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

Кваліфікаційна робота
за освітнім ступенем «бакалавр»
на тему:

ІНФОРМАЦІЙНА СИСТЕМА ПІДТРИМКИ ЕЛЕКТРОННОГО ОБІГУ ДОКУМЕНТІВ ЗАКЛАДУ ВИЩОЇ ОСВІТИ. КОНТИНГЕНТ СТУДЕНТІВ. РОЗПОДІЛ СТАВОК

Виконав:
здобувач IV курсу
групи ІПЗ-41
спеціальності 121 «Інженерія
програмного забезпечення»
Степан Сергійович Осмолович

Науковий керівник:
кандидат технічних наук, доцент
Степанія Михайлівна Бабич

Рівне – 2025

ЗМІСТ

СПИСОК ТЕРМІНІВ ТА УМОВНИХ ПОЗНАЧЕНЬ

ЗВО	Заклад вищої освіти
ЄДЕБО	Єдина Державна Електронна База з питань Освіти
СЕД	Система електронного документообігу
ЕДО	Електронний документообіг
ЕЦП	Електронний цифровий підпис.
КЕП	Кваліфікований електронний підпис.
БД	База даних
ІПЗ	Інженерія програмного забезпечення
СУБД	Система управління базами даних
API	Інтерфейс прикладного програмування
РБД	Розподілені бази даних
НП	Навчальний план

ВСТУП

Актуальність роботи. Інформаційна система підтримки електронного обігу документів для закладу вищої освіти є важливою для ефективного управління контингентом студентів і розподілу ставок. Традиційні паперові системи є затратними за часом і ресурсами, а також створюють ризик помилок при обробці даних. Автоматизація документообігу дозволяє зменшити адміністративні витрати, підвищити швидкість обробки інформації та покращити доступність даних. У зв'язку з розвитком цифрових технологій і збільшенням обсягу інформації, актуальність впровадження електронних систем для оптимізації процесів управління в освітніх закладах постійно зростає. Крім того, виникає потреба в автоматизованому розрахунку педагогічного навантаження, зокрема формування ставок викладачів на основі навчальних планів та кількості студентів, що неможливо реалізувати ефективно без сучасних інформаційних технологій.

Метою роботи є конструювання підсистеми для обліку та управління контингентом студентів і розрахунку кількості ставок навчального навантаження як складових частин цілісної інформаційної системи підтримки електронного документообігу в закладі вищої освіти.

Для досягнення сформульованої мети були поставлені та вирішені наступні **завдання**:

- ✓ *постановка задачі.* Вивчення предметної області задачі: принципи організації електронного документообігу в ЗВО; конструювання інтегрованих комп’ютерних інформаційних систем; створення веб-компонентів, зокрема мікро-сервісів для обліку студентів і для розрахунку ставок викладачів на основі ~~структури~~ навчальних планів;
- ✓ *аналіз інструментальних засобів*. Проведено порівняльний аналіз інструментів для створення інформаційних систем підтримки документообігу: фреймворки FastAPI для бекенду, бібліотеки Pandas і openpyxl для обробки Excel-файлів, PostgreSQL як СУБД. Обґрунтовано

вибір зазначених технологій як оптимальних для створення мікро-сервісів з високою продуктивністю, модульністю та підтримкою інтеграції;

- ✓ *визначення функціональних вимог.* Сформульовано вимоги до підсистеми управління контингентом, включно з обробкою особистих та академічних даних здобувачів вищої освіти, а також до мікро-сервісу для розрахунку ставок, який має виконувати обробку навчальних планів, визначати курс навчання та на основі кількості студентів у групах автоматично розраховувати обсяг годин та ставок для викладачів;
- ✓ *розробка архітектури та моделі даних підсистеми.* Створено загальну архітектуру мікро-сервісної системи, яка охоплює два незалежні, але взаємодіючі сервіси: один для імпорту та обробки студентських даних, другий – для роботи з файлами навчальних планів і розрахунку навчального навантаження. Розроблено структури баз даних та логіку взаємодії компонентів системи.
- ✓ *розроблення та програмна реалізація.* Реалізовано два мікро-сервіси з використанням FastAPI:
  - перший – для завантаження Excel-файлів зі списками студентів, парсингу даних, збереження до БД, пошуку, фільтрації та групування за курсами й групами;
  - другий – для обробки планів навчання: визначення курсу на основі семестрового навантаження, розрахунку загальної кількості годин на дисципліну з урахуванням чисельності студентів у групах, а також формування JSON-виводу з розподілом ставок. Це дало змогу створити об'єктивну основу для розрахунку навантаження викладачів [4];
- ✓ *тестування та відлагодження компонентів.* Проведено комплексне тестування мікро-сервісів для перевірки стабільності їхньої роботи, правильності обробки даних та взаємодії з фронтендом. Забезпечено надійне збереження інформації у базі да них та правильне формування результатів розрахунків, що дозволяє уникнути ручних помилок і забезпечити автоматизовану підтримку кадрових рішень.

Об’єкт дослідження: інтегровані комп’ютерні інформаційні системи підтримки електронного обігу документів закладів вищої освіти.

Предмет дослідження: розробка вебкомпонентів у складі інтегрованих систем, зокрема мікро-сервісів, що забезпечують автоматизовану обробку даних про студентів і педагогічне навантаження.

Методи дослідження: системний аналіз, методи моделювання та проєктування інформаційних процесів, підхід мікро-сервісної архітектури, технології REST API, методи тестування інформаційних систем.

Практичне значення дослідження. Розроблені компоненти інтегруються у централізовану платформу електронного документообігу закладу вищої освіти. Вони забезпечують автоматизований облік контингенту студентів, доступ до даних за різними параметрами (група, курс, спеціальність), а також реалізують логіку розрахунку педагогічного навантаження за навчальними планами. Завдяки автоматичному визначенню кількості ставок, система дозволяє уникати помилок у ручних підрахунках, спрощує планування навантаження та підвищує ефективність управління освітнім процесом. Реалізація цих мікро-сервісів значно покращує керованість і прозорість внутрішніх освітніх процесів.

Апробація і впровадження результатів дослідження. Основні теоретичні та практичні результати дослідження доповідалися і обговорювалися на XVIII Всеукраїнській науково-практичній конференції здобувачів вищої освіти та молодих учених «Наука, освіта, суспільство очима молодих» (м. Рівне, 14 травня 2025 року), звітній науковій конференції викладачів, співробітників і здобувачів вищої освіти Рівненського державного гуманітарного університету за 2024 рік (м. Рівне, 15 травня 2025 року).

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку використаних джерел та додатку. Загальний обсяг роботи становить 53 сторінки. Вона містить 21 рисунок. Список використаних джерел включає 20 найменувань. Обсяг додатків – 7 сторінок.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Інформаційні системи

Термін «система» походить з грецької мови та означає об’єднання окремих елементів у єдине ціле. У навколишньому світі нас оточують різноманітні системи: технічні пристрої, організми, спільноти людей тощо. В інформатиці предметом вивчення є саме інформаційні системи.

Під інформаційною системою розуміють таку систему, у якій реалізуються процеси, пов’язані з обробкою, збереженням і передаванням інформації.

Інформаційні процеси можуть бути різноманітними за характером і кількістю. Комп’ютер, як спеціально створений пристрій для роботи з інформацією, виконує майже всі такі процеси. Найскладнішою та найефективнішою інформаційною системою вважається людина.

Кожну інформаційну систему можна віднести до певного типу – наприклад, технічного чи соціального, що дозволяє класифікувати інформаційні системи на різні категорії (див. рис. 1.1).

Рисунок 1.1. Класифікація інформаційних систем

У межах шкільного курсу інформатики розглядаються лише технічні інформаційні системи, серед яких виділяються персональні комп’ютери та локальні мережі.

Такі системи складаються з двох основних компонентів: апаратної частини та програмного забезпечення.

Апаратне забезпечення включає всі фізичні пристрої, що забезпечують функціонування інформаційної системи.

Програмне забезпечення є сукупністю програм, які керують апаратними засобами та реалізують інформаційні процеси.

Інформаційна система не існує ізольовано – вона обмінюється даними з іншими системами (рис. 1.2). Вона здатна звертатися до зовнішніх джерел для отримання інформації, а також відповідати на запити користувачів, які взаємодіють з нею [3].

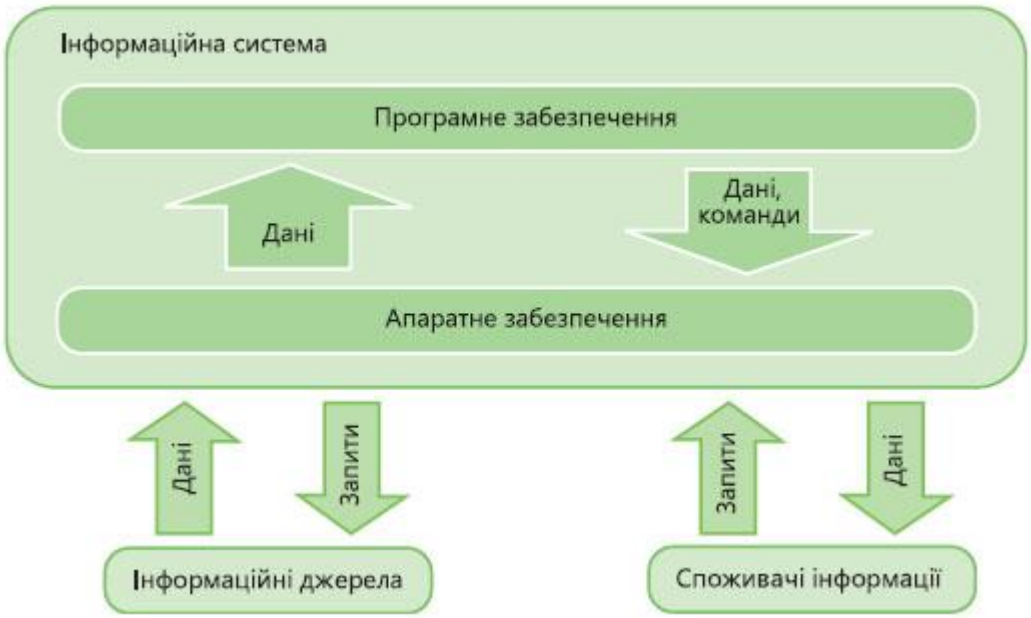


Рисунок 1.2. Схема взаємодії інформаційної системи

1.2. Електронний документообіг та системи його організації

Електронний документообіг (ЕДО) передбачає обмін, обробку та зберігання документів у цифровому форматі замість паперового. Усі етапи – від створення до збереження – відбуваються в електронному середовищі. Такий

підхід не лише прискорює обробку документів, а й значно знижує витрати на друк, папір і фізичне архівування.

Використання електронного документообігу надає змогу миттєво направляти документи на затвердження або підпис, спрощуючи внутрішні процедури та підвищуючи загальну ефективність. Крім того, цифрова форма забезпечує кращий контроль над документами, зменшує ризики втрати або пошкодження та підвищує інформаційну безпеку.

Для організації ЕДО використовують спеціалізовані інформаційні системи, що гарантують захист переданої інформації та автоматизують рутинні операції. Одним із ключових інструментів таких систем є **Система електронного документообігу** (СЕД) – програмне рішення, що охоплює весь життєвий цикл документа: від створення до архівного зберігання.

СЕД дозволяє:

- створювати та редагувати документи в цифровому форматі;
- здійснювати контроль доступу;
- організовувати погодження за заданими маршрутами;
- накладати електронний підпис (ЕЦП або КЕП);
- автоматизувати розсилання, нагадування та контроль виконання.

Залежно від області використання, СЕД можна поділити на:

- **внутрішні** – призначені для документообігу всередині установи або компанії. Це можуть бути накази, внутрішні інструкції, положення, розпорядження, службові записки тощо;
- **зовнішні** – використовуються для взаємодії з партнерами, замовниками, підрядниками. Вони забезпечують обмін документами між різними організаціями, включаючи можливість підписання їх КЕП.

У багатьох випадках обидва типи систем інтегруються в єдину платформу, що забезпечує повну цифрову взаємодію як усередині організації, так і за її межами.

Системи ЕДО також класифікуються за функціональністю:

- **системи діловодства** – орієнтовані на класичний документообіг із зовнішніми контрагентами;

- **workflow-системи** – автоматизують бізнес-процеси, пов’язані з обробкою документів;
- **архівні модулі** – забезпечують безпечне довготривале зберігання документів із можливістю швидкого пошуку;
- **ECM-системи** (Enterprise Content Management) – універсальні платформи, що поєднують усі перелічені функції, а також інструменти для управління контентом організації.

Основні можливості сучасних СЕД включають:

- погодження та підпис документів у режимі онлайн із використанням КЕП або хмарних підписів;
- надійне цифрове сховище для всіх видів документів з розподілом прав доступу;
- автоматизацію типових процесів, таких як пересилання, узгодження, контроль строків, архівування;
- організацію співпраці між користувачами незалежно від їх фізичного розташування;
- ведення історії змін та дій користувачів для забезпечення прозорості та відповідності вимогам інформаційної безпеки.

Функціональність СЕД може суттєво відрізнятися залежно від конкретного рішення. Наприклад, система **Megapolis.DocNet** надає:

- інструменти для створення, реєстрації та погодження документації;
- систему моніторингу та нагадувань щодо термінів виконання;
- автоматизовані маршрути обробки документів;
- можливість працювати з документами в автономному режимі;
- вбудовані звіти та візуальні аналітичні панелі;
- управління договорами, дорученнями, штатним розкладом та оргструктурою;
- модулі для сканування, розпізнавання, QR-кодування документів;
- інтеграцію з іншими інформаційними системами, зокрема СЕВ ОВВ.

Таким чином, СЕД виступає ключовим інструментом для оптимізації документообігу, забезпечуючи оперативність, безпечність та контрольованість обробки інформації в цифровому середовищі [15].

1.3. Законодавче регулювання електронного документообігу в Україні

У науковій статті М. О. Вронського та І. В. Панченко «Впровадження електронного документообігу в кадрову роботу закладу вищої освіти» висвітлено ключові етапи формування нормативної бази електронного документообігу в Україні протягом останніх років (рис. 1.3) [1]. Спираючись на викладений матеріал, варто здійснити аналіз причин, з яких саме ці періоди було визначено як основоположні. Автори публікації наводять такі етапи розвитку.

Рисунок 1.3. Етапи розвитку електронного документообігу в Україні

Першим значущим кроком стало запровадження організаційно-правових передумов функціонування електронного документообігу. Так, 22 травня 2003 року Верховною Радою України було ухвалено Закон України «Про електронні документи та електронний документообіг». Цей нормативно-правовий акт заклав основу для правомірного використання електронних документів, створюючи відповідне середовище для їх широкого впровадження. Закон передбачає визначення електронних документів як юридично значущих у всіх сферах діяльності, включно з державним управлінням, підприємництвом, освітнім та громадським сектором. Прийняття цього закону стало вагомим поштовхом до цифровізації документальних процесів у країні [8].

Того ж дня, 22 травня 2003 року, українським парламентом було прийнято ще один важливий нормативний акт – Закон України «Про електронний цифровий підпис». Його ключове положення полягало у закріпленні юридичної сили за електронним документом за умови наявності відповідних реквізитів та можливості його підтвердження паперовим аналогом, що мало надзвичайно важливе значення для формування довіри до цифрових рішень у документообігу [6].

Наступним етапом стало впровадження електронної звітності у податковій сфері. 14 серпня 2017 року було затверджено «Положення про застосування електронного підпису та електронної печатки», що стало важливою передумовою для розвитку цифрової звітності на державному рівні. Подальшим кроком стало прийняття 17 січня 2018 року Кабінетом Міністрів України постанови «Про деякі питання документування управлінської діяльності». Цей документ містив рекомендації органам виконавчої влади щодо переходу на електронний документообіг. Його метою було встановлення чітких норм щодо створення, оформлення, обігу та зберігання управлінських документів у цифровому форматі, що сприяло послідовному впровадженню електронної документації в державному управлінні. Постанова забезпечила системний підхід до документування управлінських процесів та посилила правову визначеність у цій сфері [2].

Значною віхою у розвитку електронного документообігу стало введення в дію 7 листопада 2018 року Закону України «Про електронні довірчі послуги». З цього моменту Закон України «Про електронний цифровий підпис» втратив чинність. Новий закон був розроблений з метою формування довіри до електронної взаємодії, цифрових комунікацій та онлайн-бізнесу. Він забезпечує легітимність та правовий захист електронного підпису, електронної печатки, а також інших інструментів ідентифікації. Цей нормативний акт є надзвичайно важливим для становлення електронної економіки, розвитку електронного урядування, онлайн-комерції та цифрових транзакцій. Закон сприяє створенню умов для безпечного та правомірного обміну електронною інформацією,

забезпечуючи юридичну силу електронних підписів і довірчих послуг, що в свою чергу стимулює покращення та автоматизацію бізнес-процесів [7].

Крім зазначених нормативно-правових актів, доцільно згадати й інші документи, які також суттєво впливають на регламентацію електронного документообігу в Україні. Одним із таких є Закон України «Про захист інформації в інформаційно-телекомунікаційних системах», прийнятий 5 липня 1995 року. Цей документ є базовим у сфері інформаційної безпеки, оскільки регулює порядок захисту даних в електронних системах, включно з платформами для електронного документообігу. Його мета полягає у запобіганні несанкціонованому доступу, втраті, пошкодженню чи розголошенню інформації, що зберігається або передається в електронному вигляді. Закон встановлює обов'язкові вимоги щодо безпеки інформації, які мають дотримуватися всі суб'єкти, що працюють з інформаційно-комунікаційними технологіями [9].

1.4. Формулювання завдання

Результати проведеного дослідження засвідчили, що організація документообігу в установах вищої освіти є одним із ключових чинників, який впливає на ефективність функціонування як адміністративного апарату, так і освітнього процесу в цілому. Особливого значення в цьому контексті набуває контроль за чисельністю студентів, а також правильне і своєчасне розподілення навчального навантаження серед науково-педагогічного складу, оскільки ці показники можуть змінюватися протягом навчального року. Впровадження сучасної електронної системи документообігу дозволить знизити обсяг рутинної адміністративної праці, заощадити робочий час і підвищити рівень точності та прозорості даних.

Розроблювана програмна система повинна забезпечувати комплексну підтримку електронного документообігу, що охоплює облік студентського складу, збереження персоніфікованих даних, відслідковування статусу студентів, а також інші критично важливі аспекти навчального процесу.

У рамках даної системи доцільно реалізувати два головні функціональні компоненти:

- **модуль обліку студентського контингенту** – відповідатиме за збереження, оновлення та доступ до структурованої інформації про студентів, класифікованої за ознаками факультету, спеціальності, курсу, академічної групи та індивідуального статусу. Цей модуль має забезпечити зручний пошук, сортування записів за заданими критеріями та формування статистичних зведень для аналітики;
- **модуль розподілу педагогічного навантаження** – спрямований на організацію процесу розподілу навчальних ставок між викладачами з урахуванням кількості студентів у групах, особливостей навчальних планів та фактичного навантаження.

Передбачено запровадження багаторівневої системи доступу, що дозволить адміністративному персоналу користуватись усіма функціональними можливостями, тоді як інші категорії користувачів отримають доступ лише до інформації, релевантної їхнім повноваженням. Інтерфейс програми має бути простим для сприйняття, зручним у використанні як для адміністративних працівників, так і для викладацького складу.

Серед основних функціональних вимог до додатка:

- забезпечення повноцінного електронного документообігу, що дозволить зменшити часові та матеріальні витрати на опрацювання документації;
- надання достовірної та прозорої інформації про студентів;
- оперативне оновлення даних у реальному часі;
- адаптивний, зручний користувацький інтерфейс, сумісний з комп’ютерами і мобільними пристроями.

Отже, створення такої інформаційної системи дозволить оптимізувати управління студентським контингентом, покращити внутрішню організацію освітнього закладу та забезпечити стабільне, ефективне функціонування ключових процесів у сфері вищої освіти [5].

РОЗДІЛ 2
ВИБІР ТА ОБГРУНТУВАННЯ ІНСТРУМЕНТІВ ПРОЄКТУВАННЯ

2.1. Середовище розробки

Вибір середовища для розробки є одним із ключових етапів під час створення програмного забезпечення, оскільки від цього залежить не лише зручність написання коду, а й швидкість реалізації проєкту та доступ до функціональних можливостей. Саме тому було розглянуто три основні середовища: Python IDLE, PyCharm та Visual Studio Code. Щоб зробити обґрунтований вибір, було проведено аналіз переваг і недоліків кожного з них.

Python IDLE (Integrated Development and Learning Environment) (рис. 2.1) – це стандартне середовище розробки, яке встановлюється разом із інтерпретатором Python. Воно має базовий набір функцій, необхідних для написання та запуску програмного коду, і за інтерфейсом подібне до PowerShell [13].

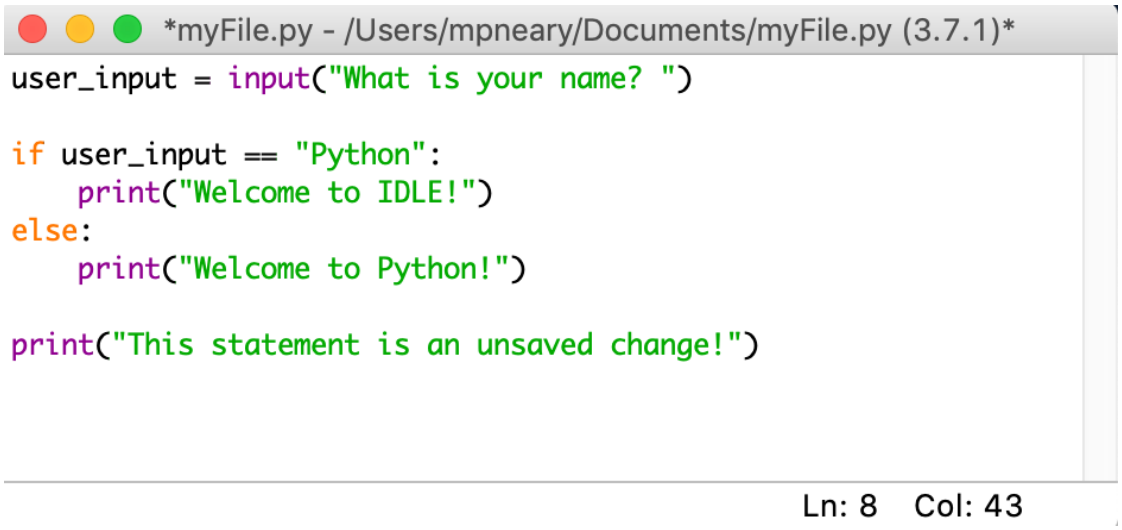


Рисунок 2.1. Середовище Python IDLE

Сильні сторони Python IDLE:

– доступність і простота використання. Не потребує додаткового встановлення чи налаштування – все готове до роботи одразу після інсталяції

Python. Інтерфейс інтуїтивно зрозумілий, що полегшує знайомство з мовою для початківців;

– редактор із підсвічуванням синтаксису. Підсвітка елементів коду дає змогу легше орієнтуватися в його структурі, що сприяє ефективнішому редагуванню;

– інтерактивний режим виконання. Дозволяє запускати фрагменти коду поетапно, що ідеально підходить для тестування і навчання;

– можливість налагодження. Підтримка покрокового виконання і виведення результатів сприяє виявленню помилок на ранніх етапах.

Обмеження Python IDLE:

– невеликий функціональний обсяг. Для реалізації складних задач і масштабних проєктів функцій Python IDLE може бути недостатньо;

– відсутність підтримки плагінів. Неможливість розширення функціоналу через модулі чи розширення обмежує його гнучкість;

– недостатня підтримка роботи з проєктами. Відсутні інструменти для структурування великого коду чи управління модулями;

– зниження продуктивності при роботі з великим обсягом даних. Ускладнює ефективну розробку складних програм.

Python IDLE доцільно використовувати для навчальних цілей або створення нескладних скриптів. Для розробки великих проєктів більш доцільно звернути увагу на сучасні IDE на кшталт PyCharm або VS Code.

PyCharm – це одне з найбільш функціональних професійних середовищ розробки, що створена компанією JetBrains і спеціалізується саме на розробці проєктів на Python (рис. 2.2).

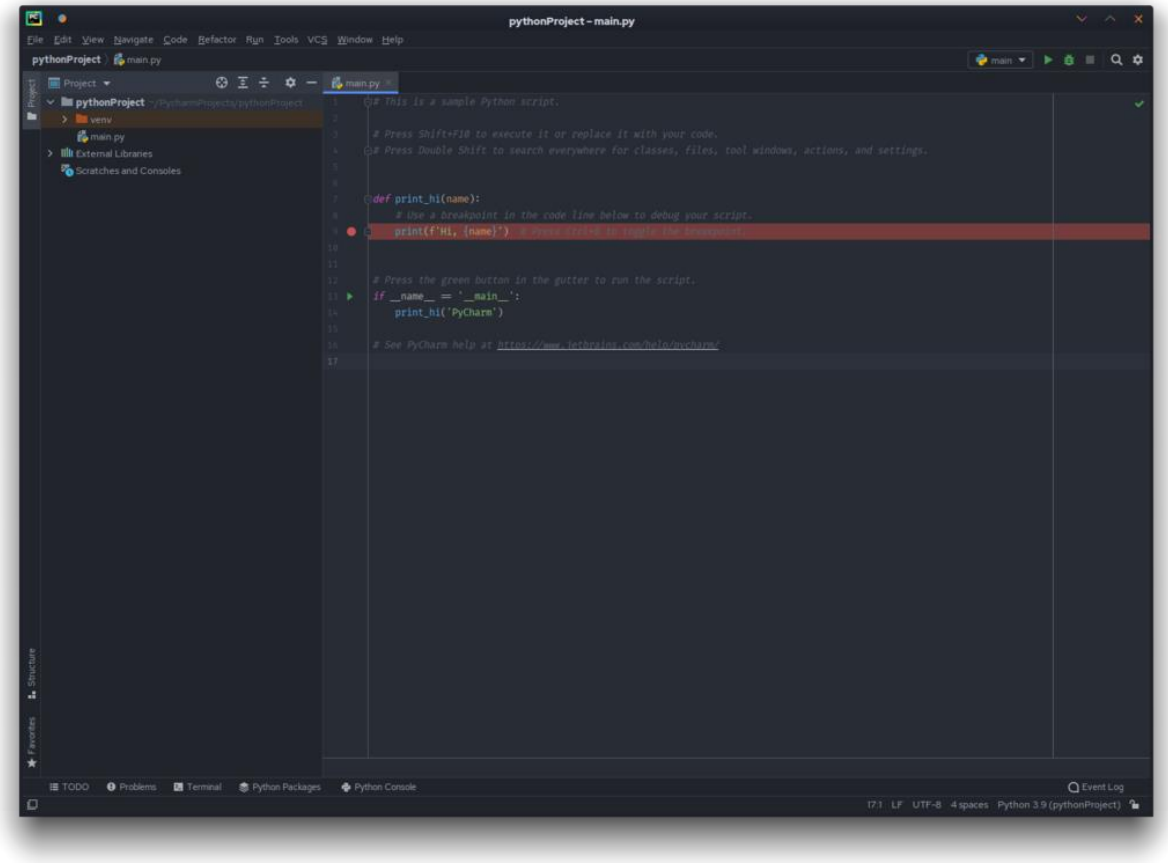


Рисунок 2.2. Середовище PyCharm

Плюси використання PyCharm:

– багатий набір функцій. Автодоповнення, рефакторинг, перевірка синтаксису, підтримка систем контролю версій, вбудований термінал – усе це значно прискорює процес розробки;

– інтелектуальний аналіз коду. IDE аналізує структуру програми, визначає можливі помилки, дублікати, зайвий код та пропонує шляхи оптимізації;

– зручний налагоджувач. Інструменти для перегляду змінних, використання точок зупину і виконання коду поетапно значно спрощують процес відлагодження;

– підтримка віртуальних середовищ. Дозволяє ізолювати залежності кожного проєкту, що усуває конфлікти між ними.

Мінуси PyCharm:

– високі вимоги до системних ресурсів. IDE є досить «важкою» і потребує потужного обладнання для стабільної роботи;

– складність для новачків. Через велику кількість інструментів і параметрів налаштування інтерфейс може здатися перевантаженим для початківців;

– наявність платної версії. Повноцінний функціонал доступний у Professional-версії, що є платною, у той час як безкоштовна Community Edition має обмеження.

PyCharm – це професійне рішення для досвідчених розробників та проєктів середнього й великого масштабу, яке забезпечує високу продуктивність і зручність роботи з кодом [14].

Visual Studio Code (VS Code) – універсальне середовище розробки, створене компанією Microsoft, яке підтримує численні мови програмування, зокрема Python. Його інтерфейс зручний і підходить навіть тим, хто тільки починає вивчати програмування (рис. 2.3).

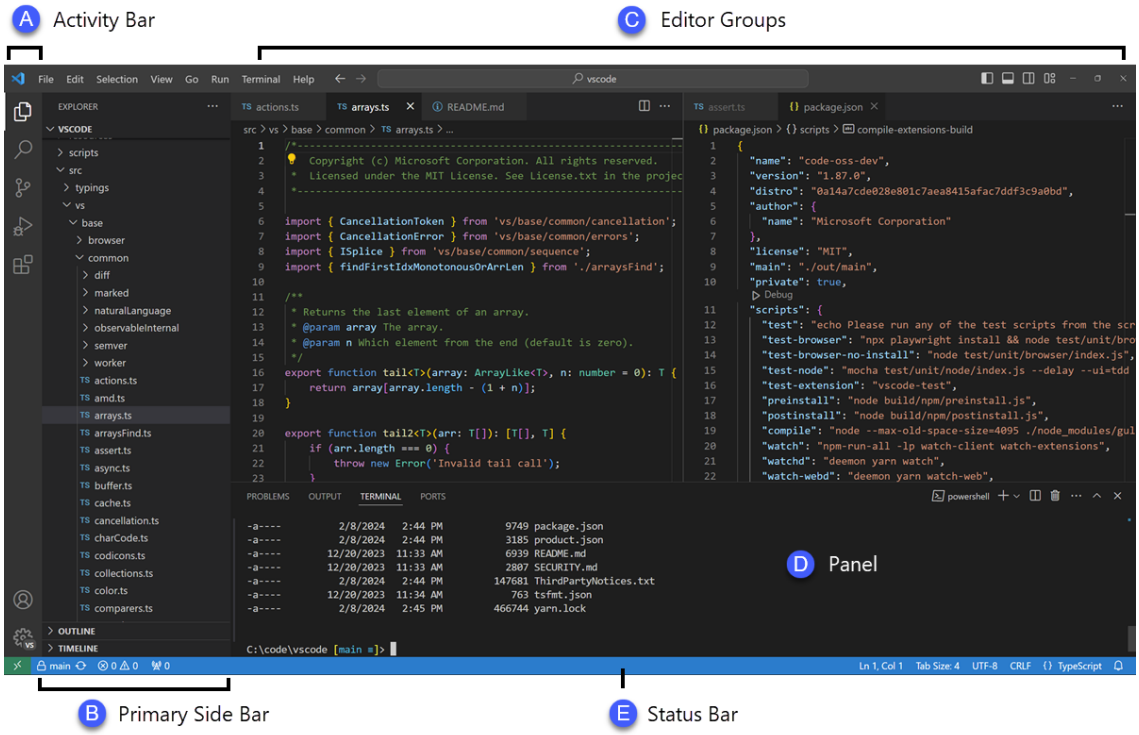


Рисунок 2.3. Інтерфейс Visual Studio Code

Переваги VS Code:

– гнучкість завдяки розширенням. Велика бібліотека плагінів дозволяє легко додавати потрібні функції під конкретні завдання;

– зрозумілий інтерфейс. Простота у використанні, зручність налаштування і редагування коду робить його дружнім навіть до початківців;

– інтеграція з системами контролю версій. Підтримка Git прямо в редакторі дозволяє керувати історією змін без сторонніх інструментів;

– інструменти налагодження. Можливість встановлювати точки зупину, переглядати змінні та виконувати код по кроках – усе це сприяє ефективному дебагінгу.

Недоліки VS Code:

– підвищене споживання ресурсів. У порівнянні з легшими редакторами, потребує більше пам’яті та обчислювальної потужності;

– необхідність конфігурації. Деякі функції працюють коректно лише після попереднього налаштування, що потребує додаткових зусиль;

– обмежений функціонал у порівнянні з повноцінними IDE. Хоча підтримує багато мов і плагінів, не завжди може конкурувати з фаховими середовищами за глибиною інтеграції;

– менша спеціалізація на Python. Порівняно з PyCharm, деякі інструменти Python підтримуються менш глибоко.

Загалом, Visual Studio Code – універсальне середовище для розробників, яке поєднує зручність, гнучкість і багатофункціональність [20].

У результаті порівняння, вибір було зупинено на PyCharm як на найповнішому за функціональністю інструменті, що найкраще відповідає вимогам і масштабам поточного проєкту.

2.2. Вибір бібліотек

Значення вибору бібліотек у реалізації програмного проєкту важко переоцінити, адже вони є основою ефективного, структурованого та швидкого процесу розробки. Застосування зовнішніх бібліотек дозволяє розробникам використовувати вже готові рішення для типових задач, що значно скорочує

час створення функціоналу та зменшує обсяг коду, який потрібно писати вручну.

Замість реалізації низькорівневих механізмів самостійно, розробник може зосередитися на логіці прикладного рівня, використовуючи перевірені часом інструменти. Таким чином, бібліотеки сприяють не лише пришвидшенню розробки, а й підвищенню надійності та стабільності коду.

Правильно підібрані бібліотеки напряму впливають на архітектурну гнучкість, масштабованість та продуктивність програмного забезпечення. Бібліотеки з активною спільнотою, регулярними оновленнями та якісною документацією значно полегшують процес інтеграції, налагодження та підтримки коду. Їх підтримка означає, що у разі виникнення труднощів можна швидко знайти приклади, відповіді або поради з реалізації потрібного функціоналу.

Крім того, вибір тієї чи іншої бібліотеки часто задає стиль організації коду та визначає структуру всього проєкту. Багато популярних бібліотек пропонують власні патерни роботи, парадигми або архітектурні підходи, що дозволяє підтримувати єдність структури в межах розробки.

Серед основних способів інсталяції бібліотек у Python-проєктах можна виокремити такі:

Використання pip – стандартного менеджера пакетів, що забезпечує завантаження та встановлення бібліотек із центрального репозиторію PyPI. Наприклад: pip install ім'я_бібліотеки [10].

Застосування інструментів pipenv або poetry, що дозволяють створювати ізольовані середовища для окремих проєктів та управляти залежностями через спеціальні конфігураційні файли (наприклад, Pipfile або pyproject.toml).

Використання пакетних менеджерів операційної системи, таких як apt, yum або brew, що часто зручно для встановлення системних бібліотек або тих, що мають зовнішні залежності.

Інтегровані засоби IDE, які дозволяють встановлювати бібліотеки безпосередньо через графічний інтерфейс середовища розробки, що особливо корисно для початківців.

У даному проєкті було використано набір бібліотек, що забезпечує ефективну обробку файлів Excel, побудову веб-інтерфейсу API, обробку HTTP-запитів і відповідей, а також збереження результатів у форматі JSON. Зокрема, були додані наступні модулі:

- **fastapi** – фреймворк для побудови високопродуктивних веб-API. Він дозволяє швидко створювати REST-сервіси з мінімальними зусиллями завдяки використанню аннотацій типів, автоматичній генерації документації та інтеграції з асинхронним програмуванням;
- **UploadFile, File, HTTPException**, що є частиною FastAPI – використовуються для реалізації завантаження файлів користувачами, обробки виняткових ситуацій і передачі файлів через API;
- **JSONResponse** – застосовується для форматування відповідей сервера у вигляді JSON-структур, що є стандартом взаємодії у веб-застосунках;
- **openpyxl** – бібліотека для читання та редагування файлів формату Excel (.xlsx), яка дозволяє працювати з таблицями без необхідності запуску офісних додатків [16];
- **datetime** – модуль стандартної бібліотеки Python, який дозволяє працювати з датами та часом, зокрема для позначення моментів обробки файлів чи логування;
- **json** – стандартна бібліотека для серіалізації та десеріалізації даних у форматі JSON, що забезпечує зберігання результатів обробки у вигляді файлів;
- **os** – використовується для роботи з файловою системою, зокрема для перевірки наявності файлів, створення директорій, шляхів тощо [17];
- **re** – бібліотека для роботи з регулярними виразами, що дозволяє здійснювати перевірку, пошук і обробку текстових даних за певними шаблонами;
- **typing.List** – модуль типізації, який дозволяє зазначити, що певна функція працює з переліком об’єктів, забезпечуючи ясність у структурах даних.

Ці бібліотеки були обрані з урахуванням специфіки завдання – обробки Excel-документів з подальшим аналізом вмісту, формуванням структурованого виводу у форматі JSON та побудовою асинхронного веб-інтерфейсу для взаємодії з користувачами через API. Їх використання дало змогу досягти високої продуктивності, гнучкості та зручності розгортання всього проекту.

РОЗДІЛ 3
СТВОРЕННЯ ПАРСЕРУ ТА БАЗИ ДАНИХ, РОЗРОБКА API

3.1. Парсинг даних

Процес парсингу даних, або ж їх аналізу та трансформації, є невід'ємною складовою при роботі з великими обсягами інформації, особливо коли дані надходять у неструктурованому чи частково структурованому вигляді, наприклад, з Excel-файлів, веб-ресурсів або API. Сутність парсингу полягає в автоматичному вилученні потрібних даних з джерел, їх підготовці до подальшої обробки, а також адаптації до потреб певного завдання чи системи.

До ключових етапів парсингу можна віднести:

- вилучення інформації (екстракція) з початкового файлу або джерела;
- очищення даних від зайвих елементів, помилок або повторів;
- перетворення структури чи формату даних у вигляд, придатний для аналізу або збереження;
- перевірку на відповідність критеріям (валідацію), що дозволяє виявити й усунути помилки ще на ранньому етапі обробки.

Застосування інструментів автоматизованого парсингу дозволяє значно скоротити час, витрачений на ручну обробку, та підвищити точність і достовірність результатів, що є критичним у системах, де необхідно опрацювати сотні або тисячі рядків інформації.

У мікро-сервісі контингенту студентів реалізовано функціонал для завантаження, парсингу та збереження інформації про студентів із Excel-файлу формату .xlsx. Процес парсингу передбачає наступні етапи:

3.1.1. Завантаження файлу

Користувач надсилає файл через HTTP POST-запит на ендпоінт /upload_excel/.

Сервер перевіряє формат файлу (повинен бути .xlsx) і тимчасово зберігає його на диск.

3.1.2. Парсинг Excel-файлу

Файл відкривається за допомогою бібліотеки openpyxl.

Для кожного аркуша у книзі:

- зчитується перший рядок як **заголовки стовпців**;
- решта рядків опрацьовується як дані;
- створюється словник entry, де ключами є назви колонок, а значеннями – дані з відповідного рядка.

3.1.3. Обробка типів даних

Якщо значення є датою (тип datetime), воно перетворюється у формат рядка "YYYY-MM-DD".

Для кожного запису перевіряється, чи належить його колонка до **необхідного набору полів** (встановлено вручну як required_columns).

3.1.4. Нормалізація назв груп

Значення поля **"Спеціальність"** нормалізується через функцію normalize_group_name():

- вилучається код та назва спеціальності;
- з назви генерується абревіатура;
- назва групи формується як <код>_<абревіатура>_<курс>.

3.1.5. Очищення та оновлення БД

Перед збереженням нових даних таблиця students видаляється та створюється заново.

Кожен студентський запис перетворюється у об’єкт моделі Student та додається до сесії SQLAlchemy .

Сесія комітиться, що зберігає всі записи в PostgreSQL.

У реалізації **мікросервісу розподілу ставок** для обробки навчальних планів було застосовано бібліотеку openpyxl, яка спеціалізується на роботі з файлами Excel формату .xlsx. Цей інструмент дає змогу не лише відкривати і читати дані з аркушів, але й модифікувати таблиці, додавати чи видаляти рядки, налаштовувати стилі, працювати з формулами та навіть створювати графіки.

Зокрема, у функціоналі розробленого застосунку було реалізовано:

- автоматичне визначення курсу навчання шляхом обчислення різниці між поточним роком та роком, вказаним у назві Excel-файлу (функція extract_year_from_filename);
- фільтрацію рядків таблиці за відповідністю семестру до поточного курсу (get_semesters_for_course);
- обчислення загального навантаження викладача за визначеними коефіцієнтами, залежно від типу заняття (лекції, лабораторні, практика, курсова, тощо);
- створення зведених звітів у вигляді .json файлів: основні результати, загальна сума навантаження, та журнал логів;
- виведення структурованої відповіді клієнту через FastAPI, включаючи повідомлення про успішну обробку, шлях до збережених файлів та логів для перегляду.

3.2. База даних

База даних (БД) – це впорядкована система зберігання інформації, яка дозволяє ефективно організовувати, оновлювати та обробляти великі обсяги взаємопов’язаних даних. Такі системи найчастіше застосовуються для веб-

ресурсів із динамічним контентом і значними обсягами інформації, наприклад: електронна комерція, корпоративні портали або інформаційні сайти.

БД можна визначити як сукупність даних, які мають спільну ознаку або характеристику, й упорядковані певним чином, наприклад, за алфавітним принципом. Об'єднання різноманітної інформації в єдину базу відкриває широкі можливості для її групування: від особистих відомостей користувача до історії його замовлень, переліку товарів, прайс-листів тощо.

Основна перевага використання баз даних полягає у високій швидкодії доступу до інформації. За допомогою спеціалізованих алгоритмів пошуку можна оперативно знайти потрібні записи – буквально за кілька секунд. Крім того, дані у базах мають логічні зв’язки: оновлення одного елемента може спричинити зміни в інших, що спрощує керування інформацією та пришвидшує її обробку.

Вебсайти активно застосовують бази даних для збереження пов’язаних між собою таблиць із критично важливою інформацією, як-от дані клієнтів, товарні позиції, описи, наявність, ціни та інші ресурси, необхідні для функціонування системи.

Для взаємодії з базою даних зазвичай використовується мова структурованих запитів (SQL – Structured Query Language), яка дозволяє додавати, змінювати та вилучати записи в таблицях. У процесі створення веб-застосунків застосовують різноманітні системи управління базами даних (**СУБД**). До найпоширеніших належать:

- об'єктно-реляційна система Oracle Database;
- вільна система PostgreSQL;
- комерційна Microsoft SQL Server;
- відкрита MySQL.

PostgreSQL – це високофункціональна об’єктно-реляційна система з відкритим кодом, яка розширює стандарт SQL та забезпечує надійне зберігання складних даних. Її розвиток розпочався ще у 1986 році в межах проекту

POSTGRES у Каліфорнійському університеті в Берклі, що забезпечило понад три десятиліття постійного вдосконалення.

Свою популярність PostgreSQL здобула завдяки стійкій архітектурі, високій надійності, точності в роботі з даними, гнучкій системі розширення функціоналу та підтримці потужної спільноти розробників. Вона підтримує всі основні ОС, відповідає стандартам ACID з 2001 року та має численні розширення, зокрема відомий модуль геоданих PostGIS. PostgreSQL є провідним вибором серед відкритих реляційних СУБД для індивідуального та корпоративного використання.

Система має великий арсенал інструментів, що сприяють створенню надійних додатків: вона дозволяє адміністраторам забезпечувати збереження даних, підтримувати безперебійну роботу сервісів, а розробникам – будувати гнучкі рішення для будь-якого масштабу. Крім того, PostgreSQL є безкоштовною, підтримує відкритий код і чудово масштабується. Вона дозволяє створювати власні типи даних, реалізовувати функції, інтегрувати код різними мовами програмування без потреби перекомпіляції ядра БД [18].

Системи управління базами даних (СУБД) – це спеціалізовані програмні комплекси, що забезпечують інтерфейс між користувачем і самою базою даних. Завдяки СУБД можлива побудова, супровід і спільне використання БД. Типова структура сучасної СУБД охоплює ядро, обробник запитів, підсистему виконання та набір сервісних утиліт, які розширюють її можливості й забезпечують обслуговування інформаційних систем [11].

3.3. Опис API

У рамках реалізації програмного рішення було розроблено два незалежні мікро-сервіси, які забезпечують автоматизовану обробку табличних даних з навчальними відомостями у форматі Excel. Основне призначення цих сервісів – перетворення вхідних файлів на структуровану інформацію, з подальшим збереженням її в базі даних і можливістю взаємодії через RESTful API.

3.3.1. Мікро-сервіс обробки студентських списків

Перший мікро-сервіс реалізує механізм приймання, обробки та збереження у базу даних інформації про студентів. Основні завдання цього компонента – обробка вхідних Excel-документів, вилучення з них структурованої інформації про студентів, а також класифікація даних за групами та навчальними курсами.

3.3.2. Функціональні можливості API:

Завантаження файлу зі списком студентів:

Користувач передає файл Excel через HTTP-запит. Система зчитує дані з таблиці, ідентифікуючи поля з прізвищами, іменами, по батькові, номерами залікових книжок, назвою групи тощо.

Аналіз та валідація даних:

Дані проходять попередню перевірку на коректність – перевіряється наявність усіх обов’язкових полів, відповідність форматів, а також можливі повтори чи аномалії.

Збереження у базу даних:

Після обробки та перевірки, інформація зберігається в базі даних PostgreSQL за допомогою ORM-бібліотеки SQLAlchemy. Записи групуються за ідентифікаторами курсів та академічних груп, що дозволяє в подальшому ефективно здійснювати фільтрацію.

Надання структурованої відповіді:

У відповідь API повертає JSON-об’єкт із деталізованими даними: кількість опрацьованих рядків, перелік нових студентів, груп і курсів, а також можливі помилки вхідних даних.

Збереження результатів:

Для зручності результати обробки дублюються у файл у форматі JSON, який можна зберігати локально або завантажити повторно.

Основні маршрути:

POST /upload_excel/

Завантаження .xlsx-файлу з даними студентів. Відбувається:

– зчитування вмісту файлу;

– валідація необхідних колонок;

– конвертація дат;

– очистка таблиці students;

– збереження нових даних у БД.

GET /students/

Отримання списку всіх студентів.

GET /students/filter/

Фільтрація студентів за будь-яким набором параметрів (ім’я, стать, курс тощо).

GET /groups

Отримання списку всіх унікальних груп (нормалізованих).

GET /students/group/{group_name}

Отримання списку студентів конкретної групи (ID та ім’я).

GET /student/{id_card}

Отримання повної інформації про студента за його ID-карткою.

GET /allYears

Отримання всіх унікальних курсів (років навчання).

GET /groups/{course}

Отримання структурованого списку груп і студентів певного курсу.

3.3.3. Мікросервіс обробки навчальних планів

Другий сервіс призначений для обробки даних навчальних планів, які містять дисципліни, види контролю, кількість годин, семестрове навантаження, а також інші параметри, що стосуються освітнього процесу. Вхідними є файли формату Excel, що відповідають визначеній структурі.

3.3.4. Призначення та завдання API:

Зчитування файлів навчального плану:

Після надсилання користувачем відповідного Excel-документа, API розпізнає структуру таблиці, виявляє назви дисциплін, розподіл годин за семестрами, види занять (лекції, практики, лабораторні), форму контролю та кількість кредитів ECTS.

Визначення курсу на основі семестру:

Для кожного запису система розраховує, до якого навчального курсу відноситься дисципліна, виходячи з семестру (наприклад, 1-2 семестри – перший курс, 3-4 – другий тощо).

Розрахунок навантаження:

На основі кількості студентів та значень у відповідних стовпцях розраховується загальне навчальне навантаження за формулою, яка враховує кількість аудиторних годин і вид занять.

Фільтрація зайвих рядків:

Пропускаються порожні рядки, службові заголовки або дані, що не відповідають навчальному змісту. Це дозволяє зосередитись на ключовій інформації без зайвих втрат продуктивності.

Виведення результатів у JSON:

API формує структуровану відповідь (рис. 3.1), у якій вказується перелік дисциплін, відповідні семестри, курс, години, а також загальне навантаження на викладачів.

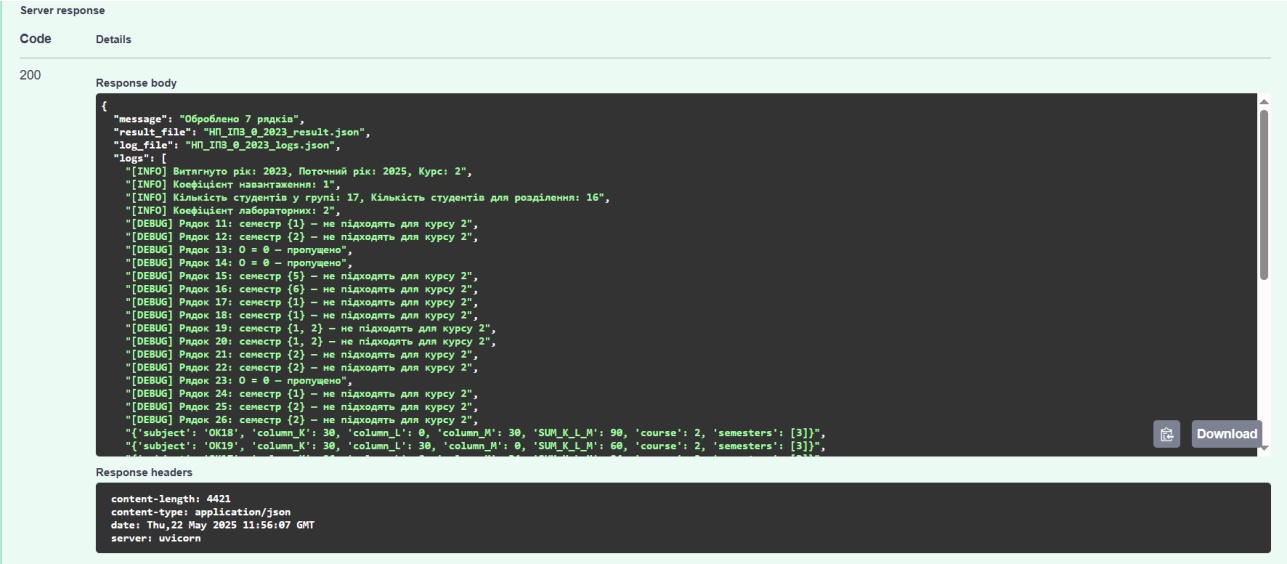


Рисунок 3.1. Відповідь на запит

Збереження JSON-файлу:

Зібрані дані паралельно зберігаються у зовнішньому JSON-файлі, який може бути використаний для подальшої обробки або архівування.

Ключові маршрути:

POST /upload-plan/ – завантаження та обробка Excel-файлу з навчальним планом;

GET /plan/ – повертає детальну структуру навантаження;

GET /subjects/ – перелік дисциплін із зазначенням курсу та семестру;

GET /hours-summary/ – підсумкове навантаження.

Дані зберігаються у змінних stored_plan_data, subjects_data, summary_data, і використовуються в GET-запитах.

3.3.5. Особливості реалізації API обох сервісів:

Обидва мікро-сервіси реалізовані з використанням фреймворку FastAPI, що забезпечує високу продуктивність, зручність у створенні документації

(через інтеграцію з Swagger UI) [19], а також асинхронну обробку запитів. API побудовано у відповідності до REST-архітектури, що полегшує взаємодію з фронтендом або іншими системами. Дані зберігаються в PostgreSQL, з використанням ORM-інструменту SQLAlchemy для моделювання та взаємодії з таблицями [12].

Мікро-сервісна архітектура дозволяє масштабувати окремі частини системи незалежно, а також спрощує тестування, супровід і розширення функціональності в майбутньому.

3.4. Програмна реалізація

3.4.1. Мікро-сервіс 1: Обробка інформації про студентів

Цей мікро-сервіс надає інтерфейс для керування та аналізу студентських даних, зосереджуючись на збереженні, фільтрації та отриманні інформації за допомогою REST API (рис. 3.2). Він підтримує завантаження Excel-файлів, з яких дані автоматично витягуються та зберігаються до реляційної бази даних PostgreSQL.

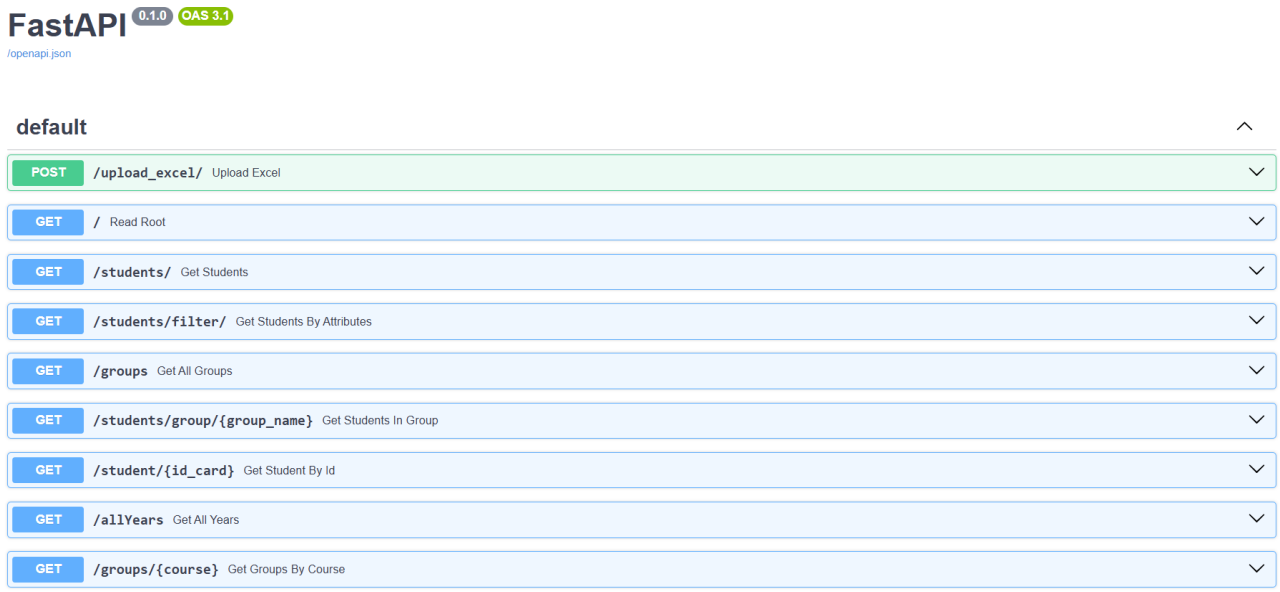


Рисунок 3.2. Swagger: API документація мікро-сервіс 1

1. Завантаження Excel-файлу з інформацією про студентів

Маршрут: /upload_excel/

Метод: POST

Призначення: приймає .xlsx-файл, перевіряє його структуру, обробляє вміст, перетворює на структурований формат та зберігає кожен запис у таблицю students. Неправильні або пошкоджені файли повертають повідомлення про помилку. (рис. 3.3)

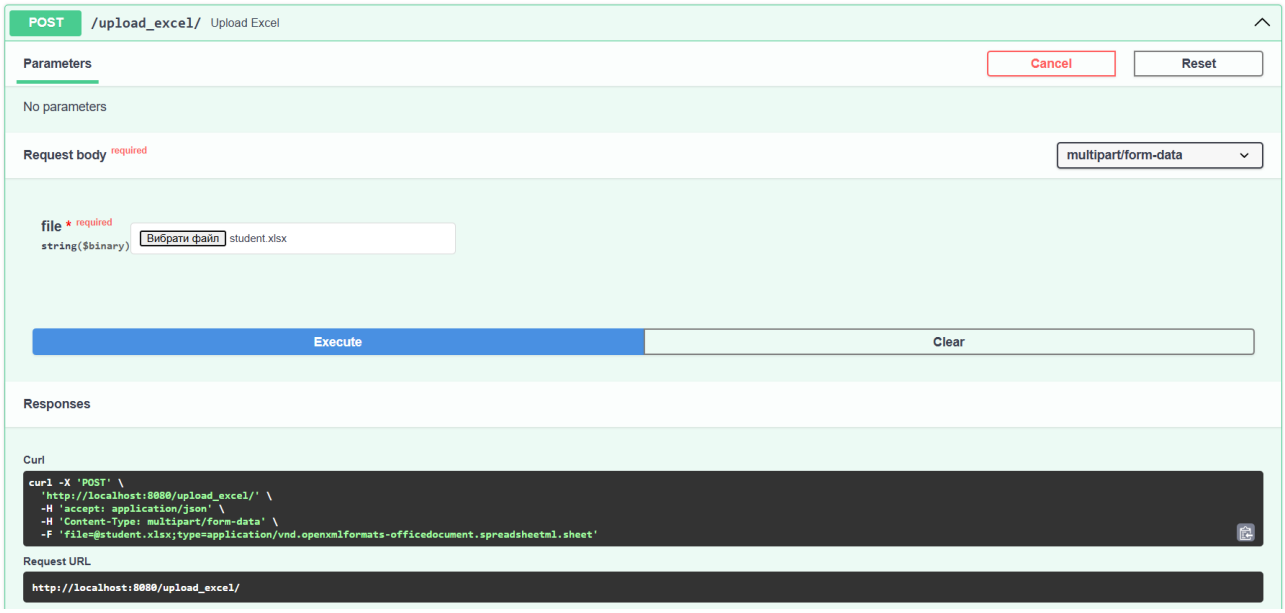


Рисунок 3.3. Запит завантаження Excel-файлу

2. Отримання повного списку студентів

Маршрут: /students/

Метод: GET

Опис: повертає перелік усіх записів студентів із бази даних. (рис. 3.4)

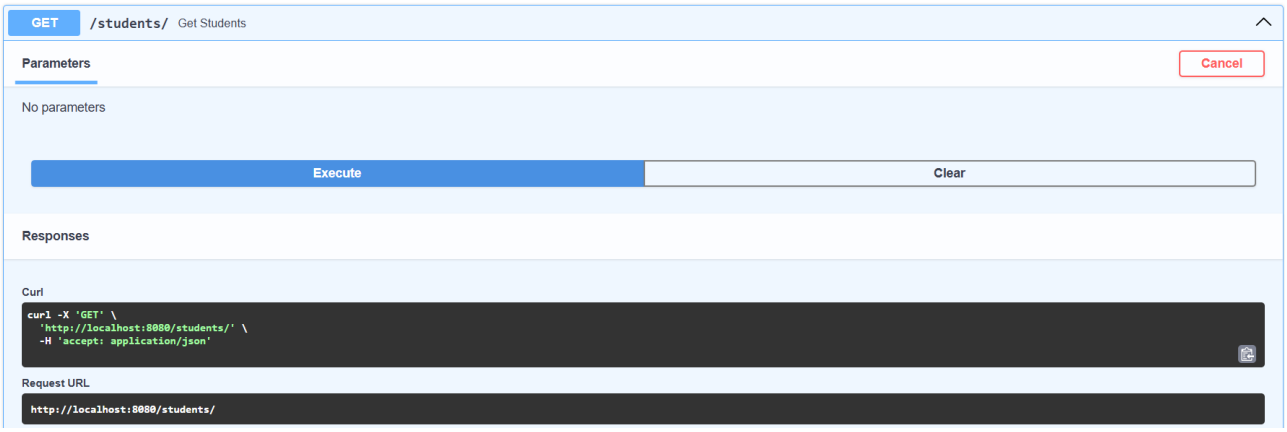


Рисунок 3.4. Запит отримання списку студентів

3. Пошук студентів за різними параметрами

Маршрут: /students/filter/

Метод: GET

Призначення: дозволяє знаходити студентів, використовуючи будь-яке поле – ПІБ, спеціальність, освітній рівень, дату народження тощо. (рис. 3.5)

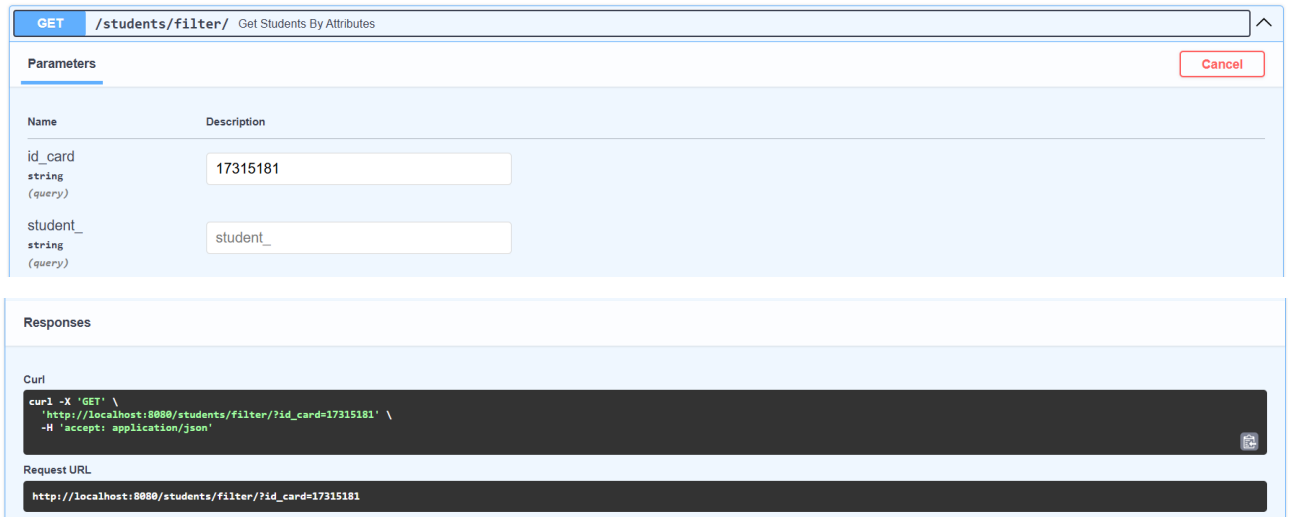


Рисунок 3.5. Запит пошуку студентів за фільтрами

4. Перегляд усіх доступних груп

Маршрут: /groups

Метод: GET

Опис: повертає список усіх унікальних назв спеціальностей, які використовуються як ідентифікатори груп студентів. (рис. 3.6)

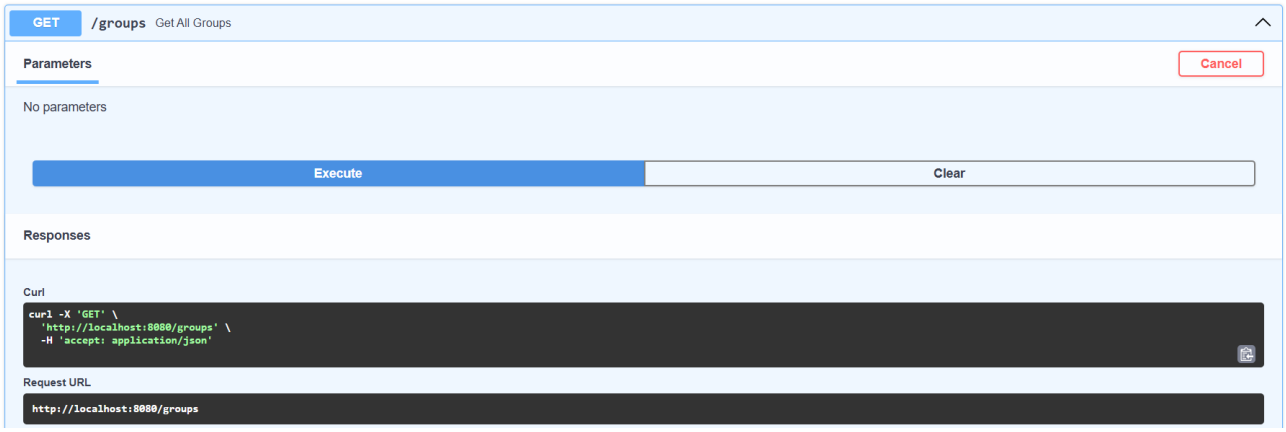


Рисунок 3.6. Запит на список груп

5. Отримання студентів певної академічної групи

Маршрут: /students/group/{group_name}

Метод: GET

Опис: повертає ідентифікатори студентів та їх ПІБ для вказаної групи (рис. 3.7)

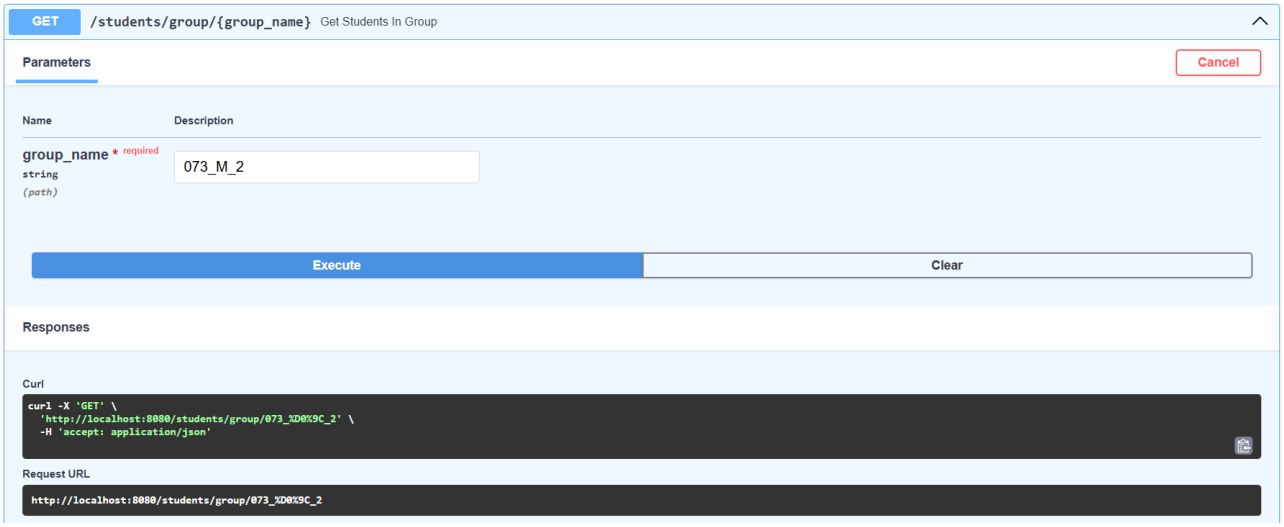


Рисунок 3.7. Запит на список студентів окремої групи

6. Інформація про студента за номером картки

Маршрут: /student/{id_card}

Метод: GET

Призначення: надає повну інформацію щодо конкретного студента. Якщо запис не знайдено – повертається повідомлення про помилку 404. (рис. 3.8)

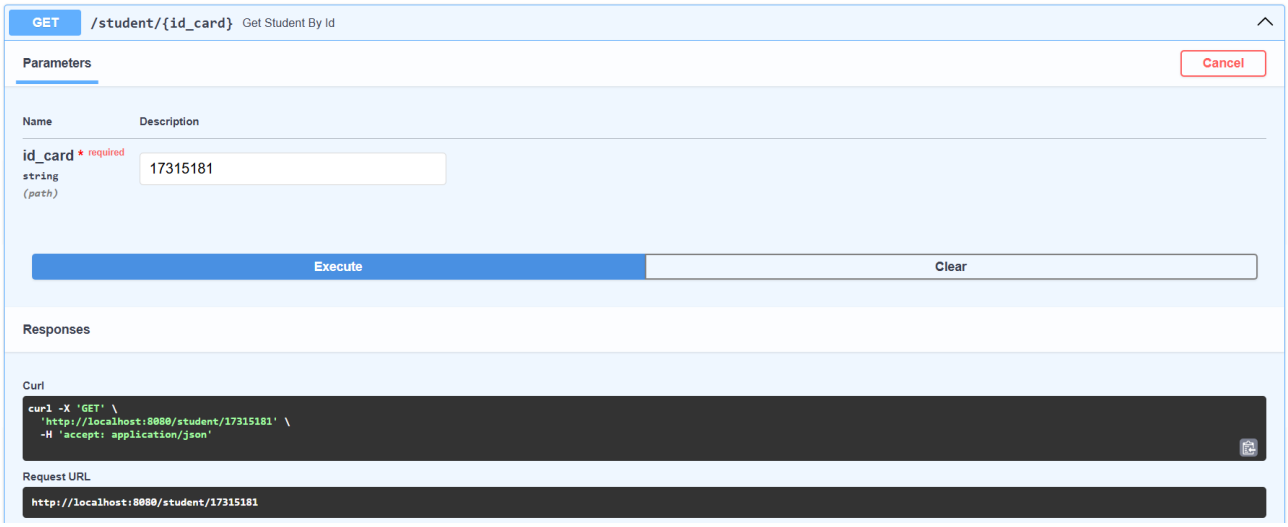


Рисунок 3.8. Запит пошуку студента за номером картки

7. Отримання студентів певного курсу

Маршрут: /students/course/{course}

Метод: GET

Опис: фільтрує студентів за курсом навчання. У випадку відсутності записів — повертає відповідне повідомлення про відсутність результатів. (рис. 3.4)

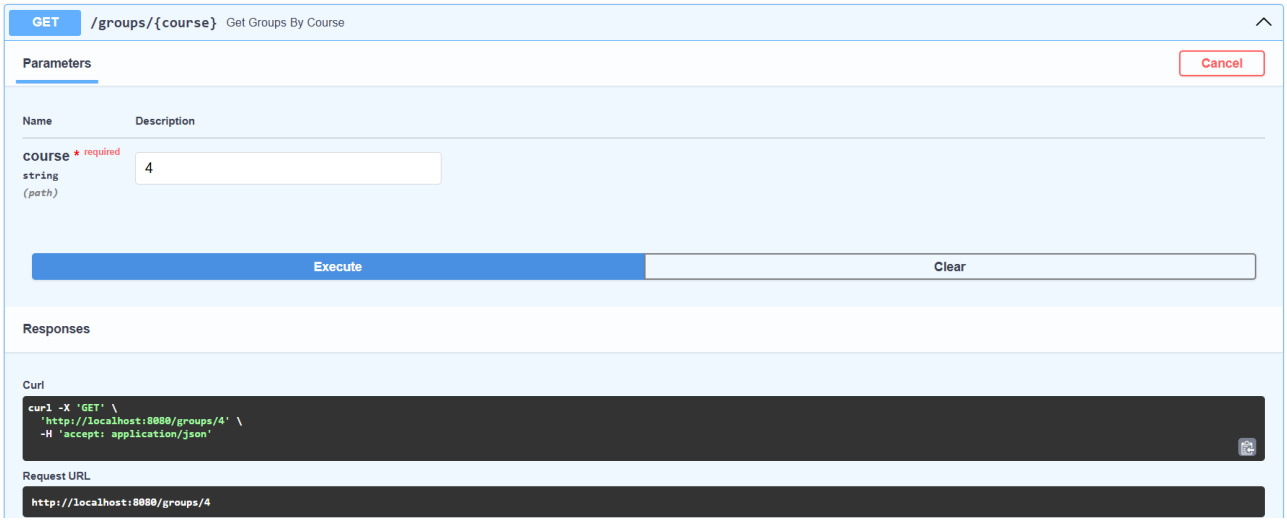
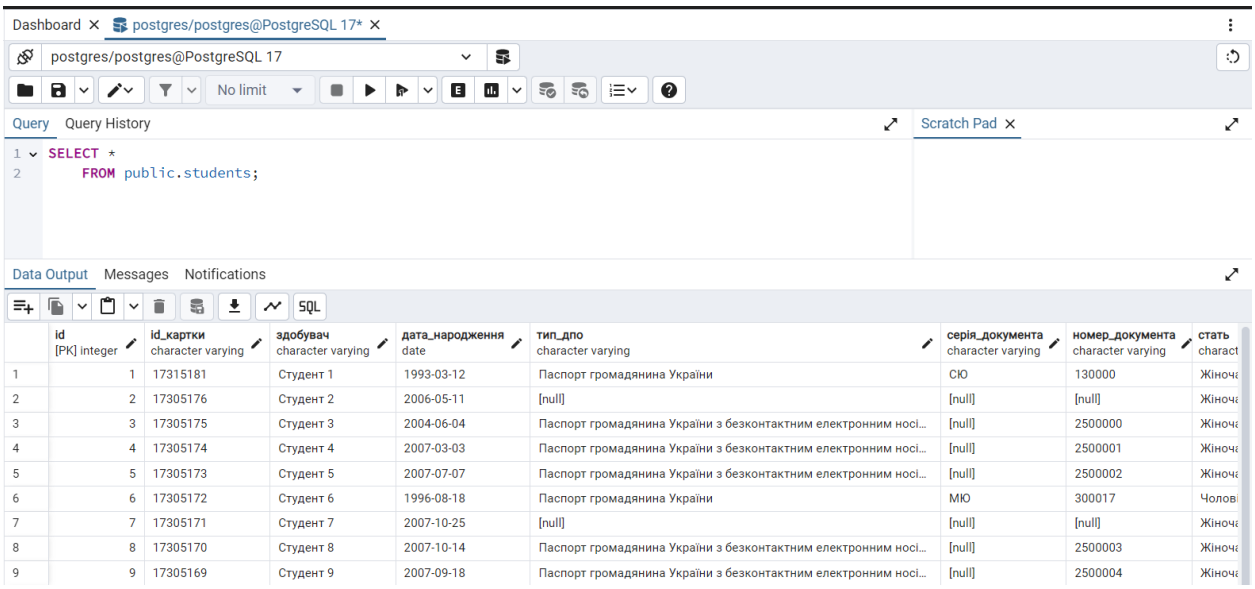


Рисунок 3.9. Запит пошуку студентів певного курсу

Структура таблиці students

Таблиця зберігає повний набір даних для кожного студента, включаючи особисті дані, академічні дані, а також інформацію про його навчання та документи (рис. 3.10).



	id [PK] integer	id_картки character varying	здобувач character varying	дата_народження date	тип_дпо character varying	серія_документа character varying	номер_документа character varying	стать
1	1	17315181	Студент 1	1993-03-12	Паспорт громадянина України	СЮ	130000	Жіноча
2	2	17305176	Студент 2	2006-05-11	[null]	[null]	[null]	Жіноча
3	3	17305175	Студент 3	2004-06-04	Паспорт громадянина України з безконтактним електронним носі...	[null]	2500000	Жіноча
4	4	17305174	Студент 4	2007-03-03	Паспорт громадянина України з безконтактним електронним носі...	[null]	2500001	Жіноча
5	5	17305173	Студент 5	2007-07-07	Паспорт громадянина України з безконтактним електронним носі...	[null]	2500002	Жіноча
6	6	17305172	Студент 6	1996-08-18	Паспорт громадянина України	МЮ	300017	Чолові
7	7	17305171	Студент 7	2007-10-25	[null]	[null]	[null]	Жіноча
8	8	17305170	Студент 8	2007-10-14	Паспорт громадянина України з безконтактним електронним носі...	[null]	2500003	Жіноча
9	9	17305169	Студент 9	2007-09-18	Паспорт громадянина України з безконтактним електронним носі...	[null]	2500004	Жіноча

Рисунок 3.10. SELECT запит таблиці students

Поля таблиці students:

- id (Integer, первинний ключ): унікальний ідентифікатор запису для кожного студента в таблиці;
- id_картки (String): ідентифікатор картки студента, який використовується для унікальної ідентифікації в системі;
- здобувач (String): ім'я студента;
- дата_народження (Date): дата народження студента;
- тип_дпо (String): тип документа про попередню освіту (ДПО);
- серія_документа (String): серія документа про освіту;
- номер_документа (String): номер документа про освіту;
- стать (String): стать студента;
- громадянство (String): громадянство студента;
- піб_англійською (String): ПІБ студента англійською мовою;
- рнокпп (String): ідентифікаційний код (РНОКПП) студента;
- початок_навчання (Date): дата початку навчання;
- завершення_навчання (Date): дата завершення навчання;

- структурний_підрозділ (String): назва структурного підрозділу (факультет, інститут);
- освітній_ступінь (String): освітній ступінь або рівень студента (наприклад, бакалавр, магістр);
- вступ_на_основі (String): інформація про підставу вступу (наприклад, свідоцтво про повну загальну середню освіту, диплом молодшого спеціаліста);
- форма_навчання (String): форма навчання (денна, заочна тощо);
- спеціальність (String): спеціальність студента (наприклад, інженерія програмного забезпечення, економіка);
- спеціалізація (String): спеціалізація студента в межах спеціальності;
- id_оп (String): ідентифікатор освітньої програми;
- освітня_програма (String): назва освітньої програми, на якій навчається студент;
- курс (String): курс студента (наприклад, 1-й, 2-й курс);
- наказ_про_зарахування (String): номер наказу про зарахування студента;
- документ_про_освіту: відомості документа про освіту.

Цей мікро-сервіс забезпечує автоматизовану обробку та доступ до даних здобувачів освіти, що критично важливо для електронного документообігу навчального закладу.

3.4.2. Мікро-сервіс 2: Аналіз навчального плану

Цей сервіс відповідає за обробку файлів навчальних планів і розрахунок загального навантаження викладачів з урахуванням кількості студентів у групі, типу занять та формули розподілу годин. (рис. 3.11)

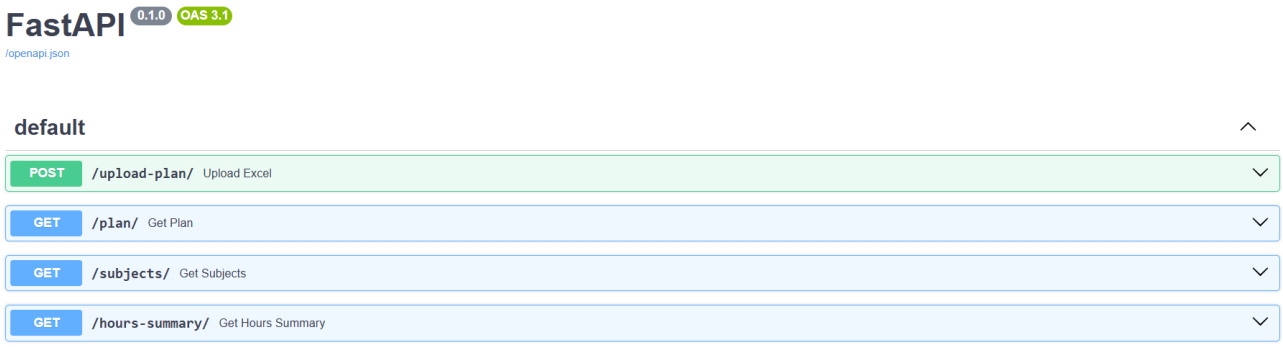


Рисунок 3.11. Swagger: API документація мікро-сервіс 2

1. Завантаження плану навчання

Маршрут: /upload-plan/

Метод: POST

Опис: приймає .xlsx-файл навчального плану, зчитує дані про предмети, семестри, кількість годин та розподіляє їх відповідно до заданих формул. Дані тимчасово зберігаються у пам’яті або у JSON-файлі для подальшої обробки. (рис. 3.12)

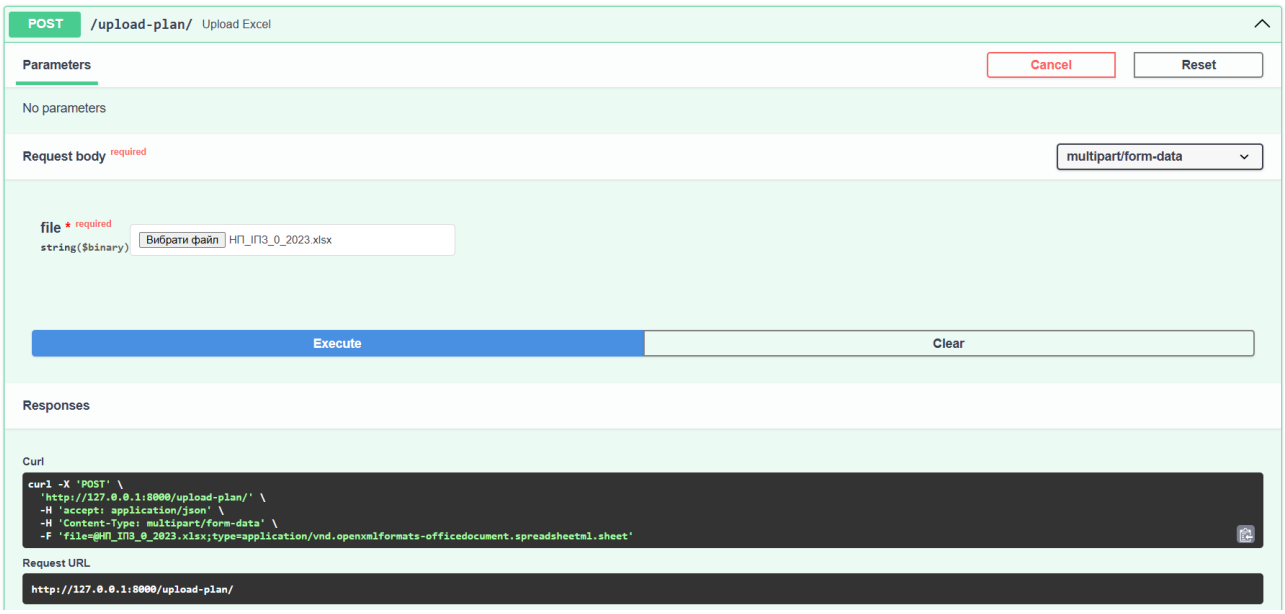


Рисунок 3.12. Запит завантаження НП

2. Перегляд обробленого навчального навантаження

Маршрут: /plan/

Метод: GET

Опис: виводить список дисциплін з деталізацією по семестрах для кожного навчального року. (рис. 3.13)

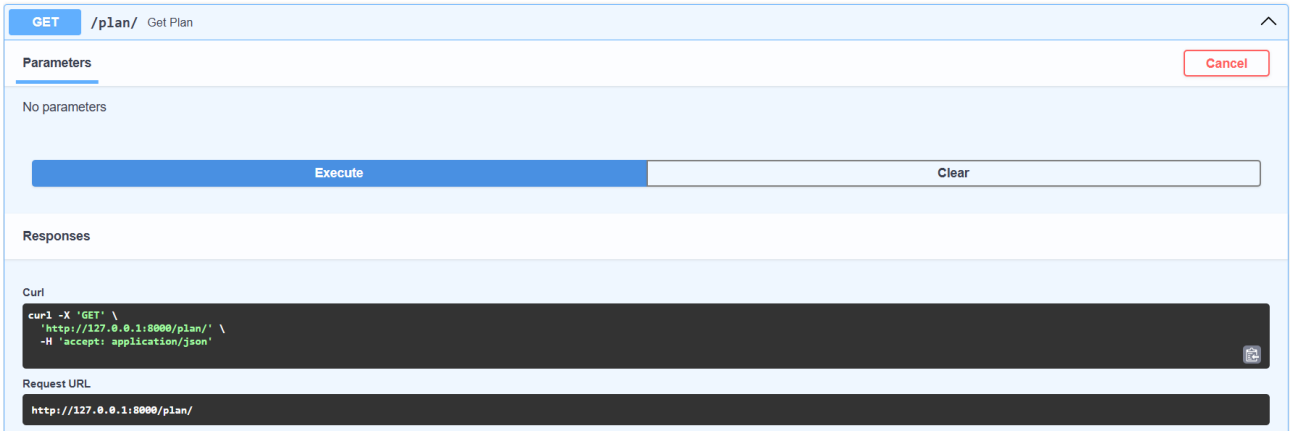


Рисунок 3.13. Запит перегляду обробленого НП

3. Отримання списку унікальних предметів

Маршрут: /subjects/

Метод: GET

Опис: повертає перелік усіх предметів, що присутні у плані, без дублювань. (рис. 3.14)

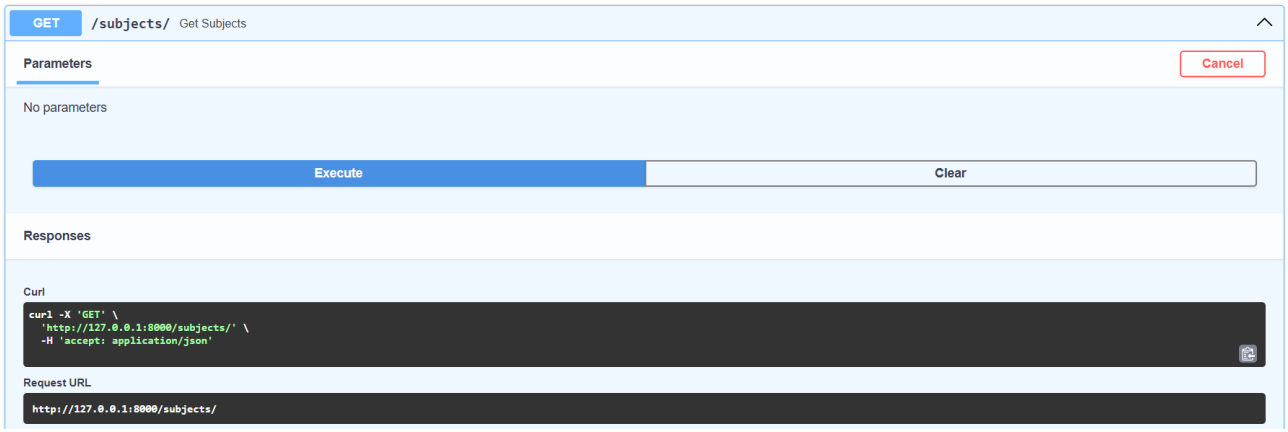


Рисунок 3.14. Запит на список предметів у НП

4. Узагальнений підрахунок годин з урахуванням кількості студентів

Маршрут: /hours-summary/

Метод: GET

Призначення: обчислює загальне навантаження за всіма видами занять, використовуючи спеціальну формулу з коефіцієнтами, прив’язаними до кількості студентів. (рис. 3.15)

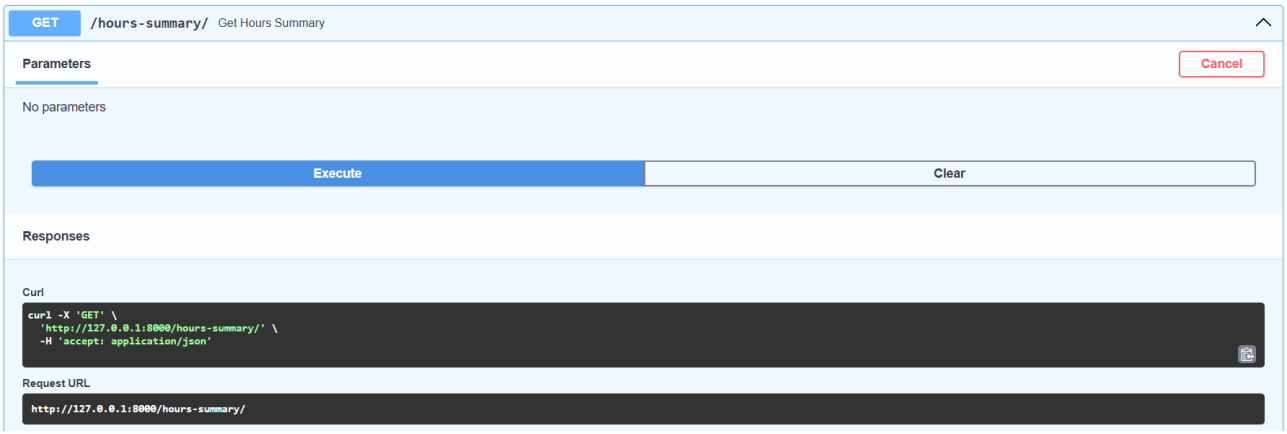


Рисунок 3.15. Запит на виведення результатів підрахунку

Особливості реалізації:

Сервіс підтримує фільтрацію даних за курсами.

Використовується формула на кшталт: N * K, де N – базова кількість годин, а K – коефіцієнт, залежний від чисельності студентів.

Цей мікро-сервіс дозволяє адміністрації та викладачам оперативно отримувати аналітику за навчальним планом та оцінювати навантаження відповідно до кількісного складу академічних груп.

ВИСНОВКИ

У рамках проєкту було реалізовано низку мікро-сервісів, спрямованих на автоматизацію процесів управління студентськими даними та розподілу педагогічного навантаження, що дозволило досягти ключових цілей системної інтеграції та обробки освітньої інформації:

- ❖ проведено глибокий аналіз вимог до підсистем, що мають забезпечувати облік студентів та розрахунок ставок викладачів на основі навчальних планів і кількості здобувачів;
- ❖ створено окремі мікросервіси, які відповідають за:
  - – імпорт, обробку і фільтрацію даних про студентів з Excel-документів із подальшим збереженням у базі даних;
  - – обробку планів навчання для визначення курсу навчання та обрахунку навчального навантаження з урахуванням кількості студентів, що дає змогу ефективно формувати розподіл ставок для викладачів;
- ❖ спроєктовано та реалізовано зручні API-інтерфейси, що забезпечують взаємодію з фронтендом і дозволяють виконувати такі операції, як завантаження файлів, пошук, фільтрація, доступ до груп, курсів та окремих студентів, а також отримання обрахованих ставок за конкретними групами;
- ❖ реалізовано логіку розрахунку педагогічного навантаження з урахуванням кількості студентів та характеристик дисциплін (кількість годин, тип занять, семестри проведення тощо), що забезпечує прозорість і об’єктивність у формуванні ставок;
- ❖ забезпечено масштабованість і гнучкість структури за рахунок мікро-сервісної архітектури, що дозволяє розвивати систему шляхом додавання нових сервісів без порушення загальної цілісності;
- ❖ виконано тестування взаємодії мікро-сервісів із зовнішніми модулями та базами даних, що дозволило досягти стабільної роботи та коректної обробки інформації.

Таким чином, розроблені мікро-сервіси демонструють ефективність підходу до децентралізованої обробки навчальних і адміністративних даних. Їх застосування в закладах вищої освіти дозволить не лише автоматизувати рутинні процеси, але й значно підвищити точність, контроль та об’єктивність в управлінні кадровим і навчальним навантаженням.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вронський М. О., Панченко І. В. Впровадження електронного документообігу в кадрову роботу закладу вищої освіти. Вісник студентського наукового товариства ДонНУ імені Василя Стуса. Том 2 № 14. 2022. С. 209-212 (дата звернення: 21.04.2025).
2. «Деякі питання документування управлінської діяльності» : Постанова від 01.12.2022 № 55-2018-п. URL: https://zakon.rada.gov.ua/laws/show/55-2018-п (дата звернення: 25.04.2025).
3. Інформаційна система – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Інформаційна_система (дата звернення: 10.05.2025).
4. Положення про навчальний план для здобувачів вищої освіти у Рівненському державному гуманітарному університеті. URL: https://www.rshu.edu.ua/images/navch/pol_nnavch_plan_04032020.pdf (дата звернення: 02.05.2025).
5. Про організацію освітнього процесу у Рівненському державному гуманітарному університеті. URL: https://www.rshu.edu.ua/images/rshu/pol_oop_2025.pdf (дата звернення: 03.05.2025).
6. «Про електронний цифровий підпис» : Закон України від 07.11.2018 № 852–15. URL: https://zakon.rada.gov.ua/laws/show/852-15 (дата звернення: 04.05.2025).
7. «Про електронні довірчі послуги» : Закон України від 01.01.2023 № 2155–19. URL: https://zakon.rada.gov.ua/laws/show/2155-19 (дата звернення: 18.05.2025).
8. «Про електронні документи та електронний документообіг» : Закон України від 01.08.2022 № 851–15. URL: http://zakon5.rada.gov.ua/laws/show/851-15 (дата звернення: 16.05.2025).
9. «Про захист інформації в інформаційно-комунікаційних системах» : Закон України від 01.07.2022 № 80/94-вр. URL: https://zakon.rada.gov.ua/laws/show/80/94-вр (дата звернення: 18.05.2025).

10. Реєстр Python-пакунків (PyPI). URL: https://pypi.org/ (дата звернення: 20.05.2025).
11. Що таке СУБД і для чого вони потрібні. URL: https://foxminded.ua/systema-upravlinnia-bazamy-danykh/ (дата звернення: 20.05.2025).
12. FastAPI. URL: https://fastapi.tiangolo.com/uk/ (дата звернення: 20.05.2025).
13. IDLE – Python editor and shell. URL: https://docs.python.org/uk/3.13/library/idle.html (дата звернення: 20.05.2025).
14. JetBrains. PyCharm features – jetbrains python IDE. JetBrains. URL: https://www.jetbrains.com/pycharm/features/ (дата звернення: 21.05.2025).
15. OCR та IDP для роботи з документами. URL: https://inbase.com.ua/post_product/megapolis-docnet-ua/ (дата звернення: 18.05.2025).
16. openpyxl 3.1.3 documentation. URL: https://openpyxl.readthedocs.io/en/stable/ (дата звернення: 20.05.2025).
17. os – Miscellaneous operating system interfaces. URL: https://docs.python.org/uk/3.13/library/os.html (дата звернення: 20.05.2025).
18. PostgreSQL: about. PostgreSQL: The world's most advanced open source database. URL: https://www.postgresql.org/about/ (дата звернення: 23.05.2025).
19. Swagger: API Documentation & Design Tools for Teams. URL: https://swagger.io/ (дата звернення: 18.05.2025).
20. Tutorial: Get started with Visual Studio Code. URL: https://code.visualstudio.com/docs/getstarted/getting-started (дата звернення: 20.05.2025).

ДОДАТКИ

ДОДАТОК А

Лістинг програмного коду

Мікро-сервіс 1

```
from fastapi import FastAPI, File, UploadFile, HTTPException, Depends, Query
from sqlalchemy import create_engine, Column, Integer, String, Date
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, Session
import openpyxl
from datetime import datetime
import os


app = FastAPI()
# Параметри для підключення до бази даних
DATABASE_URL = "postgresql://postgres:password@localhost:5432/postgres"
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()


# Модель бази даних з назвами атрибутів
class Student(Base):
 __tablename__ = "students"
 id = Column(Integer, primary_key=True, index=True)
 id_картки = Column(String, index=True) # ID картки
 здобувач = Column(String) # Здобувач
 дата_народження = Column(Date) # Дата народження
 тип_дпо = Column(String) # Тип ДПО
 серія_документа = Column(String) # Серія документа
 номер_документа = Column(String) # Номер документа
 стать = Column(String) # Стать
 громадянство = Column(String) # Громадянство
 піб_англійською = Column(String) # ПІБ англійською
 рнокпп = Column(String) # РНОКПП
 початок_навчання = Column(Date) # Початок навчання
 завершення_навчання = Column(Date) # Завершення навчання
 структурний_підрозділ = Column(String) # Структурний підрозділ
 освітній_ступінь = Column(String) # Освітній ступінь (рівень)
 вступ на основі = Column(String) # Вступ на основі
 форма_навчання = Column(String) # Форма навчання
 спеціальність = Column(String) # Спеціальність
 спеціалізація = Column(String) # Спеціалізація
 id_оп = Column(String) # ID ОП
 освітня_програма = Column(String) # Освітня програма
 курс = Column(String) # Курс
 наказ_про_зарахування = Column(String) # Наказ про зарахування


# Створення таблиці в базі даних
Base.metadata.create_all(bind=engine)

# Функція для обробки файлу Excel
def parse_excel_to_json(file_path, required_columns=None):
 wb = openpyxl.load_workbook(file_path)
 all_data = []
 for sheet_name in wb.sheetnames:
 sheet = wb[sheet_name]
 headers = [cell.value for cell in sheet[1]]
 for row in sheet.iter_rows(min_row=2, values_only=True):
```

```
  entry = {}
  for col_index, value in enumerate(row):
  if col_index < len(headers) and (required_columns is None or
headers[col_index] in required_columns):
   if isinstance(value, datetime):
   value = value.strftime("%Y-%m-%d")
   entry[headers[col_index]] = value
  if entry:
  all_data.append(entry)
 return all_data


# Ендпоінт для завантаження і обробки файлу
@app.post("/upload_excel/")
async def upload_excel(file: UploadFile = File(...)):
 if file.filename.endswith('.xlsx'):
 file_location = f"temp_{file.filename}"
 with open(file_location, "wb") as buffer:
  buffer.write(await file.read())

 required_columns = {
  "ID картки", "Здобувач", "Дата народження", "Тип ДПО", "Серія документа",
  "Номер документа", "Стать", "Громадянство", "ПІБ англійською", "РНОКПП",
  "Початок навчання", "Завершення навчання", "Структурний підрозділ",
  "Освітній ступінь (рівень)", "Вступ на основі", "Форма навчання",
  "Спеціальність", "Спеціалізація", "ID ОП", "Освітня програма","Курс", "Наказ
про зарахування"
 }

 # Парсимо файл і конвертуємо в JSON
 try:
  data = parse_excel_to_json(file_location, required_columns)
 finally:
  os.remove(file_location)

 # Зберігаємо дані в базі даних
 db = SessionLocal()
 try:
  for entry in data:
  student = Student(
   id_картки=entry.get("ID картки"),
   здобувач=entry.get("Здобувач"),
   дата_народження=datetime.strptime(entry.get("Дата народження"), "%Y-%m-
%d").date() if entry.get(
   "Дата народження") else None,
   тип_дпо=entry.get("Тип ДПО"),
   серія_документа=entry.get("Серія документа"),
   номер_документа=entry.get("Номер документа"),
   стать=entry.get("Стать"),
   громадянство=entry.get("Громадянство"),
   піб_англійською=entry.get("ПІБ англійською"),
   рнокпп=entry.get("РНОКПП"),
   початок_навчання=datetime.strptime(entry.get("Початок навчання"), "%Y-%m-
%d").date() if entry.get(
   "Початок навчання") else None,
   завершення_навчання=datetime.strptime(entry.get("Завершення навчання"),
        "%Y-%m-%d").date() if entry.get(
   "Завершення навчання") else None,
   структурний_підрозділ=entry.get("Структурний підрозділ"),
   освітній_ступінь=entry.get("Освітній ступінь (рівень)"),
   вступ_на_основі=entry.get("Вступ на основі"),
   форма_навчання=entry.get("Форма навчання"),
   спеціальність=entry.get("Спеціальність"),
   спеціалізація=entry.get("Спеціалізація"),
   id_оп=entry.get("ID ОП"),
   освітня_програма=entry.get("Освітня програма"),
```

```
    курс=entry.get("Курс"),
    наказ_про_зарахування=entry.get("Наказ про зарахування")
   )
   db.add(student)
   db.commit()
  except Exception as e:
   db.rollback()
   raise HTTPException(status_code=500, detail=f"Error saving to database: {e}")
  finally:
   db.close()

  return {"message": "Excel file successfully processed and data saved to
database"}
  else:
  raise HTTPException(status_code=400, detail="Invalid file format. Please upload
an.xlsx file.")

# Залежність для отримання сесії бази даних
def get_db():
 db = SessionLocal()
 try:
 yield db
 finally:
 db.close()

# Ендпоінт для отримання всіх студентів з бази даних
@app.get("/students/")
def get_students(db: Session = Depends(get_db)):
 students = db.query(Student).all()
 return students


# Ендпоінт для фільтрованого отримання студентів за будь-яким атрибутом
@app.get("/students/filter/")
def get_students_by_attributes(
 id_картки: str = Query(None),
 здобувач: str = Query(None),
 дата_народження: datetime = Query(None),
 тип_дпо: str = Query(None),
 серія_документа: str = Query(None),
 номер_документа: str = Query(None),
 стать: str = Query(None),
 громадянство: str = Query(None),
 піб_англійською: str = Query(None),
 рнокпп: str = Query(None),
 початок_навчання: datetime = Query(None),
 завершення_навчання: datetime = Query(None),
 структурний_підрозділ: str = Query(None),
 освітній_ступінь: str = Query(None),
 вступ_на_основі: str = Query(None),
 форма_навчання: str = Query(None),
 спеціальність: str = Query(None),
 спеціалізація: str = Query(None),
 id_оп: str = Query(None),
 освітня_програма: str = Query(None),
 курс: str = Query(None),
 наказ_про_зарахування: str = Query(None),
 db: Session = Depends(get_db)
):
 query = db.query(Student)

 # Додавання фільтрів для кожного атрибуту, якщо він заданий
 if id_картки:
 query = query.filter(Student.id_картки == id_картки)
 if здобувач:
 query = query.filter(Student.здобувач.contains(здобувач)) # пошук за частиною
```

```
імені
 if дата_народження:
 query = query.filter(Student.дата_народження == дата_народження)
 if тип_дпо:
 query = query.filter(Student.тип_дпо == тип_дпо)
 if серія_документа:
 query = query.filter(Student.серія_документа == серія_документа)
 if номер_документа:
 query = query.filter(Student.номер_документа == номер_документа)
 if стать:
 query = query.filter(Student.стать == стать)
 if громадянство:
 query = query.filter(Student.громадянство == громадянство)
 if піб_англійською:
 query = query.filter(Student.піб_англійською.contains(піб_англійською)) #
частковий пошук
 if рнокпп:
 query = query.filter(Student.рнокпп == рнокпп)
 if початок_навчання:
 query = query.filter(Student.початок_навчання == початок_навчання)
 if завершення_навчання:
 query = query.filter(Student.завершення_навчання == завершення_навчання)
 if структурний_підрозділ:
 query = query.filter(Student.структурний_підрозділ == структурний_підрозділ)
 if освітній_ступінь:
 query = query.filter(Student.освітній_ступінь == освітній_ступінь)
 if вступ_на_основі:
 query = query.filter(Student.вступ_на_основі == вступ_на_основі)
 if форма_навчання:
 query = query.filter(Student.форма_навчання == форма_навчання)
 if спеціальність:
 query = query.filter(Student.спеціальність == спеціальність)
 if спеціалізація:
 query = query.filter(Student.спеціалізація == спеціалізація)
 if id_оп:
 query = query.filter(Student.id_оп == id_оп)
 if освітня_програма:
 query = query.filter(Student.освітня_програма == освітня_програма)
 if освітня_програма:
 query = query.filter(Student.курс == курс)
 if наказ_про_зарахування:
 query = query.filter(Student.наказ_про_зарахування == наказ_про_зарахування)

 students = query.all()
 return students

# Ендпоінт для отримання списку всіх унікальних груп
@app.get("/groups")
def get_all_groups(db: Session = Depends(get_db)):
 groups = db.query(Student.спеціальність).distinct().all()
 return [group[0] for group in groups if group[0]]

# Ендпоінт для отримання списку студентів у певній групі (тільки ім'я та ID)
@app.get("/students/group/{group_name}")
def get_students_in_group(group_name: str, db: Session = Depends(get_db)):
 students = db.query(Student.id_картки,
Student.здобувач).filter(Student.спеціальність == group_name).all()
 return {"students": [{"id": student[0], "ім'я": student[1]} for student in
students]}

# Ендпоінт для отримання повної інформації про студента за його ID
@app.get("/student/{id_card}")
def get_student_by_id(id_card: str, db: Session = Depends(get_db)):
 student = db.query(Student).filter(Student.id_картки == id_card).first()
 if student is None:
 raise HTTPException(status_code=404, detail="Student not found")
```

```
 return student

# Ендпоінт для отримання списку студентів за курсом
@app.get("/students/course/{course}")
def get_students_by_course(course: str, db: Session = Depends(get_db)):
 students = db.query(Student).filter(Student.курс == course).all()
 if not students:
 raise HTTPException(status_code=404, detail="No students found for this
course")
 return students
```

Мікро-сервіс 2

```
from fastapi import FastAPI, UploadFile, File, HTTPException
from fastapi.responses import JSONResponse
import openpyxl
from datetime import datetime
import json
import os
import re
from typing import List

app = FastAPI()

# Допоміжна функція для отримання семестрів певного курсу
def get_semesters_for_course(course: int) -> set[int]:
    return {2 * course - 1, 2 * course}

# Допоміжна функція для видобування року з імені файлу
def extract_year_from_filename(filename: str) -> int:
    match = re.search(r'(\d{4})', filename)
    if not match:
        raise ValueError("Файл не містить рік у назві")
    return int(match.group(1))

# Змінні для зберігання оброблених даних
stored_plan_data = []
subjects_data = []
summary_data = {}

@app.post("/upload-plan/")
async def upload_excel(file: UploadFile = File(...)):
    logs: List[str] = []
    try:
        filename = file.filename
        file_year = extract_year_from_filename(filename)
        current_year = datetime.now().year
        course = current_year - file_year
        logs.append(f"[INFO] Витягнуто рік: {file_year}, Поточний рік:
{current_year}, Курс: {course}")
    except Exception as e:
        raise HTTPException(status_code=400, detail=f"Помилка обчислення курсу:
{e}")

    semesters_for_course = get_semesters_for_course(course)
    contents = await file.read()
    temp_filename = f"temp_{filename}"
    with open(temp_filename, "wb") as f:
        f.write(contents)

    wb = openpyxl.load_workbook(temp_filename)
```

```
    sheet = wb.active

    students = 17 # кількість студентів у групі
    if students < 1:
        raise HTTPException(status_code=400, detail="Кількість студентів повинна
бути більше 0")
    num_students = 16 # кількість студентів для розділення
    all_data = [] # для зберігання результатів
    all_sum_data = [] # для зберігання загальних сум
    all_sum = 0 # загальна сума
    koof_work = students / 10 if students < 10 else 1 # коефіцієнт навантаження
    work_num = 5 # мінімальне навантаження
    koof_lab = int(((students - 1) / num_students) + 1) # коефіцієнт
лабораторних
    koof_kursova = 3 # коофіцієнт для курсової
    koof_practuka = 1.3 # коофіцієнт для практики
    num_weeks = 2 # 2 тижні
    koof_vyrobnycha_practuka = 3 # 2 * num_weeks * кількість студентів + 0.2 *
кількість студентів
    koof_kvalificatsiyna = 13 # коофіцієнт для кваліфікаційної

    logs.append(f"[INFO] Коефіцієнт навантаження: {koof_work}")
    logs.append(f"[INFO] Кількість студентів у групі: {students}, Кількість
студентів для розділення: {num_students}")
    logs.append(f"[INFO] Коефіцієнт лабораторних: {koof_lab}")

    global stored_plan_data, subjects_data, summary_data
    stored_plan_data.clear()
    subjects_data.clear()
    summary_data.clear()

    for i, row in enumerate(sheet.iter_rows(min_row=11, values_only=True),
start=11):
        if row[1] and str(row[1]).strip().upper() == "ВК01/ ВК02/ ВК03":
            logs.append(f"[INFO] Рядок {i}: знайдено 'ВК01/ ВК02/ ВК03' —
зупинка.")
            break

        semesters = {s for s in (row[3], row[4], row[5]) if isinstance(s, int)}
        if not semesters_for_course.intersection(semesters):
            logs.append(f"[DEBUG] Рядок {i}: семестр {semesters} — не підходять
для курсу {course}")
            continue

        if row[14] != 1:
            logs.append(f"[DEBUG] Рядок {i}: O = {row[14]} — пропущено")
            if row[14] == 2:
                all_sum += students * koof_kursova
            elif row[14] == 3:
                all_sum += students * koof_practuka
            elif row[14] == 4:
                all_sum += students * koof_vyrobnycha_practuka
            elif row[14] == 5:
                all_sum += students * koof_kvalificatsiyna
            else:
                continue

        temp_sum = koof_work * (row[10] + row[11] + (koof_lab * row[12]))
        if temp_sum < work_num:
            temp_sum = work_num

        all_sum += row[10] + row[11] + (koof_lab * row[12])

        entry = {
            "subject": row[1],
            "column_K": row[10],
```

```
                "column_L": row[11],
                "column_M": row[12],
                "SUM_K_L_M": temp_sum,
                "course": course,
                "semesters": list(semesters)
            }
            logs.append(str(entry))
            all_data.append(entry)

            subject_entry = {
                "name": row[1],
                "course": course,
                "semesters": list(semesters)
            }
            subjects_data.append(subject_entry)

    all_sum_data.append({f"ALL_SUM_{course}": all_sum})
    summary_data = all_sum_data[0]
    stored_plan_data = all_data

    json_result_path = f"{os.path.splitext(filename)[0]}_result.json"
    with open(json_result_path, "w", encoding="utf-8") as f:
        json.dump(all_data, f, ensure_ascii=False, indent=4)

    log_path = f"{os.path.splitext(filename)[0]}_logs.json"
    with open(log_path, "w", encoding="utf-8") as log_file:
        json.dump(logs, log_file, ensure_ascii=False, indent=4)

    all_sum_path = f"{os.path.splitext(filename)[0]}_all_sums.json"
    with open(all_sum_path, "w", encoding="utf-8") as all_sum_file:
        json.dump(all_sum_data, all_sum_file, ensure_ascii=False, indent=4)

    os.remove(temp_filename)

    return JSONResponse(content={
        "message": f"Оброблено {len(all_data)} рядків",
        "result_file": json_result_path,
        "log_file": log_path,
        "logs": logs
    })

# Отримати повну структуру обробленого навантаження
@app.get("/plan/")
def get_plan():
    return JSONResponse(content=stored_plan_data)

# Отримати перелік дисциплін із курсом та семестром
@app.get("/subjects/")
def get_subjects():
    return JSONResponse(content=subjects_data)

# Отримати загальне підсумкове навантаження
@app.get("/hours-summary/")
def get_hours_summary():
    return JSONResponse(content=summary_data)
```