

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**  
**ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ**  
Факультет комп'ютерних наук, фізики та математики  
Кафедра комп'ютерних наук та програмної інженерії

**Проектування та розробка модуля «Електронний розклад» для  
інформаційної системи KSU24**

Кваліфікаційна робота (проект)  
на здобуття ступеня вищої освіти “бакалавр”

Виконав: здобувач 4 курсу 12-441 групи

Спеціальності 121 Інженерія програмного забезпечення

Освітньо-професійної програми

Інженерія програмного забезпечення

першого (бакалаврського) рівня вищої освіти

Метін Богдан тунджайович

Керівник кандидат педагогічних наук, доцент  
Вінник Максим Олександрович

Рецензент доктор педагогічних наук, професор,  
професор кафедри інформатики і кібернетики  
Мелітопольського державного педагогічного  
університету імені Богдана Хмельницького  
Круглик Владислав Сергійович

## ЗМІСТ

|                                                                                                                         |    |
|-------------------------------------------------------------------------------------------------------------------------|----|
| ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ .....                                                                                     | 3  |
| ВСТУП.....                                                                                                              | 4  |
| РОЗДІЛ 1 ПІДХОДИ ДО ФОРМУВАННЯ ТА ОРГАНІЗАЦІЇ РОЗКЛАДУ<br>НАВЧАЛЬНОГО ПРОЦЕСУ .....                                     | 6  |
| 1.1 Особливості організації навчального процесу та формування розкладу<br>занять у закладах вищої освіти .....          | 6  |
| 1.2 Основні підходи до формування розкладу занять у закладах вищої освіти.                                              | 8  |
| 1.3 Порівняльна характеристика ручного, напівавтоматизованого та<br>автоматизованого підходів формування розкладу ..... | 11 |
| 1.4 Критерії якості та ефективності розкладу навчального процесу .....                                                  | 16 |
| РОЗДІЛ 2 ТЕХНОЛОГІЇ ПРОЄКТУВАННЯ ТА РОЗРОБКИ МОДУЛЯ<br>ЦИФРОВОГО РОЗКЛАДУ НАВЧАЛЬНОГО ПРОЦЕСУ У СИСТЕМІ KSU24           | 19 |
| 2.1 Технології розробки серверних модулів у системі KSU24 .....                                                         | 19 |
| 2.2 Мета розробки та обґрунтування необхідності створення модулю<br>цифрового розкладу .....                            | 29 |
| 2.3 Архітектура та загальна структура модуля .....                                                                      | 32 |
| 2.4 Проєктування та реалізація моделей даних цифрового розкладу .....                                                   | 33 |
| 2.5 Реалізація механізму чорнового збереження розкладу .....                                                            | 38 |
| 2.6 Проєктування та реалізація REST API модуля цифрового розкладу .....                                                 | 41 |
| ВИСНОВКИ .....                                                                                                          | 48 |
| СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....                                                                                        | 50 |

## ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ

1. **API** — набір правил та інтерфейсів для взаємодії між програмними компонентами (Application Programming Interface).
2. **HTTP** — протокол прикладного рівня для передачі гіпертекстових документів та даних у мережі (HyperText Transfer Protocol).
3. **JARS** — система архівації та реєстрації даних у форматі JSON всистемі KSU24 (JSON Archive and Registry System).
4. **JSON** — текстовий формат серіалізації даних, що використовує синтаксис об'єктів JavaScript для обміну інформацією між системами (JavaScript Object Notation).
5. **LMS** — програмна платформа для організації, управління та доставки навчального контенту (Learning Management System).
6. **OpenAPI** — стандартизована специфікація для опису, документування та візуалізації RESTful API у форматі, незалежному від мови програмування.
7. **ORM** — технологія, що забезпечує відображення об'єктів мови програмування на таблиці реляційної бази даних (Object-Relational Mapping).
8. **REST** — архітектурний стиль побудови мережових додатків, що базується на принципах передачі стану представлення ресурсу (Representational State Transfer).
9. **RESTful API** — програмний інтерфейс, побудований відповідно до архітектурних обмежень REST для організації доступу до ресурсів через мережу.
10. **SQL** — декларативна мова для керування та маніпулювання даними в реляційних базах даних (Structured Query Language).

## ВСТУП

В умовах активної цифровізації вищої освіти та зростаючих вимог до ефективності організації навчального процесу постає нагальна потреба у створенні програмних рішень [24], що забезпечують автоматизацію адміністративних процедур університету. Одним із ключових елементів такої цифрової трансформації є система управління розкладом занять, яка дозволяє якісно координувати взаємодію студентів, викладачів та адміністративного персоналу, а також забезпечує своєчасне виявлення організаційних конфліктів, пов'язаних із перетином занять, дефіцитом аудиторного фонду або нераціональним розподілом навчального навантаження між дисциплінами.

Актуальність дослідження полягає в необхідності створення технологічного рішення для централізованого зберігання, оперативного оновлення та стандартизованої обробки даних розкладу, яке б відповідало сучасним вимогам до інформаційних систем закладів вищої освіти. Така система суттєво знижує вплив людського фактору на достовірність навчальних даних, забезпечує їх цілісність і валідність, а також створює надійну інформаційну основу для прийняття обґрунтованих управлінських рішень у сфері організації навчального процесу.

Об'єкт дослідження — організація та цифровізація процесу формування розкладу занять у закладах вищої освіти.

Предмет дослідження — принципи побудови архітектури, моделювання даних та розробки REST API для системи управління розкладом університету.

Метою роботи є проєктування та програмна реалізація серверної частини системи управління розкладом занять університету із застосуванням фреймворку Django та побудовою програмного інтерфейсу відповідно до принципів REST. Реалізація цієї мети спрямована на забезпечення надійного функціонування навчального процесу, підвищення якості організаційних рішень та створення технологічної основи для подальшого розвитку цифрової інфраструктури університету.

Відповідно до поставленої мети в роботі визначено такі завдання:

- a) Проаналізувати особливості організації розкладу навчального процесу у закладах вищої освіти та визначити основні вимоги до його цифровізації;
- b) Дослідити актуальні підходи до проєктування серверної частини веб-систем та механізми взаємодії між серверною і клієнтською частинами системи на основі архітектурного стилю REST;
- c) Спроєктувати моделі даних засобами фреймворку Django відповідно до предметної області системи управління розкладом занять, визначивши структуру сутностей, типи полів, обмеження цілісності та зв'язки між компонентами інформаційної моделі;
- d) Реалізувати механізми валідації вхідних даних на рівні серверної частини, а також уніфіковану обробку помилок, що забезпечить стабільність і передбачуваність поведінки системи в нештатних ситуаціях;
- e) Спроєктувати та реалізувати REST API серверної частини, визначивши повний перелік ендпоінтів для управління даними розкладу навчальних занять, розробити DTO-схеми запитів і відповідей, реалізувати логіку обробки CRUD-операцій, фільтрацію, пошук і пагінацію;
- f) Забезпечити коректну взаємодію серверної частини з клієнтським застосунком шляхом стандартизації форматів обміну даними та визначення уніфікованої логіки формування API-відповідей;
- g) Реалізувати систему розмежування прав доступу відповідно до ролей користувачів, що гарантуватиме захист даних та відповідність системи вимогам інформаційної безпеки;
- h) Провести тестування функціоналу серверної частини з метою перевірки коректності реалізованих механізмів та відповідності системи визначеним функціональним вимогам.
- i) Розробити технічну документацію до серверної частини системи, що міститиме опис архітектурних рішень, структури бази даних та специфікацію реалізованого REST API.

## РОЗДІЛ 1

# ПІДХОДИ ДО ФОРМУВАННЯ ТА ОРГАНІЗАЦІЇ РОЗКЛАДУ НАВЧАЛЬНОГО ПРОЦЕСУ

### 1.1 Особливості організації навчального процесу та формування розкладу занять у закладах вищої освіти

Організація навчального процесу у закладах вищої освіти України базується на нормативно-правових актах держави, які визначають загальні вимоги до структури та змісту освітніх програм, академічних груп, та навчального навантаження. Основним документом, що регламентує систему вищої освіти в Україні, є Закон України “Про вищу освіту” [30].

Основою організації навчального процесу є освітні програми, вони визначають перелік навчальних дисциплін, їх обсяг у кредитах, форму контролю та послідовність вивчення [21]. Навчальні плани включають: перелік обов’язкових та вибіркових дисциплін; розподіл кількості годин (лекційних, практичних, лабораторних); форми підсумкового контролю, та розподіл навантаження за семестрами. Саме навчальний план виступає нормативною основою для формування розкладу занять, оскільки він визначає структуру освітнього процесу, обсяг навантаження, послідовність вивчення дисциплін, форми контролю та розподіл кредитів за семестрами, що забезпечує відповідність розкладу встановленим освітнім стандартам.

Навчальний процес організовується для академічних груп студентів, сформованих відповідно до спеціальності та року навчання. Його організація має низку особливостей: окремі навчальні дисципліни можуть викладатися водночас для декількох академічних груп, що потребує узгодження їхнього розкладу, раціонального використання аудиторного фонду та врахування навантаження науково-педагогічних працівників та студентів; у межах однієї групи можливий поділ на підгрупи для проведення лабораторних або практичних занять; крім цього, різні групи та студенти можуть мати відмінний перелік вибіркових дисциплін. Особливості організації навчального процесу потребують комплексного

підходу при формуванні розкладу занять із урахуванням усіх організаційних обмежень.

Кожному викладачу визначається навчальне навантаження відповідно до його ставки та кваліфікації. При формуванні розкладу занять необхідно врахувати загальну кількість годин викладача, закріплення дисциплін за конкретним науково-педагогічним працівником, а також індивідуальні побажання викладачів щодо розподілу занять протягом тижня. Такий підхід дозволяє забезпечити збалансоване навантаження, узгоджений графік та ефективну організацію навчального процесу.

Заклади вищої освіти мають у своєму розпорядженні обмежену кількість аудиторного фонду, до складу якого входять лекційні аудиторії, спеціалізовані лабораторії, спортивні зали та інші навчальні приміщення. Організація навчального процесу потребує врахування ключових характеристик аудиторій, зокрема їхньої місткості, технічного оснащення та доступності у визначені часові інтервали. Обмеження наявного фонду аудиторій зумовлює додаткові організаційні вимоги до планування розкладу, та передбачає збалансоване використання ресурсів університету.

Організація навчального процесу здійснюється з урахуванням академічного календаря, тривалості навчального семестру, встановленої кількості пар на день та регламентованої тривалості кожного заняття. Крім того, можуть застосовуватися додаткові часові обмеження, зокрема обмеження на кількість занять підряд, заборона проведення занять у певні дні та необхідність дотримання перерв між парами. Урахування цих аспектів є важливим для забезпечення ефективності навчального процесу та збалансованого навантаження студентів та науково-педагогічних працівників.

Освітній процес в закладах вищої освіти характеризується складною структурою та наявністю значної кількості взаємопов'язаних елементів. Формування розкладу занять у таких умовах передбачає одночасне врахування навчального навантаження науково-педагогічних працівників, ефективного використання аудиторного фонду, особливостей організації академічних груп і по-

токів, а також дотримання вимог навчальних планів і чинних нормативних документів. Комплексність зазначених факторів зумовлює значну кількість детермінованих обмежень, що значною мірою ускладнює процес планування. З огляду на це, застосування виключно ручних методів формування розкладу супроводжується підвищеним ризиком виникнення логістичних конфліктів, пов'язаних із накладанням занять, неузгодженістю навантаження або нераціональним використанням ресурсів. Людський фактор у такому випадку виступає суттєвим обмеженням, оскільки одночасне опрацювання великого масиву змінних і взаємозв'язків є складним завданням навіть для досвідчених фахівців. У зв'язку з цим виникає об'єктивна потреба у впровадженні спеціалізованих систем управління розкладом, здатних автоматизувати ключові етапи його формування [35]. Це сприятиме підвищенню точності, узгодженості та прозорості процесу формування розкладу. У результаті впровадження цифрових інструментів система управління розкладом набуває ознак цілісності, верифікованості та актуальності даних. Зазначений підхід формує надійну та достовірну інформаційну основу, що забезпечує консистентність і стабільність функціонування освітнього середовища, а також сприяє підвищенню ефективності управління та організації навчального процесу в закладах вищої освіти.

## **1.2 Основні підходи до формування розкладу занять у закладах вищої освіти**

Формування розкладу занять у закладах вищої освіти є комплексною організаційною задачею, що потребує одночасного врахування значної кількості взаємопов'язаних факторів [2]. До них належать: структура освітніх програм і навчальних планів, кількість та склад академічних груп, навчальне навантаження викладачів, наявність і характеристики аудиторного фонду, а також часові межі навчального процесу, визначені академічним календарем. Окрім того, необхідно дотримуватися нормативних вимог щодо обсягу навчального навантаження здобувачів освіти, розподілу дисциплін у межах семестру, тривалості занять та регламентованих перерв між ними.

Специфіка зазначеної задачі полягає в тісному взаємозв'язку всіх її складових: коригування будь-якого параметра — наприклад, заміна викладача або перепризначення аудиторії — може спричинити необхідність каскадного перегляду значної частини вже сформованого розкладу. Зі збільшенням кількості академічних груп і навчальних дисциплін складність задачі зростає пропорційно, оскільки кількість можливих комбінацій розміщення занять суттєво розширюється. Це зумовлює високий рівень відповідальності в процесі формування розкладу та потребу в чітко визначених методологічних підходах до його побудови.

У сучасній практиці організації освітнього процесу можна виділити три основні підходи до формування розкладу занять, що відрізняються рівнем використання інформаційних технологій та ступенем автоматизації: ручний, напівавтоматизований та автоматизований [1]. Кожен із них характеризується власними особливостями, перевагами та обмеженнями, що визначають доцільність застосування залежно від масштабів закладу освіти та рівня цифровізації його внутрішніх процесів.

Ручне формування розкладу передбачає організацію навчального процесу без застосування спеціалізованих інформаційних систем. На практиці цей підхід реалізується засобами загального призначення — текстовими редакторами або електронними таблицями. Відповідальна особа самотійно вносить дані щодо навчальних планів, академічних груп, навантаження викладачів та часових обмежень, після чого послідовно здійснює розподіл занять у межах доступних часових слотів. Перевірка відсутності конфліктів, дотримання навантаження та узгодженості дисциплін виконується вручну або за допомогою елементарних формул. Особливої складності ручний підхід набуває в умовах масштабування: зі збільшенням кількості учасників освітнього процесу обсяг даних зростає, що ускладнює їх узгодження та контроль. У разі одночасної роботи кількох осіб над розкладом виникають ризики дублювання інформації, розбіжностей у даних та втрати актуальності версій. Таким чином, ефективність ручного підходу значною мірою визначається досвідом відповідальних працівників, а його

масштабування супроводжується суттєвим зростанням організаційного навантаження.

Напіваавтоматизоване формування розкладу поєднує елементи ручного планування з використанням спеціалізованих програмних рішень або внутрішніх інформаційних систем закладу. Вихідні дані щодо навчальних планів, академічних груп та навантаження викладачів вносяться до програмного середовища, яке забезпечує їх структурування та первинну перевірку. Система може автоматично контролювати відповідність встановленому навантаженню, виявляти перетини занять і сигналізувати про порушення нормативних вимог. Водночас прийняття ключових рішень щодо структури розкладу залишається в компетенції диспетчера. За умови участі кількох осіб у процесі формування розкладу такий підхід забезпечує централізоване зберігання інформації та спільний доступ до неї, проте питання координації та розмежування відповідальності потребують додаткової організаційної регламентації. Отже, напіваавтоматизований підхід характеризується частковою формалізацією процесу та раціональним поєднанням алгоритмічного контролю з експертною участю людини.

Автоматизоване формування розкладу ґрунтується на застосуванні спеціалізованих інформаційних систем, у яких процес побудови розкладу реалізується засобами алгоритмічної обробки даних [22, 33]. Усі параметри освітнього процесу попередньо формалізуються та подаються у вигляді структурованої інформації: навчальні плани, склад академічних груп, закріплення дисциплін за викладачами, часові обмеження та нормативні умови. На основі введених параметрів система здійснює автоматичний аналіз можливих варіантів розміщення занять із урахуванням усіх встановлених обмежень. У разі внесення змін — заміни викладача, коригування навчального плану або появи нової групи — система може автоматично перебудовувати розклад відповідно до оновлених даних. Разом із тим ефективне функціонування автоматизованого підходу можливе лише за умови повної та завчасної підготовки вхідних параметрів, які мають бути внесені до системи в структурованому й узгодженому вигляді до початку навчального процесу. Часті або непередбачувані зміни вихідних даних можуть

знижувати ефективність автоматичного формування та потребувати повторного циклу обчислень або додаткового адміністративного втручання. Таким чином, автоматизований підхід забезпечує високий рівень формалізації процесу, централізоване зберігання інформації та алгоритмічне формування розкладу на основі заздалегідь визначених і стабільних параметрів.

### **1.3 Порівняльна характеристика ручного, напівавтоматизованого та автоматизованого підходів формування розкладу**

З огляду на комплексність процесу створення розкладу занять та значну кількість взаємопов'язаних факторів які беруть участь під час його формування, доцільно здійснення системного порівняння трьох основних підходів [4]: ручного, напівавтоматизованого та автоматизованого. Це дозволить визначити їх функціональні можливості, обмеження, рівень ефективності та доцільність застосування залежно від масштабу освітньої установи й рівня цифровізації її управлінських процесів.

#### **Рівень автоматизації процесу**

- *Ручний підхід* характеризується мінімальним рівнем формалізації процесу формування розкладу. Всі управлінські рішення приймаються відповідальною особою (диспетчером) на основі її досвіду роботи. Програмні інструменти (текстові редактори, електронні таблиці) виконують виключно допоміжну функцію фіксації даних, не здійснюючи їх логічного аналізу.
- *Напівавтоматизований підхід* передбачає часткову формалізацію інформації та використання програмних засобів для автоматичної перевірки окремих критеріїв. Система здатна виявляти перетини занять, перевищення допустимого навантаження викладачів або місткості аудиторії. Водночас прийняття остаточних рішень щодо розподілу занять за аудиторіями та у часових рамках залишається за диспетчером.
- *Автоматизований підхід* базується на повній формалізації предметної області. Усі параметри освітнього процесу: академічні групи, дисцип-

ліни, навантаження викладачів, характеристики аудиторій, часові обмеження — подаються у вигляді структурованих даних. Формування розкладу здійснюється алгоритмічно, із застосуванням математичних моделей або оптимізаційних методів. Безпосередня участь фахівця полягає у налаштуванні правил і параметрів системи та здійсненні оцінки сформованого результату.

#### Масштабованість системи

- *Ручний підхід* є ефективним лише за невеликих обсягів даних. Зі збільшенням їх кількості складність задачі зростає експоненційно, що призводить до значного ускладнення процесу узгодження параметрів і підвищення ризику помилок. Крім того, координація роботи кількох відповідальних осіб у межах ручного підходу є складною задачею через відсутність централізованої системи контролю версій.
- *Напіваавтоматизований підхід* забезпечує вищу масштабованість завдяки централізованому зберігання інформації та автоматичному контролю частини обмежень. Проте при значному зростанні обсягу даних навантаження на відповідальну особу залишається суттєвим, оскільки саме вона приймає кінцеві рішення щодо структури розкладу.
- *Автоматизований підхід* демонструє найвищий рівень масштабованості. Алгоритмічні методи дозволяють опрацьовувати великі масиви взаємопов'язаних даних без значного збільшення трудових витрат.

#### Ризик виникнення помилок та вплив людського фактору

- У *ручному підході* ризик виникнення помилок є найбільшим. Людський фактор може призводити до пропуску конфліктів, помилкового розподілу навантаження або неточності у розміщенні занять. Перевірка відповідності обмеженням здійснюється вручну, що збільшує ймовірність виникнення помилок у кінцевому розкладі.
- *Напіваавтоматизований підхід* суттєво знижує ризик технічних недоліків, оскільки система автоматично перевіряє окремі параметри. Проте

логічні помилки, пов'язані з прийняттям управлінських рішень, залишаються можливими.

- *Автоматизований підхід* мінімізує вплив людського фактору на етапі формування розкладу. Система враховує всі задані обмеження одночасно, що практично виключає технічні конфлікти. Однак помилки можуть виникати у випадку некоректно заданих вихідних даних або неправильно сформульованих критеріїв перевірки.

Гнучкість та адаптивність до змін в освітньому процесі

- У межах *ручного підходу* будь-яке коригування потребує детального перегляду вже створеного розкладу. Зміна одного параметра (наприклад, аудиторії або викладача) часто спричиняє так званий “ланцюговий ефект”, коли необхідно узгоджувати нові часові інтервали з іншими заняттями, перевіряти відсутність конфліктів у навантаженні викладачів та доступності приміщень. У результаті навіть незначна зміна може призвести до істотних часових витрат та підвищення ризику помилок некоректного формування розкладу занять.
- *Напіваавтоматизований підхід* частково підвищує рівень адаптивності завдяки використанню програмних засобів контролю обмежень. Система здатна автоматично виявляти конфлікти, перетин занять у розкладі викладача, їх накладання у межах однієї академічної групи чи перевищення місткості аудиторії. Проте безпосередня перебудова розкладу зазвичай залишається за диспетчером, що обмежує швидкість реагування та зберігає залежність від суб'єктивних рішень. Це покращує процес контролю якості, але не забезпечує повної автоматизації процесу переформування.
- *Автоматизований підхід* передбачає використання алгоритмічних методів оптимізації, які дозволяють виконувати повторний перерахунок розкладу з урахуванням оновлених параметрів. У разі зміни вхідних даних система може автоматично сформувати новий варіант розкладу, що відповідає заданим обмеженням і критеріям. Це забезпечує висо-

кий рівень адаптивності та мінімізує вплив людського фактору. Водночас часті або масштабні зміни можуть вимагати повторної ітерації складних алгоритмів оптимізації, що супроводжується додатковими обчислювальними витратами та може впливати на швидкість формування кінцевого результату. Крім того, у випадку суттєвих змін у принципах побудови розкладу, або в наборі обмежень може виникнути необхідність модифікації або навіть повного перегляду застосовуваних алгоритмів, що значно ускладнює підтримку такої системи.

#### Організаційна доцільність застосування

- Ручний підхід є доцільним у випадках невеликого обсягу даних і відносно стабільних умов формування розкладу. За таких обставин його використання не потребує значних ресурсів і дозволяє оперативно вносити зміни. Водночас із зростанням кількості учасників процесу та частоти коригувань ефективність цього підходу знижується, а ризик помилок в сформованому розкладі суттєво зростає. Зазначений підхід практично не потребує цифровізації процесів і може реалізовуватися без використання спеціалізованих інформаційних систем або з їх мінімальним залученням.
- *Напівавтоматизований підхід* є раціональним для умов середньої складності, коли виникає потреба в упорядкуванні даних і частковій автоматизації процесів. Він дозволяє підвищити прозорість і керованість процесу формування розкладу, зменшити навантаження на відповідальних осіб, водночас зберігаючи можливість ручного втручання для врахування специфічних ситуацій. Реалізація такого підходу передбачає наявність базового рівня цифровізації, зокрема використання інформаційних систем для зберігання, обробки та обміну даними, що забезпечує узгодженість дій між учасниками процесу.
- *Автоматизований підхід* є найбільш доцільним за наявності великих обсягів даних, складних взаємозв'язків і високої динаміки змін. Його застосування забезпечує узгодженість розкладу, ефективне викорис-

тання ресурсів і швидке оновлення результатів у разі зміни параметрів. Ефективне використання цього підходу можливе лише за умов високого рівня цифровізації, що передбачає наявність комплексних інформаційних систем, інтеграцію даних між різними підсистемами та забезпечення їх актуальності.

Проведений аналіз дозволяє зробити висновок, що вибір підходу до формування розкладу занять безпосередньо залежить від масштабів закладу освіти, рівня цифровізації управлінських процесів, організаційної структури та стратегічних цілей розвитку установи.

Ручний підхід може бути прийнятним лише для невеликих освітніх структур або за умов відносно простої організації навчального процесу. Напівавтоматизований виступає проміжною моделлю, що поєднує традиційні механізми планування з інструментами цифрової підтримки та контролю обмежень. Автоматизований підхід, своєю чергою, є найбільш перспективним з огляду на сучасні тенденції цифрової трансформації освіти, оскільки забезпечує системність, масштабованість і мінімізацію впливу людського фактору. Водночас у межах проведеного аналізу для практичної реалізації було обрано саме напівавтоматизований підхід. Такий вибір зумовлений необхідністю забезпечення достатнього рівня гнучкості та можливості оперативного втручання в процес формування розкладу. Незважаючи на те, що автоматизований підхід в перспективі являє собою більш оптимальне, та амбітне рішення, такі системи, як правило, передбачають чітку формалізацію всіх параметрів та функціонування за умов стабільності й повноти вихідних даних. На практиці ж навчальний процес характеризується варіативністю, неповнотою інформації та частими змінами, що ускладнює досягнення “ідеальних” умов для коректної роботи алгоритмів оптимізації.

У результаті проведеного аналізу доцільно стверджувати, що у рамках Херсонського державного університету, напівавтоматизований підхід можна розглядати як раціональний компроміс між рівнем автоматизації та збереженням управлінської гнучкості [38], що забезпечує поєднання переваг цифрових

інструментів із можливістю ефективного управління змінами в організаційних процесах.

#### **1.4 Критерії якості та ефективності розкладу навчального процесу**

Якість розкладу занять є визначальним аспектом організації освітнього процесу, оскільки саме вона обумовлює ефективність навчання та раціональність використання ресурсів закладу вищої освіти. На відміну від формально коректного розкладу, який лише не містить технічних помилок, якісний розклад забезпечує комфортні умови для всіх учасників навчального процесу, сприяє ефективному засвоєнню навчального матеріалу та оптимальному використанню наявних часових і матеріальних ресурсів. Оцінювання якості розкладу має багаторівневий характер і потребує комплексного аналізу сукупності взаємопов'язаних критеріїв.

Базовим критерієм якості є відсутність конфліктів у часовому розміщенні занять. Під конфліктом слід розуміти ситуацію, за якої один і той самий ресурс — академічна група, викладач або аудиторія — залучається одночасно до кількох навчальних подій. Наявність таких конфліктів унеможливорює проведення занять відповідно до встановленого графіка, що робить розклад непридатним до практичного використання [3]. Забезпечення безконфліктності є обов'язковою умовою коректного розкладу та досягається шляхом узгодження всіх параметрів і перевірки їх сумісності. Разом із тим відсутність конфліктів є необхідною, але недостатньою умовою високої якості розкладу.

Не менш важливим є рівномірність розподілу навчального навантаження протягом тижня. Надмірна концентрація занять в окремі дні призводить до перевантаження студентів і викладачів, зниження рівня засвоєння матеріалу та загального зменшення продуктивності освітнього процесу. Нерівномірність навантаження, за якої одні дні є перевантаженими, а інші — недостатньо заповненими, свідчить про неефективність планування та потребує корекції на етапі формування розкладу [34].

Суттєвого значення набуває також оптимальна тривалість навчального дня. Надмірно тривалий навчальний день негативно впливає на фізичний та психологічний стан учасників освітнього процесу, тоді як не виправдано скорочений — призводить до нераціонального використання часу й ресурсів. Тому під час формування розкладу необхідно забезпечити баланс між інтенсивністю навчального навантаження та реальними можливостями його ефективного сприйняття. Особливої уваги потребує раціональна організація перерв між заняттями. Наявність тривалих часових інтервалів, так званих “вікон”, ускладнює планування робочого дня та знижує ефективність використання доступного часу, тоді як обґрунтовані перерви є необхідною умовою відновлення працездатності учасників процесу [39].

З позиції викладачів якісний розклад повинен забезпечувати можливість ефективного планування професійної діяльності в усіх її вимірах — навчальному, науковому, методичному та організаційному. Розклад із численними розривами або нерівномірним навантаженням суттєво ускладнює виконання цих функцій. Визначальною характеристикою є також стабільність розкладу, що дозволяє викладачам завчасно планувати свою діяльність і мінімізувати вплив непередбачуваних змін. З позиції студентів як основних учасників освітнього процесу зручний розклад сприяє підвищенню відвідуваності занять, покращенню організації самостійної роботи та загальному підвищенню якості навчання. Логічна послідовність дисциплін, чергування різних типів занять та уникнення надмірної інтенсивності навчального дня дозволяють створити більш сприятливі умови для засвоєння знань.

Значну роль у визначенні якості розкладу відіграє ефективність використання матеріально-технічних ресурсів закладу освіти. Раціональне використання аудиторного фонду передбачає відповідність місткості аудиторії кількості студентів, врахування специфіки занять та рівномірний розподіл навантаження на приміщення. Неефективне використання ресурсів може призводити як до їх простою, так і до перевантаження, що негативно позначається на якості освітнього процесу в цілому.

Окремим критерієм оцінювання якості розкладу є його стійкість до змін. В умовах реального функціонування освітнього процесу нерідко виникає необхідність внесення оперативних коректив, зумовлених тимчасовою відсутністю викладача, змінами у використанні аудиторій або коригуванням навчальних планів. Якісний розклад повинен допускати внесення таких змін із мінімальним впливом на інші елементи системи, що дозволяє зберігати загальну структуру і уникати суттєвих порушень навчального процесу.

В умовах цифровізації освіти важливим критерієм ефективності є інформаційна доступність розкладу. Застосування цифрових систем забезпечує оперативний доступ до актуальної інформації для всіх учасників освітнього процесу, що особливо важливо в умовах динамічних організаційних змін [15]. Сучасні цифрові платформи надають розширені функціональні можливості: персоналізовані розклади, автоматичні сповіщення про зміни, інтеграцію з календарними сервісами та доступ до супровідної інформації про аудиторії, викладачів і формати занять. Для працівників, відповідальних за формування розкладу, це означає автоматизацію рутинних операцій, скорочення кількості помилок і зручність редагування. Для викладачів і студентів — доступ до достовірної інформації без необхідності додаткових уточнень, що суттєво знижує організаційне навантаження.

Для об'єктивного оцінювання якості розкладу доцільно застосовувати формалізовані кількісні показники: кількість конфліктів між заняттями, середня кількість занять на день, тривалість перерв, рівень завантаженості аудиторного фонду та ступінь рівномірності розподілу навчального навантаження протягом тижня. Використання таких метрик дозволяє здійснювати порівняльний аналіз альтернативних варіантів розкладу на основі єдиних критеріїв, виявляти конкретні структурні недоліки — надмірну концентрацію занять, нерівномірне використання аудиторій або наявність тривалих часових розривів — та своєчасно усувати їх.

## **РОЗДІЛ 2**

# **ТЕХНОЛОГІЇ ПРОЄКТУВАННЯ ТА РОЗРОБКИ МОДУЛЯ ЦИФРОВОГО РОЗКЛАДУ НАВЧАЛЬНОГО ПРОЦЕСУ У СИ- СТЕМІ KSU24**

### **2.1 Технології розробки серверних модулів у системі KSU24**

Реалізація серверної частини модулів віртуального освітнього середовища KSU24 здійснюється із застосуванням мови програмування Python, яка на сучасному етапі розвитку інформаційних технологій широко використовується для створення веб-застосунків, інформаційних систем та аналітичних платформ [32]. Вибір цієї мови обумовлюється її здатністю забезпечити ефективну реалізацію складної бізнес-логіки, що є характерною для LMS систем, а також високим рівнем гнучкості до змін функціональних вимог [5].

Python відзначається простим і структурованим синтаксисом, що дозволяє створювати програмний код, який легко читати та підтримувати. Це має особливе значення у процесі розробки комплексних систем, де важливо забезпечити узгодженість між різними компонентами та можливість їх подальшої модифікації. У межах KSU24, це дозволяє підтримувати єдину логіку функціонування модулів і забезпечує ефективну командну розробку.

Значною перевагою програмного коду написаного мовою Python є його здатність до швидкої адаптації до змін. Освітні інформаційні системи, характеризуються доволі високою динамічністю, оскільки вони мають реагувати на зміни в законодавстві та освітніх процесах. Завдяки гнучкості даної мови програмування, ці зміни можуть бути реалізовані без суттєвих ускладнень у структурі коду та бізнес-логіки.

Мова Python підтримує об'єктно-орієнтовану парадигму програмування, що дозволяє ефективно моделювати предметну область системи [18]. У межах цифрового освітнього середовища KSU24, це забезпечує можливість представлення ключових сутностей, їх зв'язків та ієрархію в системі у вигляді об'єктів,

що мають чітко визначену структуру, властивості та методи. Цей підхід сприяє підвищенню структурованості системи та спрощує її подальший розвиток. Окрім цього, Python дозволяє використовувати функціональні підходи, що значно спрощує обробку даних.

У процесі розробки важливу роль відіграє екосистема Python. Наявність значної кількості бібліотек і фреймворків дозволяє значною мірою скоротити час витрачений на розробку окремих компонентів системи та підвищити якість програмного забезпечення. Це дозволяє використовувати перевірені і оптимізовані рішення для реалізації типових задач і зосередитися на розробці конкретної бізнес-логіки.

Суттєвою перевагою є також можливість інтеграції Python з різними технологіями та сервісами. Завдяки підтримці поширених протоколів обміну даними Python забезпечує ефективну інтеграцію бекенд частини з іншими підсистемами та зовнішніми сервісами. Крім того, Python підтримує сучасні підходи до побудови розподільних систем, включаючи використання асинхронних запитів та обробку великої кількості одночасних з'єднань. Це особливо важливо для застосунків, де необхідно забезпечити стабільну роботу при високому навантаженні та швидкий обмін даними між компонентами системи. У результаті вибір мови Python значно спрощує процес інтеграції, підвищує гнучкість архітектури та забезпечує чітку взаємодію між усіма складовими системи.

До ключових характеристик Python, які визначають доцільність його використання у системі KSU24, належать:

- висока читабельність програмного коду;
- гнучкість та адаптивність до змін;
- підтримка різних парадигм програмування;
- широка екосистема бібліотек;
- можливість інтеграції з іншими системами.

Поєднання зазначених факторів дозволяє забезпечити ефективну реалізацію серверної частини, яка відповідає сучасним вимогам.

Для реалізації серверної архітектури системи KSU24 використовується веб-фреймворк Django, який є одним із найбільш ефективних інструментів для розробки бекенд рішень для веб-застосунків на основі мови Python. Django дозволяє забезпечити високий рівень структурованості програмного коду, а також сприяє підвищенню ефективності розробки та надійності системи [8]. Django надає розробнику набір готових рішень для реалізації базових компонентів веб-застосунків. Django надає розробнику набір готових рішень для реалізації базових компонентів веб-застосунків, що дозволяє уникнути необхідності створення таких компонентів з нуля та значно полегшує процес розробки, надаючи можливість швидкої реалізації функціональних модулів, що відповідають за обробку даних, авторизацію користувачів та бізнес-логіку.

Архітектура фреймворку Django базується на використанні патерну MTV, який являє собою адаптацію класичної моделі MVC [23]. Зазначена архітектура передбачає чітку ієрархію її компонентів: модель (Model) визначає структуру даних і забезпечує взаємодію з базою даних, шаблон (Template) відповідає за відображення інформації користувачу, тоді як представлення (View) здійснює обробку запитів і реалізацію бізнес-логіки. Такий підхід дозволяє досягти високого рівня модульності системи, та чітке розмежування області відповідальності, що у свою чергу спрощує її підтримку, тестування та подальший розвиток та розширення. Однією з ключових переваг Django є його чітко визначена модульна структура. Проєкт складається з окремих додатків (apps), які представляють собою незалежні функціональні модулі. Це дозволяє організувати систему у вигляді сукупності відокремлених компонентів, процес розробки та тестування яких можуть відбуватися незалежно один від одного. Це є особливо ефективним при розробці великих інформаційних систем, зокрема освітніх платформ, де кожен модуль має власну бізнес-логіку.

Важливим компонентом Django є його ORM, яка забезпечує рівень абстракції над реляційною базою даних. ORM дозволяє працювати з базою даних через об'єктно-орієнтовану модель мови Python, що створює абстракцію, за якої операції з даними виконуються у вигляді маніпуляції з об'єктами, а не че-

рез написання безпосередніх SQL-запитів [13]. Це дає змогу розробнику працювати з базою даних у контексті об'єктно-орієнтованого програмування, що значно спрощує процес розробки та підвищує читабельність коду.

Система маршрутизації (URL dispatcher) у Django дозволяє гнучко налаштовувати обробку HTTP-запитів. Вона реалізується через декларативне визначення URL шаблонів, які зв'язуються з відповідними функціями. Це забезпечує зрозумілу та масштабовану структуру API, що є критичним фактором для побудови RESTful сервісів. Також Django містить потужну систему автентифікації та авторизації, яка входить до складу ядра фреймворку. Вона включає модель користувача, механізми керування паролями, систему груп та дозволів [7]. Завдяки цьому можна реалізувати гнучке управління доступом до ресурсів системи на основі ролей користувачів. Додатково можливе підключення сторонніх механізмів аутентифікації, таких як JWT або OAuth, що є актуальним для сучасних веб-додатків.

Адміністративний інтерфейс Django є однією із ключових переваг цього фреймворку. Він відіграє важливу роль у процесі розробки та супроводу веб-застосунків. Адміністративна панель генерується автоматично на основі опису моделей, що дозволяє розробнику отримати готовий інтерфейс для управління інформацією без необхідності реалізовувати його вручну. Це значно економить час та зусилля, особливо при роботі з великими обсягами даних, забезпечуючи швидкий доступ до основних операцій: створення, редагування, видалення та перегляду записів. Одним із ключових моментів адміністративного інтерфейсу є його гнучкість та можливість налаштування через класи `ModelAdmin` [6]. Це дозволяє адаптувати адмін панель під конкретні потреби проєкту: наприклад, додавати користувацькі фільтри, реалізовувати пошук за різними полями, налаштовувати сортування та змінювати відображення списків записів. Крім того, за допомогою inline-інтерфейсів можливо редагувати пов'язані моделі безпосередньо у формі, що значно спрощує роботу з взаємопов'язаними даними та підвищує ефективність адміністрування системи. Важливою особливістю адмін-панелі є також механізм контролю доступу, який дозволяє визначати права ко-

ристувачів та груп. Це гарантує безпеку даних та можливість обмежити доступ до певних дій відповідно до ролей, що є особливо актуальним для корпоративних систем. Додатково, ведення журналу змін дає змогу відстежувати історію редагувань, що підвищує прозорість та контроль за внесеними змінами. Завдяки широкому набору можливостей адміністративний інтерфейс Django поєднує в собі зручність, гнучкість та безпеку. Він дозволяє використовувати ефективні системи управління даними і одночасно забезпечує стандартизований підхід для роботи з інформацією, що має особливу актуальність у сучасних умовах. Адмін-панель не лише оптимізує рутинні дії з даними, але й підвищує рівень їх організованості у системі.

Суттєву роль у роботі Django відіграє система middleware. Middleware — це проміжний рівень обробки запитів і відповідей, який дозволяє реалізовувати загальну логіку. Наприклад, через middleware реалізуються обробка сесій, локалізація, кешування, безпека, логування запитів та інші важливі функції. Це забезпечує централізоване управління поведінкою системи. Не менш важливою є система міграцій, яка дозволяє керувати змінами структури бази даних. Django автоматично генерує файли міграцій на основі змін у моделях, що значно спрощує процес оновлення бази даних у різних середовищах. Це забезпечує узгодженість даних та спрощує командну розробку.

Безпека є однією з ключових характеристик Django. Фреймворк надає вбудовані механізми захисту від основних типів загроз які можуть виникнути на стороні сервера [37]:

- Захист від CSRF-атак;
- Захист від XSS через автоматичне екранування;
- Запобігання SQL-ін'єкціям завдяки ORM;
- Захист від clickjacking;
- Зберігання паролів із використанням сучасних алгоритмів хешування.

Важливою перевагою Django є наявність розвиненої екосистеми сторонніх бібліотек і пакетів, які дозволяють розширювати функціональність системи.

Це включає інструменти для роботи з API, обробки платежів, інтеграції з хмарними сервісами, аналітики та багатьох інших задач.

До основних переваг використання Django у межах розробки серверної частини доцільно віднести:

- Високу продуктивність;
- Підтримку складних запитів і ефективну роботу з базами даних;
- Надійність та стабільність функціонування;
- Наявність вбудованих механізмів безпеки;
- Підтримку масштабування системи;
- Гнучкість архітектури;
- Зручність розробки, тестування та супроводу програмного забезпечення.

Використання Django у поєднанні з мовою програмування Python дозволяє створити потужну платформу, яка відповідає сучасним вимогам до розробки інформаційних систем. Використання даного фреймворку забезпечує ефективну обробку запитів, безпечну роботу з даними та можливість масштабування системи KSU24 відповідно до зростання кількості користувачів і функціональних вимог.

Django Ninja є сучасним високорівневим фреймворком для розробки веб-API, який функціонує як надбудова над Django та орієнтований на створення продуктивних і типобезпечних серверних інтерфейсів [17]. Його архітектура поєднує можливості класичного Django із сучасними підходами до побудови RESTful-сервісів, що передбачають використання статичної типізації, автоматичної валідації даних та генерації документації. Однією з ключових характеристик Django Ninja є активне використання механізму анотацій типів (type hints), що дозволяє формалізувати структуру вхідних і вихідних даних на рівні сигнатур функцій, підвищуючи надійність програмного коду, спрощуючи його аналіз і суттєво знижуючи ймовірність виникнення помилок під час виконання; особливої важливості це набуває у серверній частині застосунків, де необхідно

забезпечити коректну обробку великої кількості запитів і гарантувати передбачуваність поведінки системи (рис. 2.1).

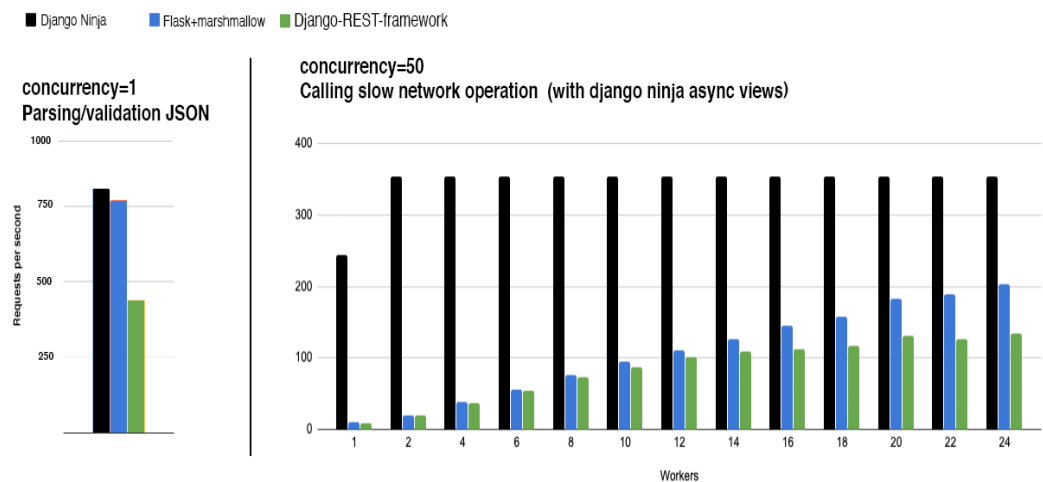


Рисунок 2.1. Продуктивність Django Ninja.

Для реалізації валідації та серіалізації даних використовується бібліотека Pydantic, яка забезпечує автоматичну перевірку відповідності даних визначеним схемам і дозволяє виконувати їх перетворення у необхідний формат без додаткових витрат на програмування [31], що сприяє підвищенню узгодженості даних у межах системи та зменшенню навантаження на бізнес-логіку. Організація маршрутизації запитів ґрунтується на декларативному підході з використанням декораторів, що дозволяє чітко встановити відповідність між HTTP-методами та функціями-обробниками, забезпечуючи зрозумілість структури API.

Важливою перевагою є автоматична генерація документації відповідно до стандарту OpenAPI [25], завдяки чому формується інтерактивне середовище для тестування та інтеграції API, що підвищує зручність системи для розробників і спрощує її подальший розвиток. У порівнянні з іншими інструментами екосистеми Django, зокрема Django REST Framework, даний фреймворк характеризується меншою кількістю надлишкових абстракцій і більш ефективною обробкою запитів, що дозволяє досягти вищої продуктивності, особливо в умовах підвищеного навантаження. Інтеграція з базовими компонентами Django забезпечує використання ORM для роботи з базою даних, систем автентифікації

та авторизації, middleware, конфігураційних механізмів і адміністративного інтерфейсу, що дозволяє органічно вписати Django Ninja у вже існуючу архітектуру проекту. Фреймворк підтримує як синхронну, так і асинхронну обробку запитів, що дозволяє ефективно використовувати ресурси сервера та є особливо актуальним при взаємодії з зовнішніми сервісами або при обробці великої кількості одночасних підключень. У сфері безпеки використовуються перевірені механізми Django, включаючи захист від типових веб-загроз та підтримку різних схем автентифікації, таких як API-токени, API-ключі та JWT, що дозволяє реалізувати гнучкий контроль доступу до ресурсів. Додатково забезпечується валідація параметрів запитів різних типів, зокрема параметрів шляху, query-параметрів і тіла запиту, що підвищує надійність функціонування системи.

До основних переваг використання Django Ninja належать висока продуктивність, формалізація структури даних за рахунок типізації, автоматизована валідація та серіалізація, інтеграція з екосистемою Django, автоматична генерація документації, підтримка асинхронності та зручність масштабування API. У межах розробки серверної частини інформаційної системи KSU24 використання Django Ninja є обґрунтованим рішенням, оскільки дозволяє забезпечити ефективну взаємодію між клієнтською та серверною частинами, підвищити продуктивність, надійність і узгодженість роботи системи. Django Ninja виступає сучасним інструментом розробки веб-API, який поєднує принципи типізації, автоматизації обробки даних і високої продуктивності, що робить його доцільним для створення масштабованих інформаційних систем, які відповідають сучасним вимогам до якості програмного забезпечення.

Система управління базами даних PostgreSQL є однією з найбільш розвинених об'єктно-реляційних СУБД з відкритим вихідним кодом, яка широко застосовується у сучасних веб-застосунках [28]. Її популярність зумовлена поєднанням високої надійності, гнучкості архітектури та можливості ефективної роботи з великими обсягами структурованих даних. PostgreSQL використовується як у невеликих проєктах, так і у складних корпоративних та аналітичних системах, де критичним є стабільність та масштабованість. Однією з ключових особ-

ливостей PostgreSQL є її розширювана архітектура. Система дозволяє створювати користувацькі типи даних, функції, оператори та індексні методи без необхідності модифікації ядра. Такий підхід забезпечує адаптивність СУБД до різних предметних областей і дозволяє ефективно реалізовувати специфічні вимоги прикладних систем.

Важливою внутрішньою особливістю PostgreSQL є реалізація багатoversійного контролю конкурентного доступу (MVCC). Завдяки цьому механізму система забезпечує паралельне виконання операцій читання і запису без взаємного блокування [26]. Кожна транзакція працює з власною версією даних, що підвищує стабільність системи при високому навантаженні та великій кількості одночасних користувачів. PostgreSQL повністю підтримує принципи ACID, що забезпечує надійну та передбачувану обробку транзакцій у багатокористувацькому середовищі [36]. Атомарність гарантує, що всі операції в межах однієї транзакції виконуються як єдине ціле: або фіксуються повністю, або повністю відхиляються у разі виникнення помилки під час виконання будь-якого з етапів. Узгодженість забезпечує перехід бази даних з одного коректного стану в інший, відповідно до визначених обмежень цілісності даних. Ізольованість запобігає впливу паралельних транзакцій одна на одну, завдяки чому результати їх виконання не залежать від проміжних станів інших операцій, що виконуються одночасно. Довговічність гарантує збереження зафіксованих змін навіть у випадку збоїв живлення, помилок процесів або перезапуску сервера бази даних. Додатково PostgreSQL використовує механізм Write-Ahead Logging, який полягає у попередньому записі змін у спеціальний журнал перед їх фактичним застосуванням до основних файлів. Це дозволяє забезпечити відновлення стану системи після збоїв шляхом повторного відтворення або відкату незавершених транзакцій на основі журналу WAL. Завдяки цьому забезпечується цілісність даних і мінімізує ризик їх втрати навіть у випадках некоректного завершення роботи сервера або переривання процесу запису на диск.

Суттєвою перевагою PostgreSQL є її розвинена система оптимізації запитів. Планувальник запитів виконує аналіз можливих стратегій виконання SQL-

операцій і обирає оптимальний варіант на основі статистичних даних про таблиці, індекси та оціночну вартість операцій доступу до даних. Такий підхід дозволяє ефективно виконувати як прості запити, так і складні багатотабличні операції з join-зв'язками, мінімізуючи час виконання та використання ресурсів.

Окрему роль у підвищенні продуктивності відіграє система індексування [40]. PostgreSQL підтримує декілька типів індексів, зокрема B-tree, Hash, GIN та GiST, кожен з яких оптимізований під певні сценарії використання. Наприклад, B-tree є універсальним для більшості операцій порівняння, тоді як GIN застосовується для повнотекстового пошуку та роботи зі складними структурами даних, такими як масиви або JSONB. Це дозволяє значно скорочувати час доступу до даних у великих таблицях та підвищувати загальну ефективність системи.

Гнучкість PostgreSQL також проявляється у підтримці широкого спектра типів даних. Окрім базових числових, текстових та часових типів, система забезпечує роботу з форматами JSON та JSONB, що є важливим для сучасних веб-застосунків із напівструктурованими даними. Використання JSONB додатково дозволяє здійснювати індексацію та ефективний пошук всередині документів. Крім того, можливість створення користувацьких типів даних та функцій дає змогу адаптувати модель зберігання під специфіку предметної області конкретної системи.

У практичному застосуванні PostgreSQL часто використовується як основна система зберігання даних. У сучасних системах вона забезпечує стабільну роботу з даними, підтримку складних зв'язків між сутностями та надійну обробку запитів завдяки транзакціям. У LMS PostgreSQL використовується для зберігання структурованих даних про користувачів, навчальні дисципліни, результати оцінювання та адміністративні записи. У системі KSU24 вона виконує роль центрального компонента зберігання даних, забезпечуючи узгоджену взаємодію між модулями платформи та стабільну роботу при одночасному доступі великої кількості користувачів у період підвищеного навантаження.

Важливим аспектом є підтримка аналітичних можливостей PostgreSQL. Завдяки наявності агрегатних, групувальних та віконних функцій система до-

зволяє виконувати обробку та аналіз великих обсягів даних без залучення зовнішніх аналітичних інструментів. Це дає змогу значно оптимізувати процес аналітики, оскільки виконання подібних операцій на рівні бази даних є в рази більш ефективними, ніж їх проміжна обробка в програмному коді.

Масштабованість PostgreSQL забезпечується як вертикальним, так і горизонтальним підходом: перший передбачає збільшення ресурсів одного сервера, тоді як другий реалізується через реплікацію та балансування навантаження між кількома вузлами, що дозволяє підтримувати стабільну продуктивність навіть при значному зростанні кількості користувачів.

Безпека даних реалізована на багаторівневій основі за допомогою ролі-вої моделі доступу, яка дозволяє гнучко налаштовувати права користувачів на рівні бази даних, схем і операцій. Додатково використовується шифрування з'єднань через SSL/TLS, а також механізми аудиту дій користувачів, що забезпечує контроль усіх операцій у системі. PostgreSQL є кросплатформною системою управління базами даних, що дозволяє використовувати її в різних операційних середовищах без зміни прикладної логіки, спрощуючи розгортання інформаційних систем у локальних, хмарних і гібридних інфраструктурах.

Додатковою перевагою PostgreSQL є активний розвиток та підтримка великої спільноти розробників. Регулярні оновлення системи включають покращення продуктивності, розширення функціональності та підвищення рівня безпеки, що забезпечує її довгострокову актуальність як технологічного рішення. PostgreSQL є універсальною та технологічно зрілою платформою управління базами даних, яка поєднує високу продуктивність, надійність та масштабованість. Її використання у сучасних інформаційних системах дозволяє забезпечити ефективну обробку даних та стабільність.

## **2.2 Мета розробки та обґрунтування необхідності створення модулю цифрового розкладу**

Метою розробки даного модуля є спрощення взаємодії користувачів із розкладом занять та усунення типових труднощів, що виникають у процесі його

формування та використання. У межах існуючих підходів значна частина операцій має рутинний характер і виконується вручну, що ускладнює роботу диспетчерів та створює додаткові незручності для студентів і викладачів. Запропонований модуль дозволяє автоматизувати частину рутинних операцій, забезпечуючи більш зручний інтерфейс створення, редагування та перегляду розкладів занять, а також знизити ймовірність помилок, пов'язаних із людським фактором.

До впровадження системи цифрового розкладу занять, розклад формувався та зберігався у вигляді електронних таблиць, зокрема у форматі Excel. Такий підхід, незважаючи на свою простоту, має ряд недоліків. Зокрема, редагування великих обсягів даних у табличному вигляді є незручним та малоефективним, особливо в умовах частих змін. Диспетчери змушені вручну вносити корективи, що підвищує складність процесу та збільшує ймовірність виникнення помилок. Додатково слід зазначити, що при одночасній роботі кількох користувачів із файлом процес стає в рази складнішим: внесення змін фактично відбувається послідовно, оскільки кожен наступний користувач змушений очікувати завершення редагування попереднім. Це суттєво знижує ефективність колективної роботи та уповільнює оновлення розкладу. Крім того, відсутність централізованого сховища даних і механізмів версіонування призводить до ризику втрати актуальності інформації. Окремі копії файлу можуть розповсюджуватися між користувачами, що ускладнює контроль за їхньою актуальністю та створює передумови для використання застарілих або некоректних даних.

Проведений аналіз існуючого підходу до формування та використання розкладу дозволяє виокремити ряд ключових проблем, що потребують вирішення:

1. Відсутність централізованого управління розкладом та єдиного актуального джерела даних;
2. Складність редагування великих обсягів даних у форматі електронних таблиць;

3. Обмежені можливості одночасної роботи кількох користувачів над одним файлом розкладу занять;
4. Висока ймовірність виникнення помилок під час ручного внесення даних;
5. Відсутність персоналізованого доступу до розкладу для різних категорій користувачів;
6. Труднощі з контролем актуальності даних та відстеженням відповідних змін;
7. Відсутність механізмів історії змін та аудиту внесених коректив до розкладу;
8. Підвищене навантаження на диспетчерів унаслідок виконання рутинних дій;
9. Складність масштабування та адаптації розкладу під структурні зміни навчального процесу;
10. Відсутність інтеграції з іншими компонентами системи KSU24 та пов'язаними модулями.

Розроблений модуль дозволить вирішити низку зазначених проблем. Зокрема такий підхід може забезпечити централізоване зберігання та обробку даних. Також однією з ключових переваг цієї реалізації є можливість автоматичного відображення індивідуального розкладу занять для кожного користувача. Система підтягує необхідну інформацію відповідно до ролі користувача, що спрощує процес доступу до актуальних даних. Окрім цього, модуль надає диспетчерам зручні інструменти для створення, редагування та перегляду розкладу, що підвищує ефективність їх роботи та знижує навантаження, пов'язане з виконанням рутинних дій. Централізований підхід до управління даними дозволяє забезпечити їх узгодженість, актуальність та доступність для всіх учасників освітнього процесу. Впровадження модуля управління розкладом спрямоване на вирішення існуючих проблем, пов'язаних із використанням застарілих підходів, та створення більш ефективного, зручного і надійного інструменту для взаємодії з розкладом занять.

## 2.3 Архітектура та загальна структура модуля

Архітектура модуля цифрового розкладу побудована за багаторівневим принципом, що забезпечує чітке розділення відповідальності між компонентами системи та дозволяє ефективно організувати процес обробки даних [19]. Такий підхід підвищує чистоту та читабельність коду, спрощує масштабування функціоналу системи та розмежування бізнес-логіки, рівня доступу до даних і рівня представлення.

На рівні структури проєкту модуль реалізовано в межах директорії `apps/schedule_V2`, де розміщено основні доменні моделі. Даний рівень відповідає за опис структури та її відображення у базі даних. Моделі визначають основні сутності предметної області, зокрема елементи розкладу, пов'язані записи та допоміжні структури, що використовуються для формування цілісного представлення навчального процесу.

Наступним рівнем архітектури є шар доступу до даних DAO, який виконує роль проміжного компонента між бізнес-логікою та базою даних [12]. Його основне призначення полягає у виконанні операцій читання, створення, оновлення та видалення даних, при цьому ізоляція запитів до бази даних дозволяє уникнути дублювання логіки на рівні бізнес-компонентів.

Вищим рівнем є шар `features`, який відповідає за реалізацію бізнес-логіки модуля. Саме на цьому рівні відбувається основна обробка даних, перевірка бізнес-правил, логіка формування розкладу та взаємодії з DAO. Даний шар можна розглядати як аналог сервісного рівня (Service Layer) у класичних багаторівневих архітектурах [11]. `Features` інкапсулюють основну логіку роботи системи та забезпечують обробку даних відповідно до вимог предметної області.

Окремо слід виділити рівень контролерів, який реалізує HTTP-інтерфейс взаємодії із системою. Саме цей рівень приймає запити від клієнтської частини, виконує первинну валідацію вхідних даних та передає виконання до відповідних методів шару `features`. У межах даного підходу також використовується шар схем (`schemas`), який виконує допоміжну функцію в межах API-layer. Його основне призначення полягає у визначенні структур вхідних та вихідних даних,

що передаються через HTTP-інтерфейс. Фактично даний компонент реалізує механізм DTO та виконує функції мапінгу між внутрішніми моделями системи та зовнішнім представленням даних.

Загальний потік обробки запиту в модулі можна представити наступним чином (рис. 2.2): клієнтська частина системи надсилає HTTP-запит до API, який обробляється на рівні контролера. Далі запит передається до шару features, де виконується бізнес-логіка та здійснюються виклики методів DAO для отримання, створення, оновлення або видалення даних. Отримані результати повертаються до шару features для подальшої обробки та формування вихідної відповіді. Після цього дані перетворюються у структурований формат відповіді API за допомогою схем (schemas) і повертаються клієнтській частині системи.

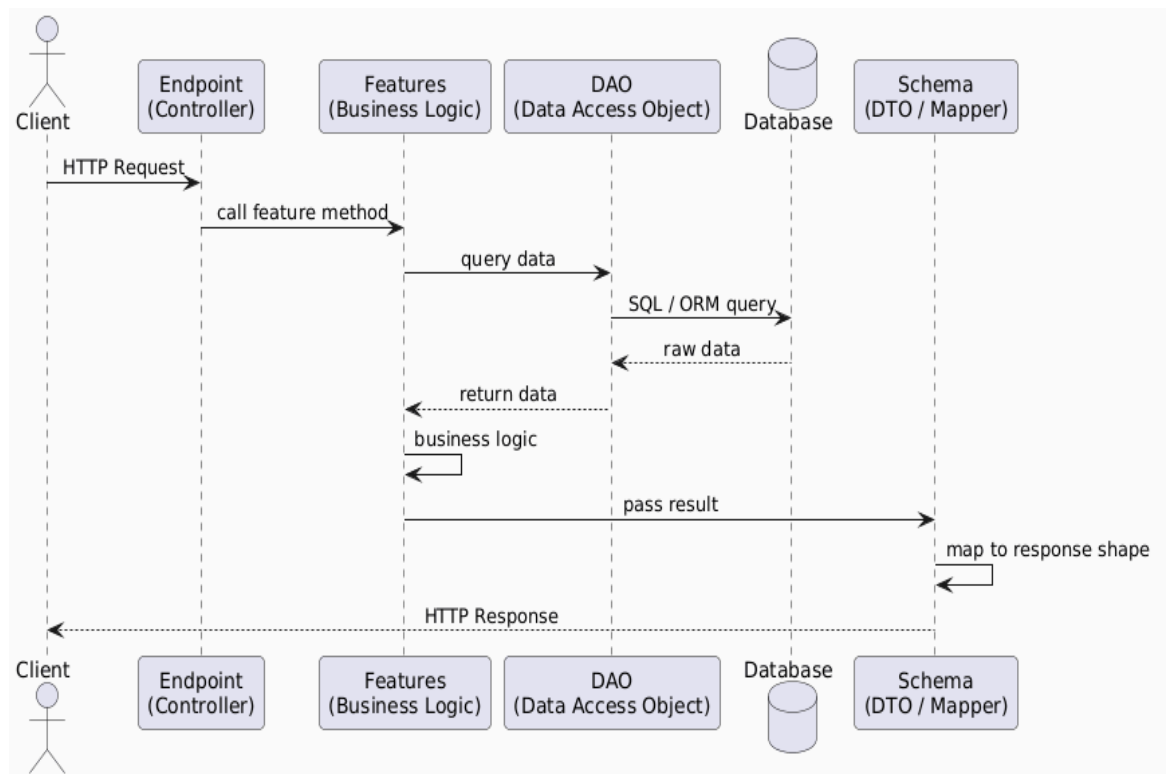


Рисунок 2.2. Sequence діаграма обробки запиту в модулі розкладу

## 2.4 Проєктування та реалізація моделей даних цифрового розкладу

Моделі даних модуля цифрового розкладу (рис. 2.3), побудовані на основі об'єктно-реляційного підходу та відображає ключові сутності предметної області, пов'язані з формуванням, зберіганням та відображенням навчального роз-

кладу. Такий підхід дозволяє забезпечити узгодженість структури даних із бізнес-логікою системи та спрощує подальше масштабування модулю. Основними сутностями є розклад — `Schedule`, заняття в розкладі — `ScheduleLesson`, а також механізм аудиту стану записів — `StatusAuditMixin`, який забезпечує контроль життєвого циклу даних, фіксацію змін при їх редагуванні та підтримання узгодженого стану у системі.

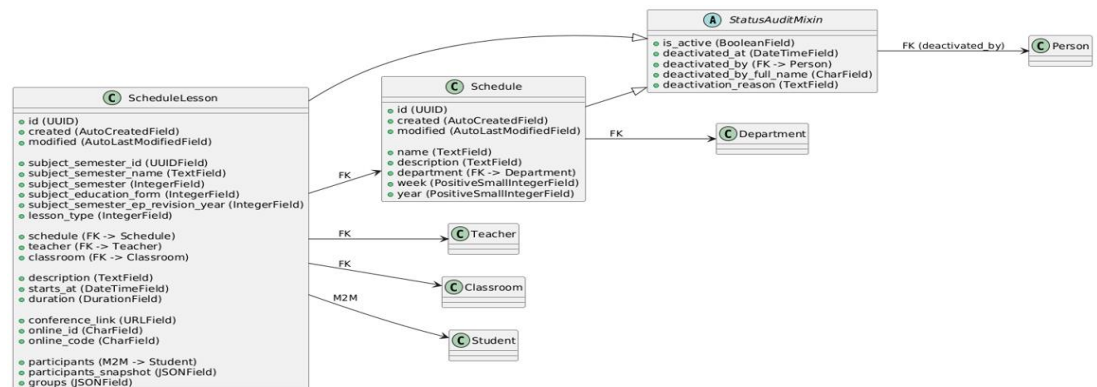


Рисунок 2.3. Діаграми класів моделей модуля цифрового розкладу

Сутність `Schedule` є базовою одиницею модуля та використовується для групування навчальних занять у межах певного календарного періоду. Вона формує логічну структуру розкладу та дозволяє об'єднувати заняття за тижнями, роками та структурними підрозділами університету.

- *Назва розкладу (name)*: текстове поле, призначене для ідентифікації розкладу та його відображення у користувацькому інтерфейсі.
- *Опис розкладу (description)*: Додаткове необов'язкове поле, яке використовується для зберігання розширеної інформації про розклад. Воно може містити пояснення щодо особливостей формування розкладу, його призначення або інших організаційних деталей.
- *Кафедра / структурний підрозділ (department)*: Поле реалізує зв'язок із сутністю структурного підрозділу університету. Воно визначає, до якої кафедри або факультету належить конкретний розклад. Це дозволяє розмежовувати доступ до даних та забезпечує їх розподіл між навчальними підрозділами.

- *Номер календарного тижня (week)*: Ціле число, яке визначає тиждень у межах календарного року (від 1 до 53). Це поле використовується для побудови розкладів за тижнями та дозволяє системі точно визначати часовий період, до якого належить набір занять.
- *Рік (year)*: Поле, що визначає календарний рік, в період якого був створений розклад.

Ця модель виконує роль верхньорівневої структури, яка об'єднує всі навчальні заняття в межах певного періоду та забезпечує логічну організацію розкладу як цілісного об'єкта.

ScheduleLesson є центральною одиницею модуля та описує конкретне навчальне заняття у розкладі. Вона містить повний набір атрибутів, необхідних для відображення, планування та проведення заняття, включаючи інформацію про дисципліну, викладача, часові рамки та учасників.

- *Ідентифікатор дисципліни (subject\_semester\_id)*: Унікальний ідентифікатор навчальної дисципліни в межах навчального плану. Використовується для зв'язку з модулем education\_process.
- *Назва дисципліни (subject\_semester\_name)*: Текстове поле, яке включає в себе назву навчальної дисципліни. Денормалізоване для оптимізації відображення даних без необхідності додаткових запитів до пов'язаних сутностей.
- *Семестр (subject\_semester)*: Визначає семестр, у якому викладається дисципліна. Це поле дозволяє сегментувати навчальний процес у межах навчального року.
- *Форма навчання (subject\_education\_form)*: Визначає форму навчання (денна, заочна тощо). Використовується для розмежування розкладів для різних категорій студентів.
- *Рік освітньої програми (subject\_semester\_ep\_revision\_year)*: Поле, яке дозволяє визначити версію навчальної програми, до якої належить дисципліна.

- *Тип заняття (lesson\_type)*: Визначає формат заняття: лекція, практика, лабораторна робота, тощо. Це поле використовується для класифікації заняття у межах розкладу.
- *Ідентифікатор розкладу до якого прив'язане заняття (schedule)*: Зовнішній ключ, який визначає приналежність заняття до конкретного розкладу. Це забезпечує ієрархічну структуру даних між сутностями Schedule та ScheduleLesson.
- *Ідентифікатор викладача який проводить заняття (teacher)*: Зв'язок з сутністю викладача, який проводить заняття. Це поле забезпечує зв'язок із кадровою частиною системи.
- *Ідентифікатор аудиторії у якій проходить заняття (classroom)*: Визначає приміщення, у якому проводиться заняття. Поле є необов'язковим, що забезпечує підтримку як аудиторних, так і онлайн заняття.
- *Опис заняття (description)*: Додаткове текстове поле, яке використовується для уточнення теми або змісту заняття.
- *Дата та час початку заняття (starts\_at)*: Визначає точний момент початку заняття. Це ключовий параметр для визначення часових рамок розкладу.
- *Тривалість заняття (duration)*: Поле, що визначає протяжність заняття у розкладі.
- *Покликання на онлайн-заняття (conference\_link)*: Використовується для організації дистанційних занять та забезпечує інтеграцію з сервісами для проведення відеоконференцій.
- *Ідентифікатор онлайн-заняття (online\_id)*: Додатковий технічний ідентифікатор, який використовується для інтеграції із зовнішніми системами відео-зв'язку.
- *Код доступу (online\_code)*: Забезпечує захищений доступ до онлайн-занять.

- *Учасники заняття (participants)*: Зв'язок many-to-many зі студентами, які беруть участь у занятті. Використовується для динамічного управління списком учасників.
- *Знімок учасників (participants\_snapshot)*: JSON-поле, яке зберігає фіксований стан списку учасників на момент створення заняття. Це дозволяє зберігати історичну узгодженість даних навіть у разі подальших змін у списках студентів.
- *Групи студентів (groups)*: JSON-поле, що містить перелік навчальних груп, які відвідують заняття. Використовується для швидкого відображення структури учасників заняття.

*ScheduleLesson* є ключовою сутністю модуля, оскільки саме вона формує фактичний зміст навчального розкладу та визначає всі параметри проведення занять. Додатково, вона забезпечує зв'язок між основними елементами розкладу, такими як викладачі, дисципліни та аудиторії, що дозволяє узгодити організацію навчального процесу та підтримувати цілісність даних у системі.

*StatusAuditMixin* є допоміжним абстрактним класом, який використовується для реалізації механізму контролю життєвого циклу записів. Він виступає аналогом логічного видалення (soft delete) та дозволяє відстежувати історію змін без фізичного видалення даних з бази [14].

- *Активність запису (is\_active)*: Булеве поле, що визначає, чи є запис доступним для використання в системі. Виступає базовим механізмом фільтрації активних даних.
- *Час деактивації (deactivated\_at)*: Фіксує момент, коли запис було деактивовано. Використовується для аудиту змін.
- *Ідентифікатор користувача, який деактивував запис (deactivated\_by)*: Вказує користувача системи, який виконав деактивацію запису, що дозволяє визначати відповідальність за зміни. Пов'язаний із сутністю *Person* у системі.
- *ПІБ користувача, який деактивував запис (deactivated\_by\_full\_name)*: Текстове представлення користувача що деактивував запис. Поле при-

значене для швидкого відображення користувача без додаткових зв'язків та запитів до бази даних.

- *Причина деактивації* (deactivation\_reason): Поле, яке використовується для фіксації причини деактивації запису, це забезпечує прозорість змін у системі.

Дана модель забезпечує контроль змін у системі, підвищує надійність зберігання даних та дозволяє реалізувати механізм історію змін без втрати інформації. На відміну від фізичного видалення записів, такий підхід дозволяє зберігати всі зміни та відновлювати попередній стан даних у разі необхідності.

Окрім зберігання атрибутів стану, міксин також реалізує логіку керування життєвим циклом записів через відповідні методи. Зокрема, метод *deactivate()* використовується для деактивації запису: він змінює статус активності, фіксує час виконання операції, зберігає користувача, який виконав дію, а також причину деактивації. При цьому передбачено перевірку поточного стану запису, що запобігає повторному виконанню операції для вже неактивних об'єктів. Метод *activate()* виконує зворотну операцію та дозволяє повторно активувати запис. У процесі активації відбувається очищення полів, пов'язаних із деактивацією, та відновлення статусу активності. Це забезпечує можливість гнучкого керування станом даних без їх повторного створення. Таким чином, поєднання атрибутів та поведінкових методів у моделі *StatusAuditMixin* дозволяє реалізувати механізм аудиту, який підвищує надійність системи, забезпечує контроль змін та підтримує цілісність даних у довгостроковій перспективі.

## 2.5 Реалізація механізму чорнового збереження розкладу

У процесі проєктування модуля окрему увагу було приділено реалізації механізму чернеток. Необхідність такого функціоналу обумовлена тим, що формування розкладу є ітеративним процесом, який передбачає багаторазове редагування, перевірку та погодження перед його остаточною публікацією. На початковому етапі було розглянуто підхід, що передбачав використання булевого поля *is\_draft* у межах основних моделей системи, таких як *Schedule* та

ScheduleLesson. Це поле мало б визначати, чи є відповідний запис чернеткою або вже опублікованою сутністю. Водночас зазначений підхід має ряд суттєвих недоліків. По-перше, він порушує принцип єдиного джерела істини (Single Source of Truth) [16], оскільки стан чернетки довелося б дублювати одночасно у двох пов'язаних моделях. У такому випадку виникає необхідність синхронізації цього поля між Schedule та всіма пов'язаними записами ScheduleLesson, це ускладнює бізнес-логіку та підвищує ризик виникнення неузгодженого стану даних. По-друге, такий підхід створює додаткове навантаження на рівні вибірки даних. У більшості сценаріїв система працює саме з опублікованими розкладами, тому практично кожен запит до бази даних мав би включати додаткову умову фільтрації (таку як, *is\_draft = false*). Це ускладнює запити, знижує їхню читабельність та потенційно впливає на продуктивність при роботі з великими обсягами даних.

З огляду на зазначені обмеження було прийнято рішення реалізувати механізм чернеток із використанням окремого внутрішнього сховища даних у системі KSU24, умовно позначеного як JARS. Цей підхід базується на збереженні структурованих даних у форматі JSON у базі даних PostgreSQL та у даному контексті використовується для роботи з проміжними станами розкладу [27]. Такий підхід є концептуально близьким до документоорієнтованих баз даних, зокрема таких, як MongoDB, оскільки дозволяє зберігати гнучкі, ієрархічні структури без жорсткої прив'язки до реляційної схеми. Основною перевагою такого рішення є повна ізоляція чернеток від основних доменних моделей. Чернетка зберігається як єдиний документ, що містить всю необхідну інформацію про розклад та пов'язані з ним заняття. Це дозволяє уникнути неконсистентних станів даних, оскільки всі зміни виконуються в межах одного об'єкта.

Структура JSON-схеми чернетки значною мірою повторює схеми, які використовуються в API системи цифрового розкладу. Зокрема, вона містить основні атрибути розкладу (назву, опис, ідентифікатор кафедри, номер тижня та рік), а також вкладений масив занять (lessons). При цьому така структура фактично поєднує в собі дані двох окремих доменних сутностей — Schedule та

ScheduleLesson у межах одного документа. Кожен елемент цього масиву описує окреме заняття та включає інформацію про дисципліну, тип заняття, викладача, часові параметри, а також додаткові атрибути, такі як посилання на відеоконференцію, групи студентів та учасників.

Публікація чернетки реалізується як окремий метод (рис. 2.4), що викликається через інтерфейс API. У нього передається відповідний ідентифікатор документа в системі JARS, після чого виконується трансформація JSON структури у реляційні моделі системи. На першому етапі створюється сутність розкладу (Schedule) на основі метаданих чернетки. Далі відбувається обробка списку занять, кожне з яких перетворюється на об'єкт ScheduleLesson. Особливістю реалізації є використання механізму групового створення записів (bulk create), що дозволяє виконати створення всіх занять у розкладі однією операцією запису до бази даних. Така оптимізація є доцільною, оскільки один розклад може містити значну кількість занять, і їх поетапне створення призводило б до збільшення кількості запитів до бази, що в свою чергу підвищило навантаження та загальний час виконання операції. Окремо здійснюється обробка учасників занять: їхні ідентифікатори витягуються з JSON-структури та встановлюються через відповідні зв'язки після створення основних сутностей. Також не менш важливою деталлю є те, що вся операція публікації виконується в межах транзакції. Це гарантує, що у випадку виникнення помилки жодна частина розкладу не буде збережена частково, що забезпечує узгодженість даних та відповідає вимогам до надійності системи [29].

Обрана реалізації механізму чернеток дозволяє досягти балансу між гнучкістю, продуктивністю та цілісністю даних. Він спрощує бізнес-логіку, мінімізує ризики неузгодженостей та забезпечує зручний підхід для поетапного формування розкладу перед його публікацією.

```

schedule.py
@classmethod 1 usage (1 dynamic)  Bohdan Metin
@transaction.atomic
def publish_draft_schedule(cls, draft_id: UUID) -> Schedule:
    document = cls.daos.jars_document.get_by_id(obj_id=draft_id)
    schedule_draft: dict = document.content
    lessons_draft: list = schedule_draft.pop("lessons")

    schedule: Schedule = cls.daos.schedule.create(**schedule_draft)

    lessons = []
    participants_map = []

    for lesson in lessons_draft:
        teacher: dict = lesson.pop("teacher", {})
        participants: list = lesson.pop("participants", [])
        participants_map.append(participants)

        lessons.append(
            ScheduleLesson(
                **lesson,
                schedule_id=schedule.id,
                teacher_id=teacher.get("id"),
                participants_snapshot=participants,
            )
        )

    created_lessons: list[ScheduleLesson] = cls.daos.schedule_lesson.bulk_create(lessons)

    for lesson, participants in zip(created_lessons, participants_map):
        if participant_ids := [p["id"] for p in participants]:
            lesson.participants.set(participant_ids)

    return schedule

```

Рисунок 2.4. Фрагмент коду публікації чернетки розкладу

## 2.6 Проєктування та реалізація REST API модуля цифрового розкладу

У сучасних веб-застосунках ключову роль у забезпеченні взаємодії між різними компонентами відіграє програмний інтерфейс застосунку. API являє собою набір визначених правил і механізмів, що дозволяють клієнтській частині системи взаємодіяти із сервером, отримувати доступ до даних та виконувати необхідні операції над ними. Використання API забезпечує чітке розмежування між клієнтською та серверною логікою, що дозволяє підвищити масштабованість системи. У межах модуля цифрового розкладу занять API виступає основним каналом доступу до функціональності системи, забезпечуючи виконання

CRUD операцій створення, отримання, оновлення та видалення даних, а також реалізацію додаткової бізнес-логіки, пов'язаної з формуванням і перевіркою розкладу. Проєктування API здійснювалось відповідно до принципів REST, що передбачає використання стандартних HTTP-методів, уніфікованих форматів запитів і відповідей, а також чітко визначеної структури ендпоінтів [10]. Такий підхід забезпечує передбачувану поведінку і зручний процес інтеграції API як для внутрішніх, так і для зовнішніх клієнтів системи [20].

У процесі розробки API окрему увагу було приділено розмежуванню зон відповідальності користувачів та контролю доступу до функціональності модуля розкладу. Це зумовлено необхідністю забезпечення коректної взаємодії різних категорій користувачів із системою, а також запобігання несанкціонованим змінам даних. Для реалізації даного підходу в системі KSU24 використовується механізм *sarability*, який дозволяє визначати набір конкретних дозволів для кожного користувача або груп користувачів [9]. На відміну від жорстко фіксованих ролей, такий підхід забезпечує більш гнучке налаштування доступу до окремих операцій та ресурсів в системі.

У модулі розкладу занять реалізовано базовий набір дозволів, який відповідає основним операціям роботи з даними:

- *schedule:schedule:read* — надає можливість перегляду інформації пов'язану з розкладом занять;
- *schedule:schedule:write* — дозволяє створювати нові записи у системі цифрового розкладу;
- *schedule:schedule:update* — забезпечує можливість внесення змін до існуючих записів;
- *schedule:schedule:delete* — дозволяє видаляти записи розкладу та його занять.
- *schedule:schedule:read\_individual* — доступ надається автоматично всім авторизованим користувачам та забезпечує доступ до перегляду індивідуального розкладу, сформованого відповідно до їхньої ролі в системі.

У процесі розробки серверної частини було реалізовано набір API-ендпоінтів, що забезпечують взаємодію клієнтської частини із системою. Дані ендпоінти покривають основні сценарії роботи з розкладом, включаючи створення, копіювання та подальше управління даними. Кожен ендпоінт спроектований відповідно до принципів REST-архітектури та супроводжується механізмом перевірки прав доступу на основі capability. Це забезпечує виконання лише тих операцій, на які користувач має відповідні повноваження.

Зокрема, у межах модуля розкладу реалізовано такі групи ендпоінтів:

1. Розклад занять

- *create\_schedule*

- HTTP-метод: POST
- URL: /schedule
- Авторизація: schedule:schedule:write

Забезпечує створення нового об'єкта розкладу на основі переданих вхідних даних. У процесі виконання запиту здійснюється валідація отриманої інформації, після чого відбувається збереження розкладу в системі.

- *copy\_schedule*

- HTTP-метод: POST
- URL: /schedule/{schedule\_id}/copy
- Авторизація: schedule:schedule:write

Ендпоінт дозволяє створити новий розклад на основі вже існуючого, включаючи всі пов'язані заняття та учасників. При цьому дати проведення занять автоматично коригуються відповідно до нових параметрів, зокрема дати початку розкладу та його занять.

- *patch\_schedule*

- HTTP-метод: PATCH
- URL: /schedule/{schedule\_id}
- Авторизація: schedule:schedule:update

Даний ендпоінт призначений для часткового оновлення даних існуючого розкладу занять. У разі відсутності розкладу з вказаним ідентифікатором повертається відповідна помилка.

- *get\_schedule*
  - HTTP-метод: GET
  - URL: /schedule/{schedule\_id}
  - Авторизація: schedule:schedule:read

Отримання детальної інформації про конкретний розклад занять за його унікальним ідентифікатором. У разі відсутності розкладу з вказаним ідентифікатором повертається відповідна помилка.

- *list\_schedule*
  - HTTP-метод: GET
  - URL: /schedule
  - Авторизація: schedule:schedule:read

Повертає список розкладів та його занять із підтримкою фільтрації та пагінації результатів.

- *deactivate\_schedule*
  - HTTP-метод: POST
  - URL: /schedule/{schedule\_id}/deactivate
  - Авторизація: schedule:schedule:write

Цей ендпоінт призначений для деактивації існуючого розкладу занять. У процесі виконання запиту передається причина деактивації, після чого відповідний розклад та пов'язані з ним заняття позначаються як неактивні у системі. Додатково фіксується користувач, який ініціював дану операцію. У разі відсутності розкладу з вказаним ідентифікатором повертається відповідна помилка.

- *publish\_draft\_schedule*
  - HTTP-метод: POST
  - URL: /schedule/publish\_draft/{draft\_id}
  - Авторизація: schedule:schedule:write

Публікація чернетки розкладу занять. У процесі виконання запиту здійснюється перевірка документа чернетки в JARS за її ідентифікатором, після чого вона переводиться у стан активного розкладу та стає доступною для подальшого використання в системі.

## 2. Заняття у розкладі

- *create\_schedule\_lesson*
  - HTTP-метод: POST
  - URL: /schedule/{schedule\_id}/lessons
  - Авторизація: schedule:schedule:write

Створює нове заняття в межах існуючого розкладу. У процесі виконання запиту здійснюється валідація переданих даних заняття та перевірка існування відповідного розкладу. Після успішної перевірки створюється новий запис заняття, який прив'язується до зазначеного розкладу.

- *get\_schedule\_lesson*
  - HTTP-метод: GET
  - URL: /schedule/lessons/{lesson\_id}
  - Авторизація: schedule:schedule:read

Ендпоінт призначений для отримання детальної інформації про конкретне заняття у розкладі за його унікальним ідентифікатором. У разі відсутності заняття з вказаним ідентифікатором повертається відповідна помилка.

- *list\_schedule\_lesson*
  - HTTP-метод: GET
  - URL: /schedule/lessons
  - Авторизація: schedule:schedule:read

Ендпоінт призначений для отримання списку занять у розкладі з можливістю застосування фільтрації та пагінації результатів. У відповідь повертає список занять, що відповідають заданим критеріям пошуку.

- *patch\_schedule\_lesson*
  - HTTP-метод: PATCH
  - URL: /schedule/lessons/{lesson\_id}

- Авторизація: schedule:schedule:update

Часткове оновлення існуючого заняття у розкладі. У разі відсутності заняття з вказаним ідентифікатором повертається відповідна помилка.

- *deactivate\_schedule\_lesson*
  - HTTP-метод: POST
  - URL: /schedule/lessons/{lesson\_id}/delete
  - Авторизація: schedule:schedule:delete

Ендпоінт призначений для деактивації заняття у розкладі. У процесі виконання запиту передається причина деактивації, після чого відповідний запис позначається як неактивний. Додатково фіксується користувач, який ініціював дану операцію. У разі відсутності заняття з вказаним ідентифікатором повертається відповідна помилка.

- *list\_individual\_schedule*
  - HTTP-метод: GET
  - URL: /schedule/individual
  - Авторизація: schedule:schedule:read\_individual

Дозволяє отримати списку індивідуальних занять у розкладі, що пов'язані з конкретним користувачем. У процесі виконання запиту формується вибірка індивідуальних занять із урахуванням ролі (викладач, студент) поточного користувача в системі. Результат повертається у вигляді списку занять.

### 3. Довідкові дані для формування розкладу

- *list\_department*
  - HTTP-метод: GET
  - URL: /schedule/departments
  - Авторизація: schedule:schedule:read

Ендпоінт призначений для отримання списку структурних підрозділів. У процесі виконання запиту застосовуються передані параметри фільтрації, після чого формується відповідна вибірка підрозділів із системи. Результат повертається у вигляді списку структурних підрозділів, що відповідають заданим критеріям пошуку.

- *list\_student\_group*
  - HTTP-метод: GET
  - URL: /schedule/student\_groups
  - Авторизація: schedule:schedule:read

Отримання списку груп студентів та їх учасників із можливістю застосування фільтрів та пагінації. Результат повертається у вигляді списку груп, що відповідають заданим критеріям пошуку.

- *list\_subject\_semester*
  - HTTP-метод: GET
  - URL: /schedule/subjects
  - Авторизація: schedule:schedule:read

Отримання списку навчальних дисциплін у межах семестру з можливістю застосування фільтрації та пагінації результатів. Повертає список навчальних дисциплін, що відповідають заданим критеріям пошуку.

- *list\_teacher*
  - HTTP-метод: GET
  - URL: /schedule/teachers
  - Авторизація: schedule:schedule:read

Призначений для отримання списку викладачів із можливістю застосування фільтрації та пагінації результатів. У процесі виконання запиту використовуються передані параметри фільтрації, після чого формується вибірка викладачів яка відповідає критеріям фільтрів.

## ВИСНОВКИ

У межах кваліфікаційної роботи було спроектовано та реалізовано серверну частину модуля цифрового розкладу занять у межах віртуального освітнього середовища KSU24.

У першому розділі досліджено специфіку організації навчального процесу у вітчизняних закладах вищої освіти. Розглянуто ключові складові, що беруть участь у формуванні розкладу: навчальні плани, академічні групи, навантаження викладачів та аудиторний фонд. Проаналізовано основні підходи до формування розкладу, визначено їх переваги та обмеження залежно від масштабу навчального закладу й рівня цифровізації його процесів. Окрім того, формалізовано основні критерії якості розкладу, які було покладено в основу вимог до спроектованої системи.

У другому розділі розглянуто технологічний стек, що застосовується при розробці модулів у системі KSU24, зокрема мову програмування Python, фреймворк Django, надбудову Django Ninja для побудови REST API та систему управління базами даних PostgreSQL. Вивчено архітектурні принципи побудови модулів, зокрема багаторівневий підхід із розмежуванням на шари моделей, доступу до даних, бізнес-логіки та контролерів. Спроектовано структуру основних моделей даних: Schedule, ScheduleLesson та StatusAuditMixin. Розроблено механізм чернеток розкладу з використанням внутрішнього сховища JARS, що забезпечує збереження проміжних станів розкладу у форматі JSON до моменту його офіційної публікації. Реалізовано набір API-ендпоінтів модуля, який охоплює операції управління розкладами та заняттями, а також механізм розмежування прав доступу користувачів на основі системи capability. У підсумку, в межах роботи було виконано наступні завдання:

- проаналізовано підходи до організації та цифровізації розкладу занять у закладах вищої освіти;
- досліджено критерії якості розкладу та основні вимоги до інформаційної системи;

- проаналізовано технологічний стек та архітектурні рішення, що застосовуються у системі KSU24;
- розроблено структуру моделей даних, яка відображає ключові сутності та взаємозв'язки предметної області розкладу занять і забезпечує цілісне представлення навчального процесу на рівні бази даних.
- впроваджено механізм контролю життєвого циклу записів на основі журналу аудиту;
- реалізовано механізм керування чернетками розкладу із застосуванням сховища JARS;
- імплементовано повний комплекс REST-ендпоінтів для управління розкладом занять, що забезпечує підтримку операцій створення, копіювання, оновлення, деактивації записів, а також формування персоналізованого розкладу для кожного користувача системи.

Розроблений модуль спрямований на вирішення актуальних проблем, притаманних традиційному підходу до ведення розкладу засобами електронних таблиць. Зокрема, до основних недоліків існуючого підходу належать: відсутність централізованого управління даними, обмежені можливості організації колективної роботи, а також підвищений ризик внесення некоректних даних внаслідок людського фактору.

Передбачається, що впровадження розробленого модуля в інформаційній системі KSU24 забезпечить більш ефективний та надійний інструментарій для всіх учасників навчального процесу. Крім того, реалізований модуль може слугувати архітектурною основою для подальшого розширення та вдосконалення функціональних можливостей підсистеми управління розкладом.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Abubakar A. et al. A Comparative Analysis of Manual and Automatic Timetabling Approaches for Resource Utilisation in Tertiary Higher Learning Institution // *International Journal of Computer Science and Mobile Computing (IJCSMC)*. — 2024. — Vol. 13, No. 12. — DOI: 10.47760/IJCSMC.2024.V13I12.007.
2. Babaei H., Karimpour J., Hadidi A. A survey of approaches for university course timetabling problem // *Computers & Industrial Engineering*. — 2015. — Vol. 86. — P. 43–59. — DOI: 10.1016/j.cie.2014.11.010.
3. Ceschia S., Di Gaspero L., Schaerf A. Educational Timetabling: Problems, Benchmarks, and State-of-the-Art Results // *European Journal of Operational Research*. — 2023. — Vol. 308, No. 1. — P. 1–18. — DOI: 10.1016/j.ejor.2022.07.011.
4. Chen R., Goh S. L., Sabar N. R., Kendall G. A Survey of University Course Timetabling Problem: Perspectives, Trends and Opportunities // *IEEE Access*. — 2021. — Vol. 9. — P. 104151–104172. — URL: <https://www.graham-kendall.com/papers/chenetal2021a.pdf> (дата звернення: 14.03.2026).
5. Development of a Learning Management System for Python Programming Language. — Academia.edu. — URL: [https://www.academia.edu/125632623/Development\\_of\\_a\\_Learning\\_Management\\_System\\_for\\_Phyton\\_Programming\\_Language](https://www.academia.edu/125632623/Development_of_a_Learning_Management_System_for_Phyton_Programming_Language) (дата звернення: 03.04.2026).
6. Django Software Foundation. Admin site // *Django Documentation*. — URL: <https://docs.djangoproject.com/en/stable/ref/contrib/admin/> (дата звернення: 15.04.2026).
7. Django Software Foundation. Authentication and Authorization // *Django Documentation*. — URL: <https://docs.djangoproject.com/en/5.2/topics/auth/> (дата звернення: 10.04.2026).
8. Django Software Foundation. *Django Documentation*. — Version 5.2. — URL: <https://docs.djangoproject.com/en/5.2/> (дата звернення: 02.04.2026).

9. Ferraiolo D., Sandhu R., Gavrila S., Kuhn D. R., Chandramouli R. Proposed NIST Standard for Role-Based Access Control // *ACM Transactions on Information and System Security*. — 2001. — Vol. 4, No. 3. — P. 224–274. — DOI: 10.1145/501978.501980.
10. Fielding R. T. *Architectural Styles and the Design of Network-based Software Architectures* : PhD dissertation. — University of California, Irvine, 2000. — URL: [https://roy.gbiv.com/pubs/dissertation/rest\\_arch\\_style.htm](https://roy.gbiv.com/pubs/dissertation/rest_arch_style.htm) (дата звернення: 08.04.2026).
11. Fowler M. *Patterns of Enterprise Application Architecture*. — Boston : Addison-Wesley, 2002. — P. 143.
12. Fowler M. *Patterns of Enterprise Application Architecture*. — Boston : Addison-Wesley, 2002. — P. 151.
13. Fowler M. *Patterns of Enterprise Application Architecture*. — Boston : Addison-Wesley, 2002. — P. 165–169.
14. Fowler M. Temporal Patterns // [martinfowler.com](http://martinfowler.com). — URL: <https://martinfowler.com/eaDev/timeNarrative.html> (дата звернення: 12.04.2026).
15. Горошко А. Мобільні додатки в освіті: розробка інтелектуального Android-застосунку для поліпшення інформаційної доступності розкладу занять // *Вимірювальна та обчислювальна техніка в технологічних процесах*. — 2024. — № 1. — С. 13–20. — DOI: 10.31891/2219-9365-2024-77-2.
16. Kleppmann M. *Designing Data-Intensive Applications*. — Sebastopol : O'Reilly Media, 2017. — P. 397–398.
17. Kucheryaviy V. Django Ninja — Fast Django REST Framework. — URL: <https://django-ninja.dev/> (дата звернення: 05.04.2026).
18. Lutz M. *Learning Python: Powerful Object-Oriented Programming*. — 5th ed. — Sebastopol : O'Reilly Media, 2013. — 1594 p.
19. Martin R. C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. — Boston : Prentice Hall, 2017. — P. 161.

20. Microsoft Azure Architecture Center. Best Practices for RESTful Web API Design. — URL: <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design> (дата звернення: 09.04.2026).
21. Міністерство освіти і науки України. Стандарт вищої освіти. — 2022. — URL: <https://mon.gov.ua/osvita-2/vishcha-osvita-ta-osvita-doroslikh/naukovo-metodichna-rada-ministerstva-osviti-i-nauki-ukraini/zatverdzeni-standarti-vishchoi-osviti> (дата звернення: 10.02.2026).
22. Murray K., Müller T., Rudová H. Automated System for University Timetabling // *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006)*. — 2006. — URL: <https://www.unitime.org/papers/patat06.pdf> (дата звернення: 20.02.2026).
23. Necula S.-C. Exploring The Model-View-Controller (MVC) Architecture: A Broad Analysis of Market and Technological Applications // *Preprints*. — 2024. — DOI: 10.20944/preprints202404.1860.v1.
24. OECD. Learning Management Systems and Other Digital Tools for System and Institutional Management // *OECD Digital Education Outlook 2023*. — Paris : OECD Publishing, 2023. — URL: [https://www.oecd.org/en/publications/oecd-digital-education-outlook-2023\\_c74f03de-en](https://www.oecd.org/en/publications/oecd-digital-education-outlook-2023_c74f03de-en) (дата звернення: 15.02.2026).
25. OpenAPI Initiative. *OpenAPI Specification*. — Version 3.1. — URL: <https://spec.openapis.org/oas/latest.html> (дата звернення: 07.04.2026).
26. PostgreSQL Global Development Group. Concurrency Control // *PostgreSQL Documentation*. — URL: <https://www.postgresql.org/docs/current/mvcc-intro.html> (дата звернення: 04.04.2026).
27. PostgreSQL Global Development Group. JSON Types // *PostgreSQL Documentation*. — URL: <https://www.postgresql.org/docs/current/datatype-json.html> (дата звернення: 04.04.2026).
28. PostgreSQL Global Development Group. PostgreSQL: About. — URL: <https://www.postgresql.org/about/> (дата звернення: 03.04.2026).

29. PostgreSQL Global Development Group. Transactions // *PostgreSQL Documentation*. — URL: <https://www.postgresql.org/docs/current/tutorial-transactions.html> (дата звернення: 04.04.2026).
30. Про вищу освіту : Закон України від 01.07.2014 р. № 1556-VII. — URL: <https://zakon.rada.gov.ua/laws/show/1556-18> (дата звернення: 05.02.2026).
31. Pydantic Team. *Pydantic Documentation*. — URL: <https://docs.pydantic.dev/latest/> (дата звернення: 06.04.2026).
32. Python Software Foundation. *The Python Tutorial*. — URL: <https://docs.python.org/3/tutorial/index.html> (дата звернення: 02.04.2026).
33. Rudová H., Müller T., Murray K. Rapid Development of University Course Timetables // *Proceedings of the 5th International Conference on Multidisciplinary Scheduling (MISTA 2011)*. — 2011. — P. 445–449. — URL: <https://www.unitime.org/papers/mista11.pdf> (дата звернення: 25.02.2026).
34. Rudová H., Murray K. University Course Timetabling with Soft Constraints // Burke E., De Causmaecker P. (eds.) *Practice and Theory of Automated Timetabling IV*. LNCS, vol. 2740. — Springer, 2003. — P. 310–328. — DOI: 10.1007/978-3-540-45157-0\_21.
35. Sharafuddin H. Automated University Timetabling: An AI Approach for Efficient Schedule Management // *International Journal of Emerging Technologies in Learning (iJET)*. — 2026. — Vol. 21, No. 2. — P. 91–99. — DOI: 10.3991/ijet.v21i02.59329.
36. Silberschatz A., Korth H. F., Sudarshan S. *Database System Concepts*. — 7th ed. — New York : McGraw-Hill, 2019. — P. 802–804.
37. Stuttard D., Pinto M. *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*. — 2nd ed. — Indianapolis : Wiley, 2011. — 912 p.
38. Vieira P. F. G., Conte V. C. B., Siqueira P. H. Class Timetable Allocation in Higher Education Using Binary Integer Programming // *Journal of Education and Learning*. — 2026. — Vol. 15, No. 3. — DOI: 10.5539/jel.v15n3p24.
39. Williams K. M., Shapiro T. M. Academic Achievement Across the Day: Evidence from Randomized Class Schedules // *Economics of Education Review*. — 2018.

— Vol. 67. — P. 158–170. — URL:

<https://www.sciencedirect.com/science/article/abs/pii/S0272775717305150> (дата звернення: 10.03.2026).

40. Ярцев В. П. *Організація баз даних та знань* : навч. посіб. — Київ : ДУТ, 2018. — С. 54–56.