw;
            }
        }
    }
}
```

Рисунок 6.4.6 – Допоміжний метод автоматичних міграцій

6.3.3. Розробка серверної частини

Основою серверної частини компоненти складання розкладів інформаційної системи ВНЗ є трирівнева архітектура. Дана архітектура – це найпоширеніша архітектура взаємодії. Вона з'явилася, коли серверну частину дворівневої архітектури розділили на дві частини: шар логіки та шар даних.

Шар клієнта – це частина «розподіленої програми», яка відповідає за взаємодію з користувачем. Цей шар не повинен містити бізнес-логіку та зберігати критично важливі дані. Також він не повинен взаємодіяти з шаром бази даних безпосередньо, а лише через бізнес-логіку.

Шар бізнес-логіки розташовується на другому рівні: на ньому зосереджена більша частина бізнес-логіки. За його межами залишаються лише фрагменти, що експортуються на клієнта, а також елементи логіки, занурені в базу даних (збережені процедури та тригери).

Шар даних забезпечує зберігання даних і виноситься на окремий рівень, реалізується, зазвичай, засобами систем управління базами даних, підключення до цього компоненту забезпечується лише з рівня сервера застосунків.

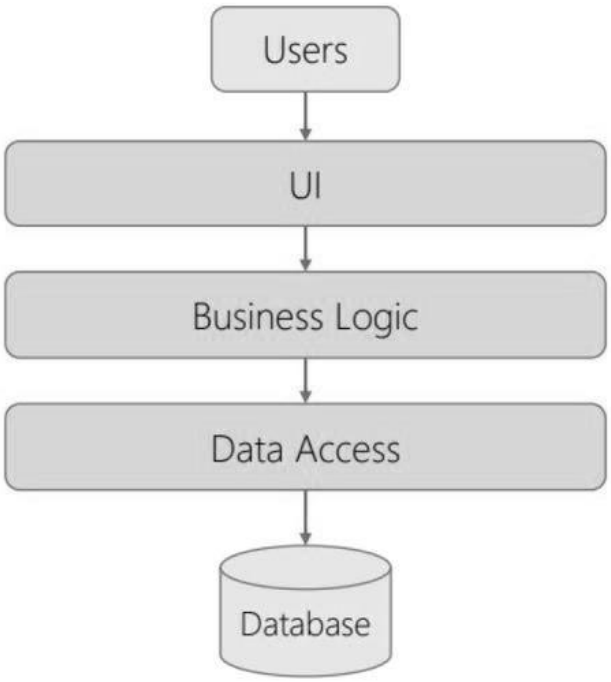

Рисунок 6.5.1 – Модель трирівневої архітектури

Так, кожен мікросервіс цілої системи поділений на три рівні, серед яких рівень даних, рівень бізнес-логіки та рівень клієнта (API). Структура мікросервіса виглядає так, як зображено на рисунку 6.5.2.

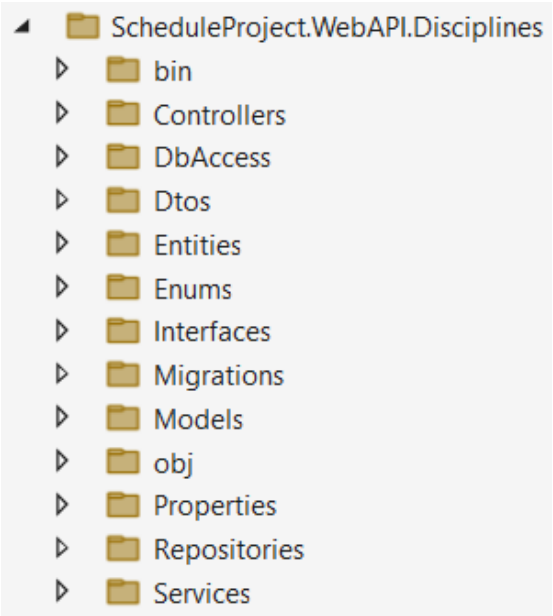

Рисунок 6.5.2 – Організація коду мікросервіса

Всі необхідні інтерфейси для репозиторіїв (шар доступу до даних) та сервісів (шар бізнес-логіки) оголошено в папці Interfaces. В даних класах оголошуються лише «заглушки» методів, що потрібно реалізувати – їх семантична основа, тобто назва, вхідні та вихідні параметри. Код інтерфейсів зображено на рисунках 6.5.3 та 6.5.4.

```
namespace DisciplineService.Interfaces
{
    public interface ICatalogService
    {
        Task<CatalogModel> AddAsync(CatalogModel model);
        Task<IEnumerable<CatalogModel>> GetAllAsync();
        Task<CatalogModel> GetByIdAsync(int id);
        Task DeleteByIdAsync(int modelId);
    }
}
```

Рисунок 6.5.3 – Код інтерфейсу для сервіса каталогів дисциплін

```
namespace DisciplineService.Interfaces
{
    public interface IDisciplineRepository : IRepository<Discipline>
    {
    }
}
```

Рисунок 6.5.4 – Код інтерфейсу для репозиторію дисциплін

З ціллю забезпечення масштабованості шару доступу до даних, конкретні репозиторії (для конкретних сутностей – моделей бази даних) є наслідниками базового спільного інтерфейсу-репозиторію IRepository, що є узагальненим типом (generic) та може бути легко розширений. Код інтерфейсу репозиторія з основними CRUD-операціями над сутністю подано на рисунку 6.5.6.

```
namespace DisciplineService.Interfaces
{
    public interface IRepository<TEntity> where TEntity : BaseEntity
    {
        /// <summary>Gets all asynchronous.</summary>
        /// <returns>
        ///   IEnumerable&lt;TEntity&gt;.
        /// </returns>
        Task<IEnumerable<TEntity>> GetAllAsync();

        /// <summary>
        /// Gets the TEntity by identifier asynchronous.
        /// </summary>
        /// <param name="id">The identifier.</param>
        /// <returns></returns>
        Task<TEntity> GetByIdAsync(int id);

        /// <summary>
        /// Adds the TEntity asynchronous.
        /// </summary>
        /// <param name="entity">The entity.</param>
        /// <returns></returns>
        Task<TEntity> AddAsync(TEntity entity);

        /// <summary>
        /// Updates the specified entity.
        /// </summary>
        /// <param name="entity">The entity.</param>
        void Update(TEntity entity);

        /// <summary>Deletes the by identifier asynchronous.</summary>
        /// <param name="id">The identifier.</param>
        /// <returns>
        /// TEntity
        /// </returns>
        Task<TEntity> DeleteByIdAsync(int id);

        /// <summary>
        /// Saves the asynchronous.
        /// </summary>
        /// <returns></returns>
        Task SaveAsync();
    }
}
```

Рисунок 6.5.6 – Код інтерфейсу базового репозиторія

Шар доступу до даних. В папці Entities зберігаються основні моделі (ентіті) в тому вигляді, що вони зберігаються та отримуються з бази даних. Всі такі моделі є наслідниками базового класу BaseEntity, що оголошує обов’язкове поле – унікальний ідентифікатор Id. Ентіті «дисципліна спеціальності» зображено на рисунку 6.5.7.

```
namespace DisciplineService.Entities
{
    public class SpecialtyDiscipline : BaseEntity
    {
        public int SpecialtyId { get; set; }
        public int DisciplineId { get; set; }
        public int Semester { get; set; }
        public Discipline Discipline { get; set; }
    }
}
```

Рисунок 6.5.7 – Код ентіті (моделі бази даних)

Папка Enums зберігає enum-типи (тип перерахування з константними значеннями). Серед них, наприклад, тип оцінювання CreditType, що може приймати значення або Exam (екзамен), або Test (залік). Даний тип зображено на рисунку 6.5.8.

```
namespace DisciplineService.Enums
{
    public enum CreditType
    {
        Test,
        Exam
    }
}
```

Рисунок 6.5.8 – Код єнам-типу

Папка DbAccess – це про організацію та коректний мапінг даних на основі Entity Framework (EF) моделей. Окрім розширеного методу для зручної міграції, що вже було подано раніше на рисунку 6.4.6, тут є основна конфігурація даних.

Дана конфігурація описана в класі, що наслідує базовий для будь-якої моделі EF-клас DbContext. В межах контексту для мікросервіса дисиплін описано DbSet, що представляють собою фактично таблиці бази даних, а також описано конфігурацію окремих стовпців таблиць. Так, для дисциплін, каталогів та асоціативних таблиць описано співставлення властивостей (полів) ентіті-моделей та стовпців вже безпосередньо в базі даних. Окрім цього, описано які стовпці є обов’язковими, а також деякі додаткові обмеження, як-от, максимальну кількість символів рядку за допомогою методу HasMaxLength. Конфігурацію для контексту бази даних дисциплін представлено на рисунку 6.5.9.

```
using DisciplineService.Entities;
using Microsoft.EntityFrameworkCore;

namespace DisciplineService.DbAccess
{
    public class DisciplineDbContext : DbContext
    {
        public DbSet<Discipline> Disciplines { get; set; }
        public DbSet<Catalog> Catalogs { get; set; }
        public DbSet<Entities.CatalogDiscipline> CatalogDisciplines { get; set; }
        public DbSet<Entities.SpecialtyDiscipline> SpecialtyDisciplines { get; set; }


        public DisciplineDbContext(DbContextOptions<DisciplineDbContext> options) : base(options)
        {

        }

        public DisciplineDbContext()
        {
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            var discipline = modelBuilder.Entity<Discipline>();
            discipline.Property(x => x.Name).IsRequired().HasMaxLength(50);
            discipline.Property(x => x.Course).IsRequired();
            discipline.Property(x => x.Description).IsRequired().HasMaxLength(200);
            discipline.Property(x => x.CreditType).IsRequired();
            discipline.Property(x => x.IsSelective).IsRequired();
            discipline.Property(x => x.Hours).IsRequired();

            var catalog = modelBuilder.Entity<Catalog>();
            catalog.Property(x => x.Name).IsRequired().HasMaxLength(10);

            var catalogDiscipline = modelBuilder.Entity<Entities.CatalogDiscipline>();
            catalogDiscipline.Property(x => x.CatalogId).IsRequired();
            catalogDiscipline.Property(x => x.DisciplineId).IsRequired();

            var specialtyDiscipline = modelBuilder.Entity<SpecialtyDiscipline>();
            specialtyDiscipline.Property(x => x.SpecialtyId).IsRequired();
            specialtyDiscipline.Property(x => x.DisciplineId).IsRequired();
            specialtyDiscipline.Property(x => x.Semester).IsRequired();
        }
    }
}
```

Рисунок 6.5.9 – Код конфігурації контексту бази даних

Шар бізнес-логіки. В папці Models зберігаються основні моделі в тому вигляді, що вони повинні спочатку мапитись з ентіті-моделей з шару доступу до даних та в тому вигляді, в якому їх повинен отримувати шар API.

Всі такі моделі (аналогічно до ентіті-моделей з попереднього шару) є наслідниками базового класу BaseModel, що оголошує обов’язкове поле – унікальний ідентифікатор. Ці моделі зазвичай більш повні та комплексні, аніж ентіті-моделі. Код моделі дисципліни подано на рисунку 6.5.10.

```
namespace DisciplineService.Models
{
    public class DisciplineModel : BaseModel
    {
        public string Name { get; set; } = String.Empty;
        public string Description { get; set; } = String.Empty;
        public int Course { get; set; }
        public CreditType CreditType { get; set; }
        public int Hours { get; set; }
        public bool IsSelective { get; set; }
        public int? CatalogId { get; set; }
        public int? UniversityId { get; set; }
    }
}
```

Рисунок 6.5.10 – Код моделі

На рівні бізнес-логіки розташовуються найбільш комплексні класи – сервіси, що об’єднують ентіті-моделі в моделі, застосовуючи додаткову логіку та розрахунки. Сервіси використовують репозиторії з шару доступу до даних. Сервіс «дисципліна спеціальності» представлено на рисунку 6.5.11.

```
namespace CatalogDiscipline.Services
{
    public class SpecialtyDisciplineService : ISpecialtyDisciplineService
    {
        private readonly ISpecialtyDisciplineRepository _specialtyDisciplineRepository;

        private readonly IMapper _mapper;

        public SpecialtyDisciplineService(ISpecialtyDisciplineRepository specialtyDisciplineRepository, IMapper mapper)
        {
            _specialtyDisciplineRepository = specialtyDisciplineRepository;
            _mapper = mapper;
        }

        public async Task<SpecialtyDisciplineModel> AddAsync(SpecialtyDisciplineModel model)
        {
            var specialtyDiscipline = _mapper.Map<SpecialtyDiscipline>(model);

            var specialtyDisciplineeCreated = await _specialtyDisciplineRepository.AddAsync(specialtyDiscipline);

            await _specialtyDisciplineRepository.SaveAsync();

            return _mapper.Map<SpecialtyDisciplineModel>(specialtyDisciplineeCreated);
        }

        public async Task DeleteByIdAsync(int modelId)
        {
            await _specialtyDisciplineRepository.DeleteByIdAsync(modelId);

            await _specialtyDisciplineRepository.SaveAsync();
        }

        public async Task<IEnumerable<SpecialtyDisciplineModel>> GetAllAsync()
        {
            var specialtyDisciplines = await _specialtyDisciplineRepository.GetAllAsync();

            return _mapper.Map<IEnumerable<SpecialtyDisciplineModel>>(specialtyDisciplines);
        }

        public async Task<IEnumerable<DisciplineModel>> GetDisciplinesBySpecialtyIdAsync(int specialtyId)
        {
            var specialtyDisciplines = await _specialtyDisciplineRepository.GetAllAsync();

            return specialtyDisciplines.Where(x => x.SpecialtyId == specialtyId).Select(p => _mapper.Map<DisciplineModel>(p.Discipline));
        }

        public async Task<IEnumerable<DisciplineModel>> GetDisciplinesBySpecialtyIdAndSemesterAsync(int specialtyId, int semester)
        {
            var specialtyDisciplines = await _specialtyDisciplineRepository.GetAllAsync();

            return specialtyDisciplines.Where(x => x.SpecialtyId == specialtyId && x.Semester == semester).Select(p => _mapper.Map<DisciplineModel>(p.Discipline));
        }

    }
}
```

Рисунок 6.5.11 – Код сервісу

Шар клієнта (API). Якщо мікросервіс взаємодіє з іншим шляхом синхронної (по HTTPS-з'єднанню) або асинхронної (як у нас, шляхом передачі подій через event-bus RabbitMQ) комунікації, на даному шарі оголошуються додаткові DTO-моделі для відповідності моделей, що надходять з іншого мікросервіса та тих, що там обробляються. Так, для отримання в мікросервісі дисциплін об'єктів-спеціальностей з іншого мікросервіса, що відповідає за спеціальності, варто створити DTO спеціальності. Такі моделі зберігає відповідна папка DTO. DTO-модель зображено на рисунку 6.5.12.

```
namespace DisciplineService.Dtos
{
    public class SpecialtyDto : BaseModel
    {
        public string Name { get; set; } = string.Empty;
        public string Description { get; set; } = string.Empty;
        public string Cipher { get; set; } = string.Empty;
        public int? UniversityId { get; set; }
    }
}
```

Рисунок 6.5.12 – Код DTO-моделі

Ключова папка даного шару – це папка Controllers. Контролери і є точкою входу для клієнтської частини системи для отримання даних з серверів. Вони оголошують ендпоінти, що мають різні маршрути, звертаючись до яких, можна отримати чи обробити різні необхідні клієнту дані. Код контролера мікросервіса дисциплін зображено на рисунках 6.5.13 – 6.5.15.

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace DisciplineService.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    [Authorize]
    public class DisciplinesController : ControllerBase
    {
        private readonly IDisciplineService _disciplineService;
        private readonly ICatalogService _catalogService;
        private readonly ICatalogDisciplineService _catalogDisciplineService;
        private readonly ISpecialtyDisciplineService _specialtyDisciplineService;

        public DisciplinesController(IDisciplineService disciplineService, ICatalogService catalogService,
            ICatalogDisciplineService catalogDisciplineService, ISpecialtyDisciplineService specialtyDisciplineService)
        {
            _disciplineService = disciplineService;
            _catalogService = catalogService;
            _catalogDisciplineService = catalogDisciplineService;
            _specialtyDisciplineService = specialtyDisciplineService;
        }
```

Рисунок 6.5.13 – Код мікросервіса – 1

```
[HttpGet]
public async Task<ActionResult<IEnumerable<DisciplineModel>>> Get()
{
    var disciplines = await _disciplineService.GetAllAsync();

    return Ok(disciplines);
}

[HttpGet("byEI/{id}")]
public async Task<ActionResult<IEnumerable<DisciplineModel>>> GetByEI(int id)
{
    var disciplines = await _disciplineService.GetAllAsync();

    return Ok(disciplines.Where(p => p.UniversityId == id));
}

[HttpGet("{id}")]
public async Task<ActionResult<DisciplineModel>> Get(int id)
{
    var discipline = await _disciplineService.GetByIdAsync(id);

    return Ok(discipline);
}

[HttpGet("selective")]
public async Task<ActionResult<IEnumerable<DisciplineModel>>> GetSelective()
{
    var discipline = await _disciplineService.GetSelective();

    return Ok(discipline);
}

[HttpGet("mandatory")]
public async Task<ActionResult<IEnumerable<DisciplineModel>>> GetMandatory()
{
    var discipline = await _disciplineService.GetMandatory();

    return Ok(discipline);
}

[HttpDelete("{id}")]
public async Task<ActionResult> Delete(int id)
{
    await _disciplineService.DeleteByIdAsync(id);

    return NoContent();
}

[HttpPut("{id}")]
public async Task<ActionResult> Update(int id, DisciplineModel model)
{
    await _disciplineService.UpdateAsync(id, model);

    return NoContent();
}

[HttpPost]
public async Task<ActionResult> Add(DisciplineModel model)
{
    var created = await _disciplineService.AddAsync(model);

    return CreatedAtAction(nameof(Add), new { id = created.Id }, created);
}

[HttpGet("catalogs")]
public async Task<ActionResult<IEnumerable<CatalogModel>>> GetCatalogs()
{
    var catalogs = await _catalogService.GetAllAsync();

    return Ok(catalogs);
}

[HttpGet("catalogs/{id}")]
public async Task<ActionResult<CatalogModel>> GetCatalogById(int id)
{
    var catalog = await _catalogService.GetByIdAsync(id);

    return Ok(catalog);
}
```

Рисунок 6.5.14 – Код мікросервіса – 2

```
[HttpGet("catalogDisciplines")]
public async Task<ActionResult<IEnumerable<CatalogDisciplineModel>>> GetAllCatalogDisciplinesAsync()
{
    var catalogDisciplines = await _catalogDisciplineService.GetAllAsync();

    return Ok(catalogDisciplines);
}

[HttpGet("catalogDisciplines/{id}")]
public async Task<ActionResult<IEnumerable<DisciplineModel>>> GetDisciplinesByCatalogIdAsync(int id)
{
    var catalogDisciplines = await _catalogDisciplineService.GetDisciplinesByCatalogIdAsync(id);

    return Ok(catalogDisciplines);
}

[HttpPost("catalogDisciplines")]
public async Task<ActionResult> AddCatalogDiscipline(CatalogDisciplineModel model)
{
    var created = await _catalogDisciplineService.AddAsync(model);

    return CreatedAtAction(nameof(Add), new { id = created.Id }, created);
}

[HttpDelete("catalogDisciplines/{id}")]
public async Task<ActionResult> DeleteCatalogDiscipline(int id)
{
    await _catalogDisciplineService.DeleteByIdAsync(id);

    return NoContent();
}

[AllowAnonymous]
[HttpGet("specialtyDisciplines")]
public async Task<ActionResult<IEnumerable<SpecialtyDisciplineModel>>> GetAllSpecialtyDisciplinesAsync()
{
    var specialtyDisciplines = await _specialtyDisciplineService.GetAllAsync();

    return Ok(specialtyDisciplines);
}

[AllowAnonymous]
[HttpGet("specialtyDisciplines/{specialtyId}")]
public async Task<ActionResult<IEnumerable<DisciplineModel>>> GetDisciplinesBySpecialtyIdAsync(int specialtyId)
{
    var specialtyDisciplines = await _specialtyDisciplineService.GetDisciplinesBySpecialtyIdAsync(specialtyId);

    return Ok(specialtyDisciplines);
}

[AllowAnonymous]
[HttpGet("facultyDisciplines/{facultyId}")]
public async Task<ActionResult<IEnumerable<DisciplineModel>>> GetDisciplinesByFacultyIdAsync(int facultyId)
{
    var specialtyDisciplines = await _disciplineService.GetDisciplinesByFacultyId(facultyId);

    return Ok(specialtyDisciplines);
}

[AllowAnonymous]
[HttpPost("specialtyDisciplines/bySpecialtyAndSemester")]
public async Task<ActionResult<IEnumerable<DisciplineModel>>> GetDisciplinesBySpecialtyIdAsync(
    SpecialtyDisciplinesRequestModel model)
{
    var specialtyDisciplines = await _specialtyDisciplineService.GetDisciplinesBySpecialtyIdAndSemesterAsync(
        model.SpecialtyId, model.Semester);

    return Ok(specialtyDisciplines);
}

[HttpPost("specialtyDisciplines")]
public async Task<ActionResult> AddSpecialtyDiscipline(SpecialtyDisciplineModel model)
{
    var created = await _specialtyDisciplineService.AddAsync(model);

    return CreatedAtAction(nameof(Add), new { id = created.Id }, created);
}

[HttpDelete("specialtyDisciplines/{id}")]
public async Task<ActionResult> DeleteSpecialtyDiscipline(int id)
{
    await _specialtyDisciplineService.DeleteByIdAsync(id);

    return NoContent();
}
```

Рисунок 6.5.15 – Код мікросервіса – 3

6.3.4. Розробка клієнтської частини

Так як клієнтська частина компонента складання розкладів ВНЗ побудована за допомогою фреймворку Angular, то логічно застосувати всі основні елементи архітектури даного фреймворку. Дані елементи називають компонентами, що в свою чергу є розширенням інших важливих елемент – директив, але із додаванням HTML-шаблону. Так, фундаментом системи є компоненти – сукупності трьох файлів: файлу розмітки (HTML), файлу/таблиці стилів (CSS) та файлу власне бізнес-логіки (TypeScript-код, далі – TS-код). Компоненти ж об'єднані в модулі, а об'єднується все це у файлі app.module.ts, код якого зображений на рисунку 6.6.1.

```
import { MatSnackBarModule } from '@angular/material/snack-bar';
import { MatOptionModule } from '@angular/material/core';
import { JwtHelperService, JwtModule, JWT_OPTIONS } from '@auth0/angular-jwt';
import { TokenInterceptorService } from './shared/services/http-intrecepter.service';

export function getToken() {
  return localStorage.getItem('token');
}

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    CommonModule,
    AppRoutingModule,
    FormsModule,
    ReactiveFormsModule,
    BrowserModule,
    MatDialogModule,
    BrowserAnimationsModule,
    HttpClientModule,
    SharedModule,
    MatSnackBarModule,
    MatOptionModule
  ],
  providers: [{
    provide: HTTP_INTERCEPTORS,
    useClass: TokenInterceptorService,
    multi: true,
  },
  { provide: JWT_OPTIONS, useValue: JWT_OPTIONS },
  JwtHelperService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Рисунок 6.6.1 – Код app.module.ts

Не менш важливим для правильної роботи клієнта є і роутинг-конфігурація, тобто налаштування шляхів/маршрутів до різних модулів та їх компонент в межах однієї цілісної системи. На рисунку 6.6.2 зображено код файлу app-routing.module.ts, що для даної мети і створений.

```
const routes: Routes = [{
    path: '',
    redirectTo: 'management',
    pathMatch: 'full',
},
{
    path: 'management',
    loadChildren: () => import('./modules/management/management.module').then((m) => m.ManagementModule),
},
{
    path: 'management-edit',
    loadChildren: () => import('./modules/management-edit/management-edit.module')
        .then((m) => m.ManagementEditModule),
},
{
    path: 'authorization',
    loadChildren: () => import('./modules/authorization/authorization.module')
        .then((m) => m.AuthorizationModule),
},
{
    path: 'groups',
    loadChildren: () => import('./modules/groups/groups.module').then((m) => m.GroupsModule),
},
{
    path: 'schedule',
    loadChildren: () => import('./modules/schedule/schedule.module').then((m) => m.ScheduleModule),
},
{
    path: 'settings',
    loadChildren: () => import('./modules/settings/settings.module').then((m) => m.SettingsModule),
},
{ path: '**', redirectTo: '', pathMatch: 'full' }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
```

Рисунок 6.6.2 – Код app-routing.module.ts

Модуль shared об’єднує всі файли, що є спільними для різних компонент. Це – моделі, що повертаються з серверної частини додатку, основні єнам-моделі, основні сервіси (тут – сервіс, що стосується користувачів, авторизації та суміжних процесів, а також для роботи зі сховищем браузера localStorage, а також сервіс для спливаючих вікон) та деякі спільні компоненти (тут – всі спливаючі вікна та хедери). Структуру модуля зображено на рисунку 6.6.3.

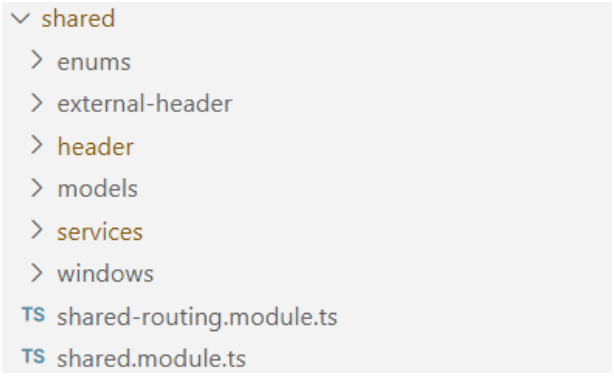

Рисунок 6.6.3 – Організація файлів модуля shared

Клієнтська частина поділена на декілька основних модулів, серед яких модуль авторизації, навчальних груп, менеджменту, редагування, розкладів та налаштувань. Структуру модулів зображено на рисунку 6.6.4.

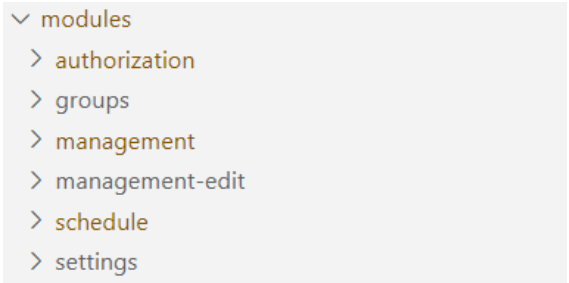

Рисунок 6.6.4 – Організація файлів основних модулів

Модуль авторизації (authorization) є сукупністю сторінки для авторизації та її внутрішніх компонент: форми реєстрації та форми входу в систему. Структуру даного модуля зображено на рисунку 6.6.5.

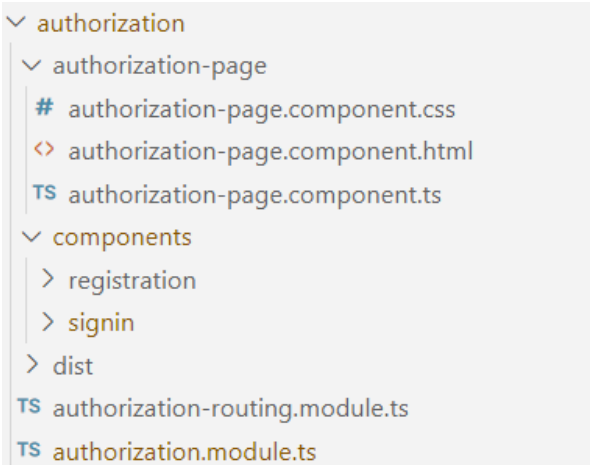

Рисунок 6.6.5 – Організація файлів модулю авторизації

Частковий TS-код модуля авторизації зображено на рисунку 6.6.6.

```
constructor(private router: Router,
  private route: ActivatedRoute,
  private usersService: UsersService, private notificationService: NotificationService) { }

ngOnInit(): void {
}

register(){
  this.registrationModel.PasswordRepeat = this.registrationModel.Password;

  this.usersService.eiNameIsUnique(this.eiName).subscribe(res => {
    if (res == true) {
      this.usersService.register(this.registrationModel)
      .subscribe(res1 => {
          this.usersService.createEI({ name: this.eiName, link: (this.eiName).toLowerCase()}).subscribe(res2 => {
            this.usersService.createUserEI({ eiId: res2.id, userId: res1,
              isAccepted: true, isAdmin: true, isAnswered: true }).subscribe(res => {
                this.notificationService.showSuccessMessage("You are successfully registered!");
                this.router.navigate(['authorization/sign-in']);
              });
          })
      });
    }
    else {
      this.notificationService.showErrorMessage("EI with this name is already created!");
    }
  });
```

Рисунок 6.6.6 – Код registration.component.ts

Часткова HTML-розмітка модуля авторизації зображена на рисунку 6.6.7.

```
          <img class="image-form" src="assets/AuthorizationImage.png" />
        </div>
    </div>
    <div class="form-container">
        <div class="form-logo">
            <div><img src="assets/Logo.png" style="width:50px;"/></div>
            <div style="margin-left:6px;">SCHEDULY</div>
        </div>
        <form>
            <div class="form">
                <div class="form-labels">
                    <a href='authorization/signin' style="color: #8E8E8E; font-weight: 500; text-decoration: none;">
                        Sign in
                    </a>
                    <span style="border-bottom: 3px solid #000; padding-bottom: 0.3px;">Registration</span>
                </div>
                <div>
                    <div class="form-group">
                        <label>EI* name</label>
                        <input [(ngModel)]="eiName" class="form-control" name="ei name" required
                        placeholder="Enter EI name"/>
                        <div class="errors-container">
                            <div *ngIf="!eiName" class="error">
                                Ei Name is required.
                            </div>
                        </div>
                    </div>
                    <div class="form-group">
                        <label>E-mail</label>
                        <input [(ngModel)]="registrationModel.Email"
                        class="form-control" name="email" required
                        placeholder="Enter e-mail" #email="ngModel" required
                        pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4}$"/>
```

Рисунок 6.6.7 – Код registration.component.html

Модуль навчальних груп (groups) представляє окрему сторінку для маніпуляції групами навчального закладу. TS-код та HTML-розмітку частково зображено на рисунках 6.6.8 та 6.6.9 відповідно.

```
    this.groups.forEach(element => {
      const semester = this.getCurrentSemesterByGroup(element);
      var request: IDisciplinesRequest = { groupId: element.id, semester: semester };
      console.log({ groupId: element.id, semester: semester })
      this.scheduleService.getScheduleDisciplinesByGroupAndSemesterId(request).subscribe(res => {
        this.scheduleService.getDisciplinesToAddBySpecialtyIdAndSemester({ specialtyId: element.specialtyId,
          semester: semester }).subscribe(res2 => {
            if (!res.length) {
              this.groupAndDisciplines.push({ groupId: element.id, scheduleDisciplines: res, disciplinesToAdd: res2,
              scheduleStatus: "Empty" });
            } else {
              if (res2.every(p => (res.some(o => o.disciplineId == p.id && o.isLecture))
                && (res.some(o => o.disciplineId == p.id && !o.isLecture)))) {
                this.groupAndDisciplines.push({ groupId: element.id, scheduleDisciplines: res, disciplinesToAdd: res2,
                  scheduleStatus: "Filled" });
              } else {
                this.groupAndDisciplines.push({ groupId: element.id, scheduleDisciplines: res, disciplinesToAdd: res2,
                  scheduleStatus: "In progress" });
              }
              }
        });
      });
    });
    });

  this.facultyService.get().subscribe(res => this.faculties = res);
}

redirectToAddGroup() {
  if (this.faculties.length) {
    this.router.navigateByUrl("/management-edit/create-group");
```

Рисунок 6.6.8 – Код groups-management.component.ts

```
<div class="Table" *ngIf="groups.length; else noGroups">
    <div class="Offer">
        <p id="Cipher" class="Offer_text">Cipher</p>
        <p id="Course" class="Offer_text">Course</p>
        <p id="Faculty" class="Offer_text">Faculty</p>
        <p id="Speciality" class="Offer_text">Speciality</p>
    </div>
    <hr class="main_line">
    <div class="Items">
        <div class="i" *ngFor="let group of groups">
            <div class="Item">
                <div class="Is">
                    <img *ngIf="currentSemesterScheduleStatus(group) == 'Filled'" src="assets/green.png" alt="">
                    <img *ngIf="currentSemesterScheduleStatus(group) == 'In progress'" src="assets/orange.png" a
                    <img *ngIf="currentSemesterScheduleStatus(group) == 'Empty'" src="assets/red.png" alt="">
                    <p class="Item_group">{{ group.cipher }}</p>
                </div>
                <div class="Course"><span class="course">{{ group.course }}</span></div>
                <div class="Form"><span class="name">{{ group.faculty?.name }}</span></div>
                <div class="Form"><span class="name">{{ group.specialty?.name }}</span></div>
                <div class="Images">
                    <img class="Item_img" src="assets/Edit.png" alt="" (click)="redirectToEditGroup(group.id)" >
                    <img class="Item_img" src="assets/Trash_bl.png" alt="" (click)="deleteGroup(group.id)">
                </div>
            </div>
            <hr class="dot_line">
        </div>
    </div>
```

Рисунок 6.6.9 – Код groups-management.component.html

Модуль налаштувань (settings) – це сторінка з інформацією про доступи до різних навчальних закладів та їх учасників. TS-код та HTML-розмітку частково зображено на рисунках 6.6.10 та 6.6.11 відповідно.

```
    this.selectedEIToTeamView = this.eis[0];

    this.userService.get().subscribe(res => this.allUsers = res);

    this.userService.getAllUserEis().subscribe(res => this.allUserEIs = res);

    this.userService.getAllUserEisFullInfoByEIId(this.eis[0].id).subscribe(res => {
      res.forEach(element => {
        this.eiTeamMembers.push({ user: element.user ? element.user : {} as User, isAdmin: element.isAdmin,
          isAccepted: element.isAccepted, isAnswered: element.isAnswered, id: element.id });
      });
    });
  }

  deleteMember(value: {user: User, isAdmin: boolean, isAccepted: boolean, isAnswered: boolean, id: number }) {
    const currentUser = this.userService.user;
      this.userService.deleteUserEI(value.id).subscribe(res => {
        window.location.reload();
      });
  }

  isActiveEI(ei: EI) {
    return { 'active-ei': this.selectedEIToTeamView.id == ei.id };
  }

  getTeamMembers() {
    return this.eiTeamMembers.filter(p => p.isAnswered && p.isAccepted);
  }

  getRejectedMembers() {
    return this.eiTeamMembers.filter(p => p.isAnswered && !p.isAccepted);
```

Рисунок 6.6.10 – Код eiteam-management.component.ts

```
<div class="team-members-list">
    <div class="member-item" *ngFor="let eiTeamMember of getTeamMembers()">
        <div class="mark-email"><i class="fa-solid fa-check fa-lg member-mark"></i> {{ eiTeamMember.user.ema
        <div class="role">
            <ng-container *ngIf="eiTeamMember.isAdmin; else member">
                Admin
            </ng-container>
            <ng-template #member>
                Member
            </ng-template>
        </div>
        <div class="delete-icon" (click)="deleteMember(eiTeamMember)"
        *ngIf="!eiTeamMember.isAdmin"><i class="fa-solid fa-lg fa-xmark delete-icon"></i></div>
    </div>
</div>
<div class="label-2" *ngIf="getWaitingMembers().length">Waiting</div>
<div class="team-members-list" *ngIf="getWaitingMembers().length">
    <div class="member-item" *ngFor="let eiTeamMember of getWaitingMembers()">
        <div class="mark-email"><i class="fa-solid fa-spinner fa-lg spinner-mark"></i> {{ eiTeamMember.user.
        <div class="role">Member</div>
        <div class="delete-icon" (click)="deleteMember(eiTeamMember)"
        *ngIf="!eiTeamMember.isAdmin"><i class="fa-solid fa-lg fa-xmark delete-icon"></i></div>
    </div>
</div>
<div class="label-2" *ngIf="getRejectedMembers().length">Rejected</div>
<div class="team-members-list" *ngIf="getRejectedMembers().length">
    <div class="member-item" *ngFor="let eiTeamMember of getRejectedMembers()">
        <div class="mark-email"><i class="fa-solid fa-xmark fa-lg rejected-mark"></i> {{ eiTeamMember.user.e
        <div class="role">Member</div>
        <div class="delete-icon" (click)="resendInvitation(eiTeamMember.user.email)"
```

Рисунок 6.6.11 – Код eiteam-management.component.html

Модуль розкладів (schedule) створений з ціллю маніпулювання розкладами. Сюди включено основні моделі, що пов'язані з розкладами та підписками електронною поштою на розклади. Основні компоненти, що містяться в папці components – це компонент власне розкладу групи, компонент розкладу викладача та компонент для редагування розкладу групи. Окрім цього, виділено окремий компонент, що представляє вигляд розкладів для зовнішнього користувача. Структуру даного модуля зображено на рисунку 6.6.12.

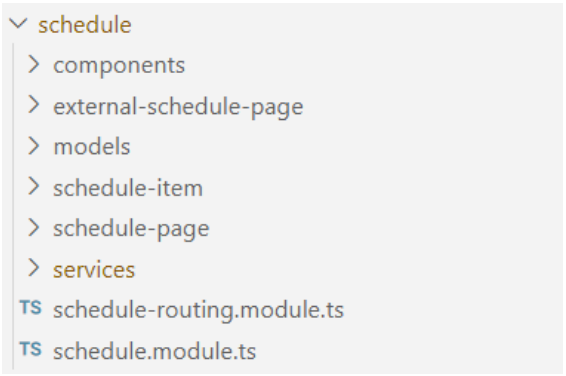

Рисунок 6.6.12 – Організація файлів модулю розкладів

На рисунках 6.6.13-6.6.14 зображено TS-код розкладу груп та викладачів.

```
containsSelective(day: number, lesson: number) {
  return this.searchByDayAndLesson(day, lesson).some(p => p.catalogName);
}

getCatalogName(day: number, lesson: number) {
  return this.searchByDayAndLesson(day, lesson)[0].catalogName;
}

isEmpty(day: number, lesson: number) {
  return this.searchByDayAndLesson(day, lesson).length == 0;
}

endOfDisc(day: number, lesson: number, i: number) {
  return this.searchByDayAndLesson(day, lesson).length == i + 1;
}

isEmptyDay(day: number) {
  const isEmptyDay: boolean = this.scheduleDisciplines.some(p => p.day == day);

  return { 'blured': !isEmptyDay };
}

isGroupSelected() {
  return { 'disabled-edit' : !this.selectedGroupId }
}

isTodayForExternal(day: number) {
  var now = new Date();
  if (this.isExternalRoute() && now.getDay() == day) {
    return { 'today' : true }
```

Рисунок 6.6.13 – Код group.component.ts

```
scheduleIdAndGroupCipher: { scheduleId: number, groupCipher: string }[] = [];

private allScheduleDiscipline: IScheduleDiscipline[] = [];

constructor(private router: Router, private groupService: GroupsService, private usersService: UsersService,
  private scheduleService: ScheduleService, private notificationService: NotificationService) { }

ngOnInit(): void {
  this.groupService.getByEIId(this.usersService.getCurrentEIId()).subscribe(res => {
    this.groups = res;
  });

  this.scheduleService.get().subscribe(res => {
    res.forEach(element => {
      this.groupService.getById(element.groupId).subscribe(res2 => {
        this.scheduleIdAndGroupCipher.push({ scheduleId: element.id, groupCipher: res2.cipher });
      })
    });
  })

  if (!this.selectedSemester) {
    this.selectedSemester = 1;
  }
}

redirectToGroupManagement() {
  this.router.navigateByUrl("groups/management");
}

addItemGroup(newItem: any) {
  this.scheduleDisciplines = [];
  if (newItem == 'Select Teacher') {
    this.selectedTeacherId = -1;
  } else {
    this.selectedTeacherId = newItem;
```

Рисунок 6.6.14 – Код teacher.component.ts

Модулі менеджменту (management та management-edit) є частиною один одного і використовуються для можливості маніпулювати сутностями навчального закладу (каталогами, факультетами тощо). Модуль management (6.6.15) використовується для перегляду сутностей навчального закладу.

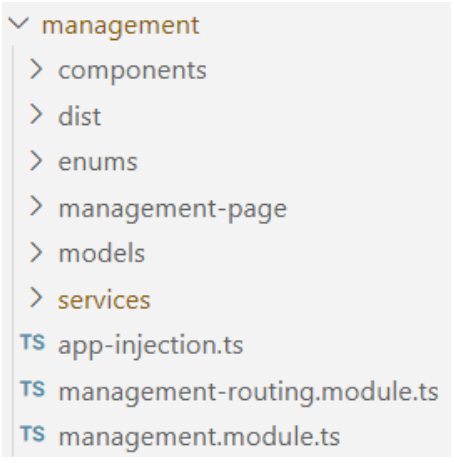

Рисунок 6.6.15 – Організація файлів модулю менеджменту (перегляду)

Модуль management-edit (рисунок 6.6.16), в свою чергу, необхідний саме для створення та редагування сутностей.

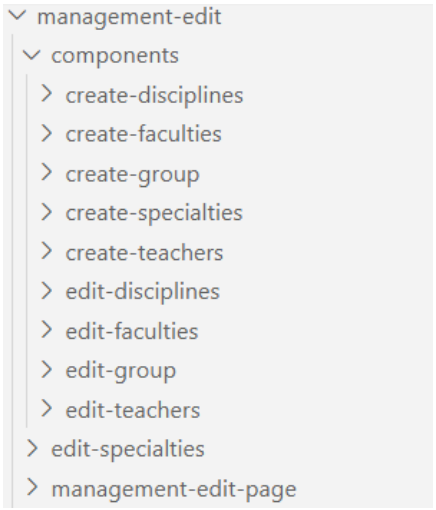

Рисунок 6.6.16 – Організація файлів модулю менеджменту (редагування)

На рисунках 6.6.17-6.6.20 зображено TS-код менеджменту дисциплін та викладачів (додавання та створення/редагування відповідно).

```
creditTypeByIndex(index: number): string {
  switch(index)
  {
    case 0:
      return 'Test';
    case 1:
      return 'Exam';
    default:
      return 'Not found';
  }
}

changeFilter(selectiveChanged: boolean) {
  if (selectiveChanged) {
    this.showSelective = !this.showSelective;
  } else {
    this.showMandatory = !this.showMandatory;
  }

  this.filter(this.showSelective, this.showMandatory)
}

filter(showSelective: boolean, showMandatory: boolean) {
  if (showSelective && !showMandatory) {
    this.disciplinesService.getSelective().subscribe(
      res => this.disciplines = res
    );
  } else if (showMandatory && !showSelective) {
    this.disciplinesService.getMandatory().subscribe(
      res => this.disciplines = res
    );
  } else if (showSelective && showMandatory) {
    this.disciplinesService.getByEIId(this.usersService.getCurrentEIId()).subscribe(
```

Рисунок 6.6.17 – Код disciplines.component.ts

```
showDisciplinesList(teacher: ITeacher) {
  var teacherUniqueDisciplines: IDiscipline[] = [];

  this.disciplines.filter(p => p.teacherId == teacher.id).map(i => i.discipline).forEach(element => {
    if (!teacherUniqueDisciplines.some(p => p.id == element.id)) {
      teacherUniqueDisciplines.push(element);
    }
  });

  this.windowService.openShowDisciplinesListDialog({
      buttons: [
        {
          title: "OK",
          onClickEvent: new EventEmitter<void>(),
        },
      ],
      title: 'Disciplines list',
      message: `${teacher.name} ${teacher.surname} ${teacher.patronymic}`,
      teacher: teacher,
      disciplinesOfTeacher: teacherUniqueDisciplines
  });
}

getByName(event: Event, value: any) {
  if (value == "") {
    this.teachers = this.allTeachers;
  } else if (this.allTeachers.some(p => p.name.toLowerCase().includes(value.toLowerCase()))) {
    this.teachers = this.allTeachers.filter(p => p.name.toLowerCase().includes(value.toLowerCase()));
  }
  else {
    this.notificationService.showErrorMessage("Nothing found by input name");
  }
}
```

Рисунок 6.6.18 – Код teachers.component.ts

```
submit(form: NgForm) {
  if (this.disciplines.some(p => p.name == form.value["name"]) && !this.isEditRoute()) {
    this.notificationService.showErrorMessage("Discipline with this name already exists!");
  } else if (!this.isEditRoute()) {

    var discipline: ISaveDiscipline = { name: form.value["name"], description: form.value["description"],
    course: form.value["course"], creditType: form.value["creditType"] == "Test" ? 0 : 1, hours: form.value["hours"],
    isSelective: form.value["isSelective"], catalogId: +this.selectedCatalogId ? +this.selectedCatalogId : undefined,
    universityId: JSON.parse(localStorage.getItem('selectedEI') as string) };

    console.log(discipline)

    if (discipline.isSelective) {
      this.disciplineService.create(discipline)
      .subscribe((res) => {
        const catalogDiscipline: ISaveCatalogDiscipline = { catalogId: +this.selectedCatalogId,
          disciplineId: res.id };
        this.catalogDisciplineService.create(catalogDiscipline)
        .subscribe(() => {
          if (this.lecturers) {
            this.lecturers.forEach(element => {
              const disciplineTeacher: ISaveDisciplineTeacher = { teacherId: element.id, disciplineId: res.id,
                isLecturer: true };
              this.disciplineTeacherService.create(disciplineTeacher).subscribe();
            });
          }

          if (this.practicians) {
            this.practicians.forEach(element => {
              const disciplineTeacher: ISaveDisciplineTeacher = { teacherId: element.id, disciplineId: res.id,
                isLecturer: false };
              this.disciplineTeacherService.create(disciplineTeacher).subscribe();
            });
          }
```

Рисунок 6.6.19 – Код create-disciplines.component.ts

```
ngOnInit(): void {
  this.route.params.subscribe(params => {
    this.id = params['id'];
  });

  this.teacherService.get().subscribe(res => this.teachers = res);

  this.isEditRoute();
}

submit(form: NgForm) {
  if (this.teachers.some(p => p.name == form.value["name"] && p.surname == form.value["surname"]
  && p.patronymic == form.value["patronymic"]) && !this.isEditRoute()) {
    this.notificationService.showErrorMessage("Teacher with this name, surname and patronymic already exists!");
  } else if (this.isEditRoute()) {
    var teacher: ITeacher = { name: form.value["name"], surname: form.value["surname"],
    patronymic: form.value["patronymic"], universityId: JSON.parse(localStorage.getItem('selectedEI') as string),
    id: this.id };

    this.teacherService.update(this.id, teacher)
    .subscribe(() => {
      this.notificationService.showSuccessMessage("Teacher was successfully updated!");
      this.redirectToManagement();
    });
  } else {
    var saveTeacher: ISaveTeacher = { name: form.value["name"], surname: form.value["surname"],
    patronymic: form.value["patronymic"], universityId: JSON.parse(localStorage.getItem('selectedEI') as string) };

    this.teacherService.create(saveTeacher)
    .subscribe(() => {
      this.redirectToManagement();
    });
  }
}
```

Рисунок 6.6.20 – Код create-teachers.component.ts

Додатково до компонент, використано важливу функціональність фреймворку Angular – гуард (guard). Guards дозволяють обмежити навігацію по певним маршрутам. Наприклад, якщо для доступу до певного ресурсу необхідна наявність аутентифікації тощо. Тобто guards захищають доступ до ресурсу. Код гуарду для доступу до сторінок лише авторизованим користувачам зображено на рисунку 6.6.21. Використаний він в файлах налаштування маршрутизації.

```
@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {
  constructor(public accountService: UsersService, public router: Router){}

  canActivate():boolean {
    if(!this.accountService.isAuthenticated()){
      this.router.navigate(['/authorization/sign-in']);
    }
    return true;
  }
}
```

Рисунок 6.6.21 – Код auth.guard.ts