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

ПРОЄКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ АВТОМАТИЗОВАНОГО ОБЛІКУ НАВЧАЛЬНОЇ АКТИВНОСТІ ЗДОБУВАЧІВ У ZOOM

Кваліфікаційна робота (проєкт)
на здобуття ступеня вищої освіти «бакалавр»

Виконав: здобувач 4 курсу 441 групи
Спеціальності 121 Інженерія програмного забезпечення
Освітньо-професійної програми Інженерія програмного забезпечення
Кусік І. А.
Керівник старша викладачка Черненко І. Є.
Рецензент Левченко Д.О., Провідний спеціаліст із FullStack розробки в компанії "Ivorysoft", ФОП Левченко Дмитро Олександрович

Івано-Франківськ – 2026

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API	Application Programming Interface – інтерфейс програмування застосунків
БД	база даних
DDD	Domain-Driven Design – предметно-орієнтоване проєктування
DI	Dependency Injection – впровадження залежностей
HTTP	HyperText Transfer Protocol – протокол передачі гіпертекстових документів
JSON	JavaScript Object Notation – текстовий формат обміну даними
JWT	JSON Web Token – стандарт токена автентифікації
KSU24	Електронне освітнє середовище Херсонського державного університету
LMS	Learning Management System – система управління навчанням
LTI	Learning Tools Interoperability – стандарт інтероперабельності навчальних інструментів
OAuth	Open Authorization – відкритий стандарт авторизації
ORM	Object-Relational Mapping – об’єктно-реляційне відображення
REST	Representational State Transfer – архітектурний стиль побудови API
SQL	Structured Query Language – мова структурованих запитів до БД
SSO	Single Sign-On – технологія єдиного входу
UUID	Universally Unique Identifier – універсальний унікальний ідентифікатор
ВОС	віртуальне освітнє середовище
ЗВО	заклад вищої освіти
НПП	науково-педагогічний працівник
ПЗ	програмне забезпечення

ВСТУП

Актуальність теми. В умовах війни та релокації цифрові технології відіграють ключову роль у збереженні інституційної спроможності українських закладів вищої освіти [2] , що зумовлює потребу в послідовній оптимізації та автоматизації рутинних академічних процедур. Платформи відеоконференцзв'язку, зокрема Zoom, формують масив аналітичних даних про навчальну активність здобувачів освіти [5] , який може бути використаний у межах освітньої аналітики. Водночас їхня функціональна ізольованість від внутрішньої інформаційної екосистеми закладу вищої освіти зумовлює необхідність дублювання облікових операцій та ручного внесення даних до електронних журналів. Цей процес ускладнюється неструктурованими ідентичностями користувачів та фрагментацією сесій через нестабільний зв'язок. Тому розробка програмного модуля, здатного алгоритмічно обробляти ці дані, проводити консолідацію мікророзривів зв'язку та автоматично формувати звіти про відвідуваність, є актуальним завданням, що дозволить вивільнити час викладачів, нівелювати людський фактор і забезпечити адміністрацію інструментами для оперативного моніторингу освітнього процесу.

Ступінь наукової розробки теми. Питання інтеграції платформ відеоконференцзв'язку з освітніми системами досліджували О. Лемещук, О. Співаковський та М. Новіков. Зокрема, М. Новіков реалізував проксі-мікросервіс ConferenceAPI для взаємодії KSU24 із Zoom. Водночас у наявних рішеннях відсутній механізм консолідації фрагментованих сесій та відсутня нативна інтеграція з Django-екосистемою, що визначає актуальність цього дослідження.

Об’єкт дослідження: процес автоматизованого обліку навчальної активності, зокрема відвідуваності, здобувачів вищої освіти у віртуальному середовищі.

Предмет дослідження: архітектура, алгоритми та програмні засоби інтеграції даних платформи Zoom для валідації фактичної присутності та автоматичного формування звітності у віртуальному освітньому середовищі закладу вищої освіти.

Мета дослідження: проєктування та розробка програмного модуля на основі патерну «Порти та адаптери», що забезпечує збір даних з Zoom, алгоритмічну консолідацію розірваних сесій та автоматичне формування звіту-рекомендації для викладача щодо відмічування присутності в електронному журналі.

Відповідно до поставленої мети визначено такі **завдання**:

- Проаналізувати проблематику ручного обліку відвідуваності в умовах дистанційного навчання та описати специфіку обробки фрагментованих сесій.
- Здійснити огляд існуючих рішень інтеграції платформ відеоконференцзв’язку з освітніми середовищами та визначити їх архітектурні обмеження.
- Провести порівняльний аналіз методів збору даних з Zoom API та обґрунтувати вибір асинхронної моделі через Past Meeting API.
- Спроєктувати архітектуру модуля на основі патерну «Порти та адаптери» та визначити доменну модель із переліком сутностей, портів і юз-кейсів.
- Розробити алгоритми ідентифікації учасників за корпоративною електронною адресою та консолідації фрагментованих сесій, а також спроєктувати реляційну структуру бази даних.
- Реалізувати програмний модуль мовою Python у фреймворку Django з інтеграцією Zoom API та внутрішніх сервісів KSU24.

- Провести автоматизоване тестування юз-кейсів модуля з ізоляцією інфраструктурних залежностей через mock-об’єкти.

Методи дослідження. У роботі застосовано системний аналіз для декомпозиції предметної галузі, порівняльний аналіз для обґрунтування архітектурних та технічних рішень, предметно-орієнтоване проєктування (DDD) для моделювання доменних сутностей, математичне моделювання для формалізації алгоритму консолідації сесій, а також методи автоматизованого тестування програмного забезпечення.

Практичне значення результатів. Розроблений програмний модуль інтегровано до віртуального освітнього середовища KSU24 Херсонського державного університету та включено до базової конфігурації збірки платформи разом із комплектом навчальних відеоматеріалів. Модуль продемонстровано закладам вищої освіти, що використовують платформу KSU24, як одну з ключових функціональних можливостей; за результатами демонстрації функціональність автоматизованого обліку відвідуваності отримала позитивну оцінку керівництва закладу-партнера як зручний інструмент для викладачів. Застосована архітектура «Порти та адаптери» забезпечує перспективу масштабування рішення на інші платформи відеоконференцзв’язку без модифікації доменної логіки.

Структура та обсяг роботи. Дипломна робота складається зі вступу, трьох розділів основної частини, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ВИМОГ ДО СИСТЕМИ

Передмова до розділу. Метою даного розділу є комплексне дослідження проблематики ручного обліку відвідуваності в умовах дистанційного навчання та аналіз існуючих підходів до автоматизації цього процесу. У розділі розглядаються технічні складнощі, пов'язані з ізольованістю даних у хмарних сервісах відеоконференцзв'язку, та проводиться критичний огляд попередніх спроб інтеграції платформи Zoom у віртуальне освітнє середовище закладу вищої освіти. Отримані результати дозволять обґрунтувати вибір методів збору даних та архітектурних патернів для розробки модуля консолідації звітності.

1.1 Проблема обліку в дистанційній освіті

Пандемія COVID-19 стала каталізатором масового переходу закладів вищої освіти на дистанційний формат навчання, що кардинально змінило вимоги до цифрової інфраструктури освітнього процесу [1] . В Україні цей процес набув додаткового загострення з початком повномасштабного вторгнення у 2022 році: релокація закладів вищої освіти та необхідність забезпечення безперервності навчання в екстремальних умовах перетворили дистанційне навчання з допоміжного формату на фундаментальний інструмент збереження інституційної спроможності [2] . Цифрові технології, зокрема платформи відеоконференцзв'язку, стали критичною інфраструктурою, яка замінила фізичні аудиторії та забезпечила синхронну взаємодію між учасниками освітнього процесу незалежно від їхнього географічного розташування [3] .

Проте швидка цифровізація виявила суттєву проблему, а саме ізольованість даних у хмарних сервісах [4] . Платформи

відеоконференцзв’язку, зокрема Zoom, що набула широкого застосування в освітньому секторі [4] , у процесі кожного заняття формують масив аналітичних даних щодо активності користувачів: час входу, тривалість перебування, технічні параметри з’єднання тощо . Однак ці дані залишаються на сторонніх серверах провайдера послуг, не маючи прямого зв’язку з внутрішніми інформаційними системами закладу вищої освіти (зокрема, з платформою KSU24). Внаслідок цього віртуальні освітні середовища часто виконують обмежену роль «агрегатора посилань», де система лише перенаправляє здобувача вищої освіти на зовнішній ресурс, втрачаючи контроль над його подальшою активністю [4] . Для викладачів це означає необхідність щоденного виконання рутинної роботи: вручну звіряти списки учасників конференції, копіювати статистику та переносити її в електронні журнали, що призводить до дублювання функцій та значних часових витрат [6] .

Важливим аспектом проблеми є низька якість вхідних даних, так звані «брудні» дані. Вона проявляється у неструктурованості ідентичностей користувачів. Здобувачі освіти нерідко використовують особисті акаунти або пристрої з некоректними псевдонімами (наприклад, технічні назви гаджетів «iPhone», «User-1» або скорочені імена), що робить однозначну автоматичну ідентифікацію здобувача вищої освіти за його іменем у Zoom практично неможливою без застосування спеціальних алгоритмів зіставлення з реальною базою даних закладу.

Додатковим дестабілізуючим фактором є фрагментація сесій, спричинена дискретністю логів [7] . В умовах нестабільного інтернет-зв’язку, характерного для періодів енергетичної кризи чи військових дій [2] , здобувачі освіти часто стикаються з мікророзривами з’єднання. Як наслідок, у системних логах Zoom одна людина протягом однієї академічної пари може відображатися як численні окремі підключення з різною тривалістю [7] . Це створює проблему валідації фактичної

присутності: просте сумування часу або фіксація лише одного факту входу не дає об’єктивної картини залученості здобувача освіти до заняття.

У рамках даного дослідження як цільове віртуальне освітнє середовище обрано платформу KSU24 [8] , що є інформаційною системою управління освітнім процесом Херсонського державного університету, побудованою на технологічному стеку Python/Django. KSU24 забезпечує централізоване управління розкладом, електронними журналами та обліком успішності здобувачів вищої освіти. Вибір цієї платформи як об’єкта інтеграції зумовлений практичною потребою автоматизації обліку відвідуваності у конкретному закладі, проте запропоновані архітектурні рішення не прив’язані до конкретної платформи та потенційно можуть бути адаптовані до інших віртуальних освітніх середовищ.

Таким чином, розрив між технічними можливостями збору даних у Zoom та реальними потребами адміністративного обліку в закладі вищої освіти створює гостру потребу в розробці модуля, який би автоматизував процес збору, очищення та консолідації даних про відвідуваність.

1.2 Огляд існуючих рішень інтеграції платформ відеоконференцзв’язку з освітніми середовищами

Проблема інтеграції платформ відеоконференцзв’язку в освітній процес досліджується в контексті побудови систем управління навчанням (англ. *Learning Management System*, скор. LMS) починаючи з масового переходу ЗВО до дистанційного навчання [1, 9] . Поширені LMS-платформи пропонують власні модулі або плагіни для роботи із Zoom: Moodle [10, 11, 12] , Canvas та Blackboard . Переважно ці рішення базуються на стандарті інтероперабельності навчальних інструментів (англ. *Learning Tools Interoperability*, скор. LTI), що дозволяє

забезпечити базову синхронізацію: створення конференцій безпосередньо з інтерфейсу курсу, автоматичне додавання подій у календар та передачу списку учасників [13]. Однак ці плагіни не інтегровані напряму з внутрішніми інформаційними системами закладів вищої освіти: навіть за умови перегляду даних про відвідуваність у Zoom неможливо автоматично відмітити присутність здобувача освіти в електронному журналі університетської системи. Тому існуючі плагіни варто розглядати лише як приклади базових інтеграцій, а не як завершені рішення для автоматизованого обліку. Крім того, вони переважно орієнтовані на стабільні канали зв'язку та не враховують специфіку обробки «фрагментованих» даних, що стає критичним для українських закладів вищої освіти у сучасних умовах.

У межах розвитку віртуального освітнього середовища KSU24 вже здійснювалися спроби автоматизації взаємодії із сервісом Zoom. Єдиним спеціалізованим попереднім рішенням у межах KSU24 став мікросервіс ConferenceAPI . Цей сервіс виступав безпечним проксі-шаром між KSU24 та Zoom API, забезпечуючи непряму взаємодію з хмарною платформою. На початкових етапах впровадження ConferenceAPI дозволив вирішити першочергові завдання: створення конференцій для занять згідно з розкладом та централізоване управління посиланнями для входу.

Проте детальний технічний огляд даного рішення [14] виявив низку суттєвих архітектурних обмежень, які перешкоджають його подальшій масштабованості та глибокій інтеграції:

- **Технологічна неоднорідність (*Heterogeneous technology environment*):** Основне ядро системи KSU24 побудоване на мові програмування Python із використанням фреймворку Django. Натомість мікросервіс ConferenceAPI був реалізований на стеку технологій .NET (C#). Підтримка двох різних екосистем потребує розширення штату розробників з різними компетенціями, ускладнює процес розгортання та

моніторингу системи. Після міграції інфраструктури закладу сервіс фактично припинив функціонування через відсутність у команді відповідних компетенцій для підтримки .NET-середовища.

- **Інфраструктурна надмірність:** Використання зовнішнього мікросервісу вимагає виділення окремих серверних потужностей, що в умовах обмежених ресурсів закладу вищої освіти є нераціональним. Кожен запит від KSU24 до Zoom проходить через додаткову ланку, що створює мережеві затримки та збільшує кількість точок відмови.
- **Відсутність автоматизації обліку:** ConferenceAPI функціонував за принципом проксі-шлюзу: він лише перенаправляв запити від KSU24 до Zoom API, не виконуючи жодної аналітичної обробки. Усі виклики все одно ініціювалися з боку KSU24, а сервіс не автоматизував жоден процес обліку відвідуваності. Зокрема, він не вирішував задачу консолідації фрагментованих сесій, описану в підрозділі 1.1, залишаючи дані у необробленому вигляді.

Таким чином, хоча ConferenceAPI виконав роль перехідного етапу в цифровізації [14] , визначені у підрозділі 1.1 вимоги до точності обліку відвідуваності та надійності системи [13] зумовлюють необхідність розробки нового модуля. Такий модуль має бути інтегрований безпосередньо в екосистему KSU24 (на базі Python/Django), що дозволить уникнути інфраструктурних втрат та реалізувати складну алгоритмічну логіку обробки даних на рівні ядра системи.

1.3 Порівняльний аналіз методів збору аналітичних даних із хмарних сервісів відеоконференцзв’язку

Надійність модуля автоматизованого обліку навчальної активності суттєво визначається обраним способом отримання первинних даних із хмарної інфраструктури Zoom. Станом на 2025 рік платформа Zoom надає два принципово різних механізми інтеграції [14] : подієву модель

реального часу (Webhooks) та асинхронну модель запитів до архівних даних (Past Meeting API).

Перший підхід (Webhooks) базується на механізмі зворотних викликів (HTTP POST запити від серверів Zoom до сервера закладу вищої освіти), які ініціюються автоматично у момент настання певної події. У контексті обліку відвідуваності це події «participantjoined» (вхід учасника) та «participantleft» (вихід учасника). Попри перевагу у швидкості отримання даних, цей метод має критичні недоліки для освітнього середовища з нестабільними каналами зв'язку:

- **Надмірне навантаження від масових подій:** У випадках масових розривів зв'язку, що є поширеним явищем під час нестабільного енергопостачання чи мережевих збоїв [2] , система генерує шквал однотипних повідомлень про вхід та вихід користувачів. Це створює пікове навантаження на серверну частину системи KSU24, що може призвести до її тимчасової деградації.
- **Ризик втрати цілісності даних:** Модель Webhooks є ефемерною. Якщо у момент відправлення події сервер закладу вищої освіти був недоступний (наприклад, через технічні роботи або збій на лінії), дані про конкретний вхід чи вихід студента будуть безповоротно втрачені. Це робить неможливим побудову достовірної звітності, оскільки відсутність лише однієї події виходу порушує всю логіку розрахунку часу присутності.
- **Складність верифікації:** Дані, що надходять через Webhooks, потребують миттєвої обробки та збереження стану в реальному часі, що ускладнює архітектуру системи та робить її вразливою до помилок синхронізації.

Альтернативним підходом є використання Past Meeting API (Reports API) [14] . Ця модель передбачає асинхронний збір консолідованих даних про завершені конференції. Замість відстеження

окремих транзакцій у реальному часі, система робить один стабільний запит після фактичного завершення навчального заняття.

Для кращого сприйняття та порівняння архітектурних особливостей обох технологій, їх комплексний аналіз за визначеними критеріями наведено у табл. 1.1.

Таблиця 1.1 – Порівняльний аналіз методів збору даних із платформи Zoom

Критерій порівняння	**Webhooks**	**Past Meeting API**
Швидкість отримання даних	У реальному часі	Після завершення сесії
Стійкість до розривів зв’язку	Низька, ризик втрати подій	Висока, дані архівуються
Серверне навантаження	Пікове під час заняття	Прогнозоване, одиничний запит
Цілісність даних	Фрагментована	Консолідований звіт
Придатність для агрегації	Потребує синхронізації станів	Готовий масив для обробки

Обґрунтування вибору моделі Past Meeting API для даного дослідження базується на таких факторах:

- **Гарантована цілісність:** API повертає повний архівний звіт про всіх учасників та всі їхні підключення протягом сесії. Це виключає проблему втрачених подій та забезпечує високу достовірність первинних даних.
- **Оптимізація обчислювальних ресурсів:** Асинхронний запит дозволяє змістити навантаження на сервер у періоди низької активності, не створюючи пікових запитів під час проведення самих пар.
- **Придатність для алгоритмічної обробки:** Оскільки формування звітності у закладі вищої освіти відбувається постфактум, найбільш логічним є отримання повної та цілісної картини сесії одним масивом. Це дозволяє ефективно застосувати алгоритми агрегації розірваних часових інтервалів, нівелюючи вплив мікророзривів зв’язку та технічних збоїв.

Таким чином, для розробки модуля у межах системи KSU24 обрано асинхронну модель взаємодії через Past Meeting API. Це дозволить забезпечити стабільність системи за умов нестабільного зв’язку та створить надійний фундамент для подальшої валідації фактичної присутності здобувачів освіти.

1.4 Обґрунтування вибору архітектурного підходу для інтеграційного модуля

Розробка модуля, який інтегрується зі стороннім хмарним сервісом та водночас є частиною існуючої платформи, висуває суворі вимоги до архітектурної гнучкості. Модуль має бути незалежним від конкретного провайдера відеозв’язку (щоб забезпечити можливість заміни Zoom на альтернативний сервіс) і при цьому не створювати жорстких зв’язків з внутрішньою реалізацією навчальної платформи. Аналіз попереднього рішення (підрозділ 1.2) наочно продемонстрував, як відсутність архітектурної ізоляції призводить до технологічної залежності та інфраструктурних проблем.

Серед архітектурних підходів до побудови модульних систем, описаних у науково-технічній літературі, виділяють такі основні. Класична багатошарова архітектура (*Layered Architecture*) передбачає поділ на шари презентації, бізнес-логіки та доступу до даних, однак зберігає залежність бізнес-логіки від інфраструктурного шару [15]. Цибулева архітектура (*Onion Architecture*) [16] вирішує цю проблему інверсією залежностей, розташовуючи доменне ядро у центрі та спрямовуючи всі залежності назовні. Патерн «Порти та адаптери» (*Ports and Adapters [17]*) розвиває цей підхід далі, формалізуючи межі системи через абстрактні інтерфейси (порти) та їх конкретні реалізації (адаптери).

Для даного дослідження обрано поєднання принципів предметно-орієнтованого проєктування (*Domain-Driven Design, DDD*) [18] та патерну «Порти та адаптери». Такий підхід передбачає поділ програмної

системи на концентричні шари, де у центрі знаходиться ядро бізнес-логіки (домен), яке є повністю незалежним від зовнішніх технологій, фреймворків чи баз даних. Зв’язок ядра із зовнішнім світом відбувається через абстракції (порти), а конкретна технічна реалізація (робота з Zoom API, запис у базу даних, вебінтерфейс) виноситься в окремий шар адаптерів. [21]

Ключовими перевагами обраного підходу для розробки інтеграційного модуля є:

- **Ізоляція доменної логіки:** алгоритми консолідації фрагментованих сесій та ідентифікації користувачів залишаються незалежними від того, як саме Zoom віддає дані або в якому форматі вони зберігаються в базі даних.
- **Незалежність від провайдера:** у разі змін у Zoom API або необхідності міграції на альтернативний сервіс достатньо замінити лише відповідний адаптер, не торкаючись ядра системи.
- **Захист від зв’язності з платформою:** патерн дозволяє створити «антикорупційний шар» (*Anticorruption Layer*) [18] , що захищає модуль від специфіки внутрішньої реалізації навчальної платформи, забезпечуючи взаємодію виключно через визначені порти.
- **Тестованість:** відсутність жорстких зв’язків з інфраструктурою дозволяє покрити алгоритми автоматичними тестами, що є критичним для гарантування точності звітів, які слугують офіційною підставою для академічного оцінювання [19] .

Важливо зазначити, що обраний підхід не прив’язаний до конкретної мови програмування чи фреймворку і може бути реалізований будь-якими програмними засобами, включно з технологічним стеком KSU24 (Python/Django). Детальна адаптація патерну до конкретних потреб модуля описана у другому розділі.

1.5 Вимоги до модуля автоматизованого обліку відвідуваності

На підставі проведеного аналізу предметної галузі, огляду існуючих рішень та обраних технологічних підходів сформульовано вимоги до модуля автоматизованого обліку відвідуваності.

Функціональні вимоги:

1. Автоматичне завантаження даних про учасників конференції з Zoom API (Past Meeting API) після завершення навчального заняття.

2. Ідентифікація здобувачів освіти за корпоративною електронною адресою з підтримкою SSO-автентифікації; позначення учасників, яких не вдалося зіставити, прапорцем неверифікованих.

3. Злиття фрагментованих сесій одного учасника в єдиний запис з налаштовуваним порогом толерантності (за замовчуванням 60 секунд).

4. Розрахунок відсотку присутності з конфігурованим мінімальним порогом (за замовчуванням 75%).

5. Ізоляція даних у межах одного заняття: одне заняття за розкладом відповідає одній унікальній конференції.

6. Запис результатів аналізу у реляційну базу даних KSU24 та формування звіту-рекомендації для електронного журналу.

Нефункціональні вимоги:

- **Надійність:** коректна робота за умов нестабільного мережевого зв’язку; асинхронна модель виключає безповоротну втрату подій, характерну для Webhook-підходу.
- **Розширюваність:** незалежність від конкретного провайдера відеоконференцзв’язку через рівень абстракції (порти/адаптери), що дозволяє замінити Zoom на інший сервіс без зміни доменної логіки.
- **Технологічна сумісність:** реалізація у вигляді Django-модуля в межах існуючого стеку KSU24 (Python/Django/PostgreSQL) без введення нових технологічних залежностей.

- **Тестованість:** ізоляція бізнес-логіки від інфраструктури для покриття алгоритмів автоматичними тестами, що є критичним для гарантування точності звітів.
- **Конфігурованість:** параметри обліку (поріг толерантності, мінімальний відсоток присутності) задаються на рівні конкретного ресурсу (заняття, дисципліна, особа).

Висновки до розділу 1

Проведений аналіз предметної галузі та існуючих технологічних підходів дозволив сформулювати концептуальні засади розробки системи автоматизованого обліку навчальної активності. Встановлено, що ключовою проблемою дистанційної освіти є ізольованість аналітичних даних у хмарних сервісах та складність їх валідації через нестабільний зв’язок учасників. Дослідження попереднього досвіду (зокрема мікросервісу ConferenceAPI) довело недоцільність використання різнорідних технологічних стеків (.NET vs Python) та зовнішніх транзитних шлюзів, що створюють інфраструктурну надмірність.

На основі порівняльного аналізу методів отримання даних (за критеріями швидкості, стійкості до розривів, навантаження на сервер та цілісності масивів) було обґрунтовано відмову від синхронних подієвих моделей (Webhooks) на користь асинхронного збору архівних даних через Past Meeting API. Це рішення, у поєднанні із застосуванням патерну «Порти та адаптери», дозволить розробити нативний модуль, здатний вирішувати проблему дискретності сесій та ідентифікації користувачів. На основі проведеного аналізу сформульовано функціональні та нефункціональні вимоги до модуля (підрозділ 1.5), які визначають межі системи та слугують основою для його проєктування у другому розділі.

РОЗДІЛ 2
ПРОЄКТУВАННЯ СИСТЕМИ АВТОМАТИЗОВАНОГО ОБЛІКУ НАВЧАЛЬНОЇ АКТИВНОСТІ

Передмова до розділу. На основі проведеного у першому розділі аналізу та сформульованих вимог (підрозділ 1.5), у даному розділі представлено повне проєктування системи. Відповідно до підходу «від загального до часткового», розділ починається з опису загальної архітектурної моделі модуля (підрозділ 2.1), після чого детально розглядаються алгоритми ідентифікації (підрозділ 2.2) та консолідації сесій (підрозділ 2.3) як ключові проєктні рішення. Завершують розділ проєкт реляційної структури бази даних (підрозділ 2.4) та UML-діаграми, що фіксують поведінку системи (підрозділ 2.5).

2.1 Архітектурне проєктування модуля

Відповідно до вимог, сформульованих у підрозділі 1.5, та проблем, описаних у підрозділі 1.1, модуль потребує архітектурної ізоляції доменної логіки від зовнішніх систем, можливості заміни провайдера відеозв’язку та чіткого поділу відповідальностей. Як було обґрунтовано у підрозділі 1.4, для реалізації цих вимог обрано підхід на основі патерну «Порти та адаптери» [17] з елементами предметно-орієнтованого проєктування [18].

Слід зазначити, що для модуля обрано адаптований підхід: вузькоспеціалізовані доменні сервіси (*AttendanceAnalyzer* для консолідації сесій та *MeetingConfigResolver* для каскадного пошуку конфігурації) реалізують лише стабільні обчислювальні алгоритми без зовнішніх залежностей. Всю оркестрацію між доменними сервісами, портами та адаптерами виконують прикладні сценарії (юз-кейси): кожен юз-кейс інкапсулює один конкретний бізнес-процес, отримує запит від

зовнішнього рівня, координує виклики до портів і повертає результат. Такий поділ забезпечує чіткі межі між «стабільними алгоритмами» (доменні сервіси) та «оркестрацією» (юз-кейси), зберігаючи при цьому незалежність ядра від інфраструктури.

Доменне ядро модуля містить сутності, що моделюють предметну область обліку відвідуваності. Перелік основних доменних сутностей та їх призначення наведено у табл. 2.1.

Таблиця 2.1 – Доменні сутності модуля автоматизованого обліку відвідуваності

Сутність	**Призначення**
Meeting	Конференція: зв’язок із заняттям, зовнішній ідентифікатор провайдера, параметри доступу, організатор
ParticipantSession	Окремий часовий інтервал підключення учасника (час входу, час виходу)
MergedSession	Результат консолідації кількох суміжних інтервалів в один неперервний проміжок
UserAttendanceSummary	Підсумковий запис про присутність здобувача освіти: загальний час, прапорець верифікації
MeetingAttendanceAnalysis	Агрегований результат аналізу конференції: тривалість та перелік підсумків по учасниках
MeetingConfig	Конфігурація: мінімальний відсоток присутності, налаштування провайдера відеозв’язку

На прикладному рівні визначено набір юз-кейсів, кожен з яких інкапсулює один бізнес-процес: створення конференції для заняття (*UpsertMeetingForLesson*), отримання та аналіз звіту про відвідуваність (*GetMeetingAttendanceUseCase*), пошук конференцій за фільтрами (*FindMeetingsUseCase*), управління конфігурацією (*UpsertMeetingConfigUseCase*, *GetMeetingConfigUseCase*). Доменний сервіс *MeetingConfigResolver* реалізує послідовну перевірку наявності налаштувань на рівні конкретного заняття, потім на рівні журналу дисципліни, далі на рівні персональних налаштувань науково-педагогічного працівника, і лише за відсутності будь-яких збережених конфігурацій застосовує значення за замовчуванням.

Взаємодія доменного ядра та юз-кейсів із зовнішнім світом відбувається виключно через порти, тобто абстрактні інтерфейси, що визначають контракт без прив'язки до реалізації. У модулі визначено шість портів:

- *IMeetingProviderPort*: інтерфейс взаємодії з провайдером відеоконференцій (створення конференції, отримання архівних даних про учасників);
- *IMeetingRepository*: інтерфейс персистентного зберігання та пошуку об'єктів конференцій;
- *ILearningPlatformPort*: інтерфейс взаємодії з навчальною платформою (отримання даних про заняття, зіставлення електронних адрес із профілями здобувачів освіти);
- *IMeetingAttendanceRecordRepository*: інтерфейс зберігання результатів аналізу відвідуваності;
- *IMeetingConfigRepository*: інтерфейс зберігання та отримання конфігурацій;
- *IMeetingInviteeRepository*: інтерфейс зберігання та пошуку списків запрошених учасників конференцій.

Конкретна реалізація кожного порту винесена в шар адаптерів. Вхідним адаптером виступає набір HTTP-ендпоінтів (REST API), через які інтерфейс науково-педагогічного працівника ініціює операції з модулем. Вихідні адаптери включають: адаптер Zoom API (для створення конференцій та отримання архівних звітів через HTTP-клієнт), адаптер навчальної платформи KSU24 (для доступу до даних про заняття та профілі користувачів через Django ORM) та набір репозиторіїв для персистентного зберігання даних у реляційній базі. Зв'язування портів з адаптерами здійснюється через контейнер впровадження залежностей (*Dependency Injection container*) [18] , що дозволяє замінювати адаптери без модифікації доменного коду.

Компонентну діаграму архітектури модуля, що ілюструє описані шари та зв’язки між ними, наведено на рис. 2.1.

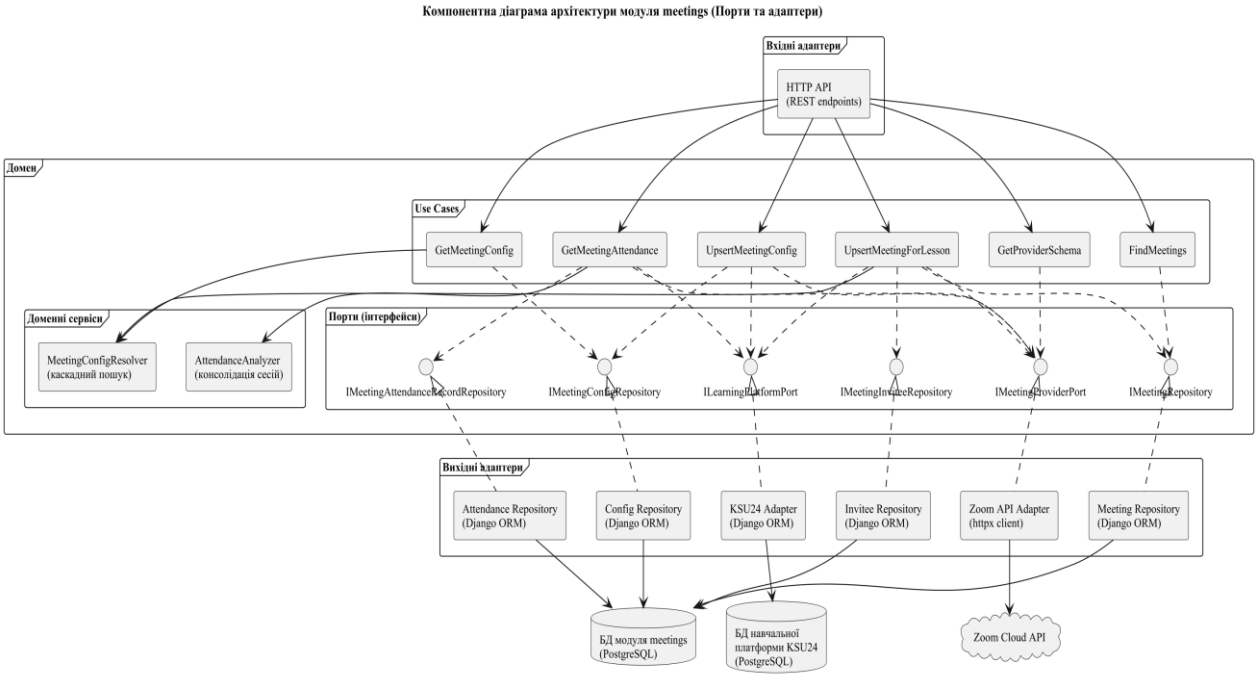

Рисунок 2.1 – Компонентна діаграма архітектури модуля meetings

2.2 Проєктування механізмів ідентифікації учасників конференцій

Однією з ключових проблем, описаних у підрозділі 1.1, є неструктурованість ідентичностей учасників конференцій. Здобувачі освіти можуть підключатися до Zoom під псевдонімами, технічними назвами пристроїв або скороченими іменами, що унеможливлює пряме зіставлення за текстовим ім’ям. Для вирішення цієї проблеми у модулі реалізовано стратегію ідентифікації на основі електронної адреси з використанням механізму єдиного входу (*Single Sign-On, SSO*) [13] .

Обрана стратегія базується на тому, що при використанні SSO-автентифікації в Zoom кожен учасник конференції автоматично ідентифікується за корпоративною електронною адресою, яка є єдиним джерелом істини (*Single Source of Truth, SSOT*) для встановлення особи [13] . Це дозволяє обійти проблему довільних псевдонімів, оскільки електронна адреса прив’язана до облікового запису закладу вищої освіти

і не може бути змінена учасником під час сесії. Такий підхід також мінімізує ризики несанкціонованого доступу до конференцій [20] .

Процес ідентифікації реалізований як двоетапний алгоритм. На першому етапі система отримує з архівного звіту Zoom перелік електронних адрес усіх учасників конференції. При цьому виконується нормалізація адрес: приведення до нижнього регістру та видалення зайвих пробілів. Це усуває розбіжності, спричинені різним форматуванням адрес у різних клієнтських додатках Zoom.

На другому етапі нормалізовані електронні адреси передаються через порт навчальної платформи (*ILearningPlatformPort*) до функції зіставлення, яка виконує пошук відповідних профілів здобувачів освіти у базі даних KSU24. Для кожної адреси, знайденої в системі, встановлюється зв’язок із внутрішнім ідентифікатором особи (*personid*). Адреси, які не вдалося зіставити з жодним профілем (наприклад, гостьові підключення або зовнішні учасники), позначаються спеціальним прапорцем *unverified*, що дозволяє науково-педагогічному працівнику ідентифікувати їх вручну.

Додатковим елементом алгоритму є автоматична фільтрація організатора конференції. Оскільки науково-педагогічний працівник, який проводить заняття, є організатором конференції у Zoom, його присутність у звіті про відвідуваність здобувачів освіти є нерелевантною. Модуль автоматично виключає запис організатора зі списку учасників на основі збереженого ідентифікатора *organizerpersonid*, що усуває необхідність ручного коригування звіту.

2.3 Математична модель та логіка консолідації сесій

Важливим архітектурним обмеженням модуля є принцип «одна подія розкладу – одна віртуальна зустріч». Для кожного навчального заняття за розкладом створюється окрема конференція з унікальним посиланням для підключення. Це забезпечує чітку ізоляцію даних:

неможливо гарантувати, що заняття відбулося у точно заплановані терміни, а перевикористання посилання могло б призвести до змішування даних різних пар. Таке обмеження дещо ускладнює бізнес-логіку, проте значно підвищує достовірність та безпеку зібраних даних.

Центральним алгоритмічним компонентом модуля є механізм консолідації фрагментованих сесій. Як було описано у підрозділі 1.1, в умовах нестабільного інтернет-з'єднання один здобувач освіти протягом одного заняття може генерувати значну кількість окремих записів про підключення та відключення. Без алгоритмічної обробки ці фрагменти неможливо використати для об'єктивної оцінки тривалості присутності.

Для формалізації задачі введемо математичну модель агрегації фрагментованих сесій із застосуванням поняття «толерантності до мережевих збоїв», що доповнює класичні евристичні моделі реконструкції сесій [7] . Нехай хронологічно впорядкована множина часових інтервалів присутності конкретного авторизованого користувача на занятті має вигляд:

$$S = \{(t_{in,1}, t_{out,1}), (t_{in,2}, t_{out,2}), \dots, (t_{in,n}, t_{out,n})\}$$

де $t_{in,i}$ позначає час входу, $t_{out,i}$ позначає час виходу для i -го підключення, n позначає загальну кількість дискретних сесій учасника.

Алгоритм ітеративно обчислює різницю між часом виходу з попередньої сесії $t_{out,i}$ та часом входу в наступну $t_{in,i+1}$. Якщо ця різниця не перевищує наперед встановленого порогу толерантності Δt_{tol}, два суміжні інтервали автоматично об'єднуються в один неперервний проміжок:

$$(t_{in,i}, t_{out,i}) \cup (t_{in,i+1}, t_{out,i+1}) \rightarrow (t_{in,i}, \max(t_{out,i}, t_{out,i+1}))$$

за умови:

$$t_{in,i+1} - t_{out,i} \leq \Delta t_{tol}$$

Для цілей дослідження обрано значення порогу $\Delta t_{tol} = 60$ секунд, що відповідає типовій тривалості мікророзриву з'єднання за характеристиками нестабільних телекомунікаційних мереж [2, 7].

Розроблений алгоритм консолідації складається з двох послідовних фаз. На першій фазі виконується групування всіх записів про підключення (*ParticipantSession*) за електронною адресою учасника.

На другій фазі для кожної групи виконується алгоритм злиття часових інтервалів. Послідовність операцій є наступною:

1. Сортування сесій у хронологічному порядку за часом входу t_{in}.
2. Ініціалізація першого консолідованого інтервалу значеннями першої сесії.
3. Для кожної наступної сесії обчислюється розрив між поточним часом виходу та часом входу нової сесії.
4. Якщо розрив не перевищує порогу Δt_{tol} , сесія вважається продовженням попередньої, і поточний інтервал розширюється відповідно до формули (2).
5. Якщо розрив перевищує порогове значення, поточний консолідований інтервал зберігається як завершений, і розпочинається новий інтервал.
6. Після обробки всіх сесій зберігається останній консолідований інтервал.

Підсумковий час присутності здобувача освіти обчислюється як сума тривалостей усіх консолідованих інтервалів. Це значення (*totalseconds*) зберігається у записі *UserAttendanceSummary* разом з ідентифікатором особи та прапорцем верифікації.

Значення порогу Δt_{tol} є параметризованим: науково-педагогічний працівник може змінити його через параметр запиту при отриманні звіту, що дозволяє адаптувати алгоритм до конкретних умов проведення заняття. Додатковим конфігураційним параметром є мінімальний відсоток присутності, який за замовчуванням становить 75%. Цей поріг

визначає, чи вважається здобувач освіти присутнім на занятті, та використовується при формуванні звіту-рекомендації для електронного журналу.

2.4 Проєкт структури бази даних

Для персистентного зберігання даних модуля спроєктовано реляційну схему, що складається з чотирьох взаємопов’язаних таблиць у базі даних PostgreSQL системи KSU24.

Центральною таблицею є *meetingsmeetings*, що зберігає інформацію про конференції. Кожен запис містить унікальний внутрішній ідентифікатор (UUID), зовнішній ідентифікатор провайдера (*externalid*), тип та ідентифікатор пов’язаного ресурсу навчальної платформи (заняття), ідентифікатор організатора, URL-адреси для підключення та управління, пароль, тему конференції та час початку. Унікальне обмеження на пару полів (*resourcetype*, *resourceid*) гарантує, що для кожного заняття існує не більше однієї конференції.

Таблиця *meetingsconfigurations* зберігає конфігурації модуля на різних рівнях ієрархії. Кожен запис прив’язаний до ресурсу певного типу (заняття, журнал або персона) та містить мінімальний відсоток присутності та JSON-об’єкт з налаштуваннями провайдера (увімкнення зали очікування, вимкнення мікрофонів при вході тощо). Використання JSON-поля для налаштувань провайдера забезпечує гнучкість: при зміні набору параметрів Zoom не потрібно модифікувати структуру бази даних.

Таблиця *meetingsattendancerecords* зберігає результати аналізу відвідуваності. Кожен запис пов’язаний із конференцією через зовнішній ключ та містить електронну адресу учасника, зіставлений ідентифікатор особи (або порожнє значення для неверифікованих учасників), загальний час присутності у секундах та прапорець

верифікації. Унікальне обмеження на пару (*meeting*, *useremail*) запобігає дублюванню записів.

Таблиця *meetingsinvitees* фіксує список запрошених учасників конференції. Вона пов’язує конференцію з ідентифікаторами осіб, які мали бути присутніми на занятті згідно зі списком групи. Ця інформація використовується для визначення відсутніх здобувачів освіти, що не підключилися до конференції.

Усі таблиці використовують UUID як первинні ключі та містять автоматичні мітки часу створення та оновлення для забезпечення аудиту змін.

ER-діаграму реляційної структури бази даних модуля, що ілюструє описані таблиці, їх поля та зв’язки між ними, наведено на рис. 2.2.

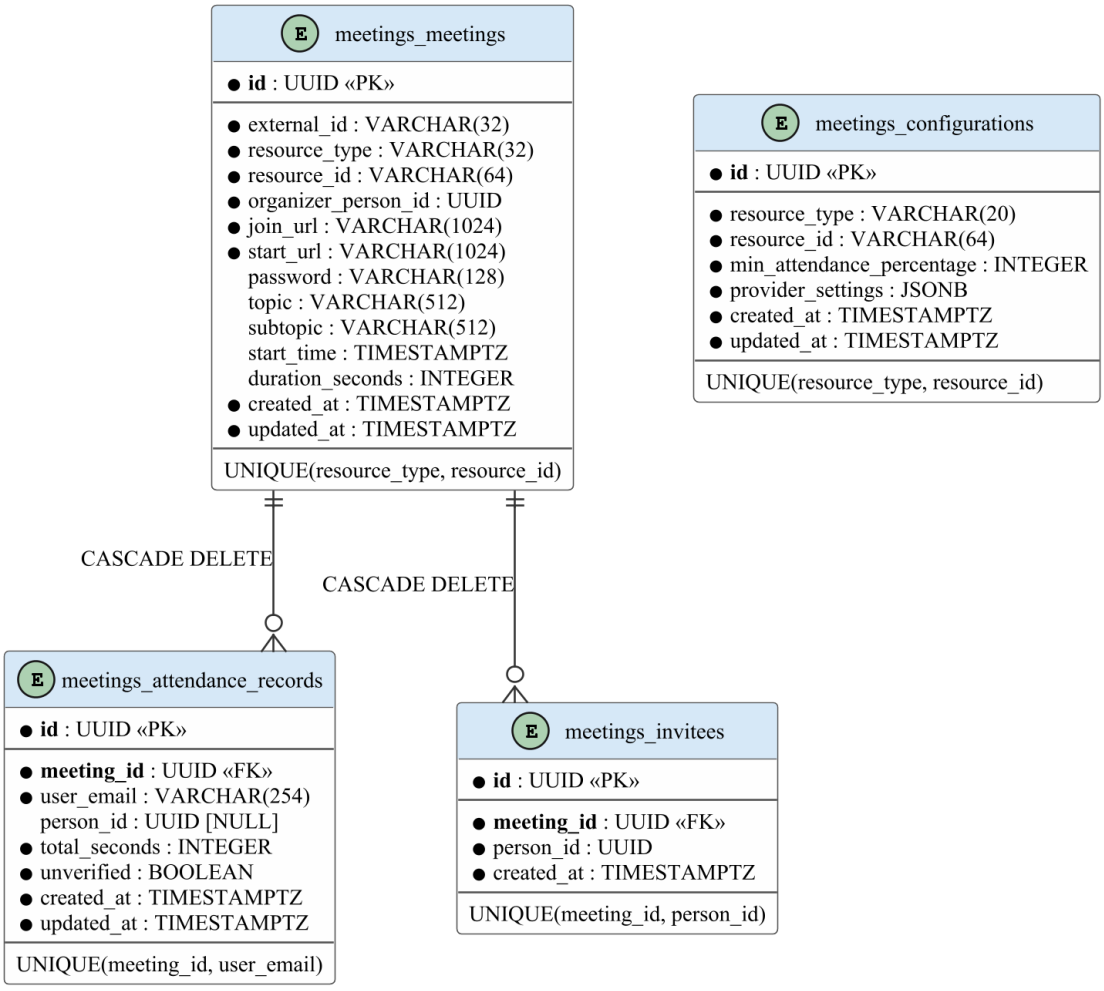

Рисунок 2.2 – ER-діаграма реляційної структури бази даних модуля

2.5 UML-моделювання поведінки системи

Для формалізації сценаріїв взаємодії між акторами та компонентами системи розроблено UML-діаграми двох типів: діаграму прецедентів (*Use Case Diagram*) та діаграму послідовностей (*Sequence Diagram*).

Діаграма прецедентів відображає зовнішніх акторів системи (Науково-педагогічний працівник, Адміністратор ЗВО) та їх взаємодію з функціональними можливостями модуля: створення конференції для заняття, отримання звіту про відвідуваність (з аналізом даних Zoom), налаштування конфігурації, перегляд списку конференцій.

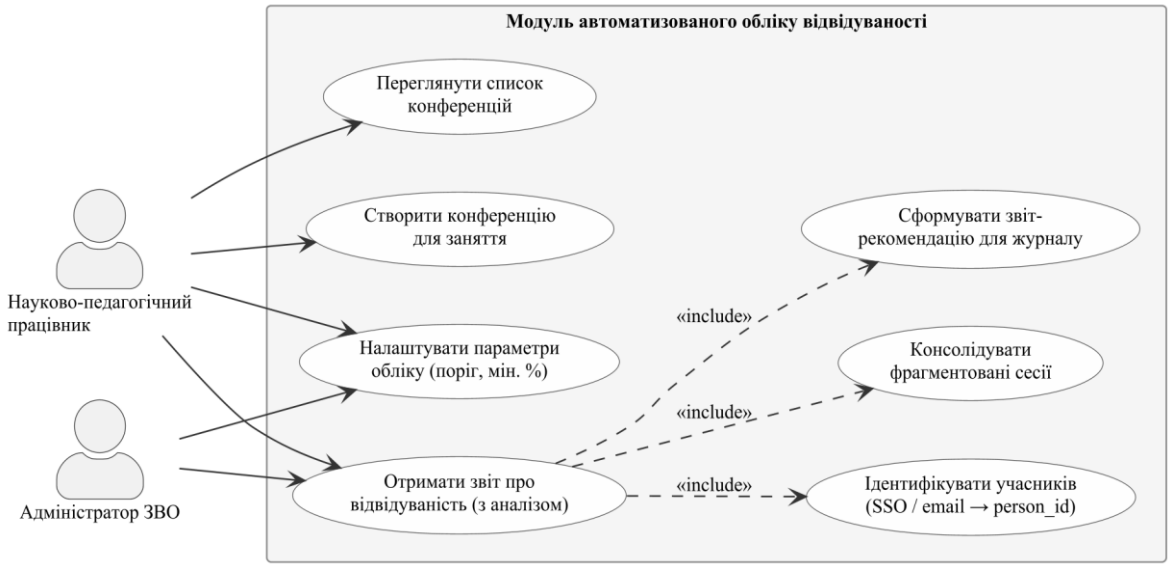

Рисунок 2.3 – Діаграма прецедентів модуля автоматизованого обліку відвідуваності

Діаграма послідовностей ілюструє потік виконання основного сценарію – отримання звіту про відвідуваність за HTTP-запитом, що включає перевірку кешу, завантаження даних з Zoom API та консолідацію сесій. Відображаються взаємодії між: клієнтом KSU24, HTTP-ендпоінтом, юз-кейсом *GetMeetingAttendanceUseCase*, доменним сервісом *AttendanceAnalyzer*, портами *IMeetingProviderPort* та *IMeetingAttendanceRecordRepository*, а також адаптерами Zoom API та KSU24.

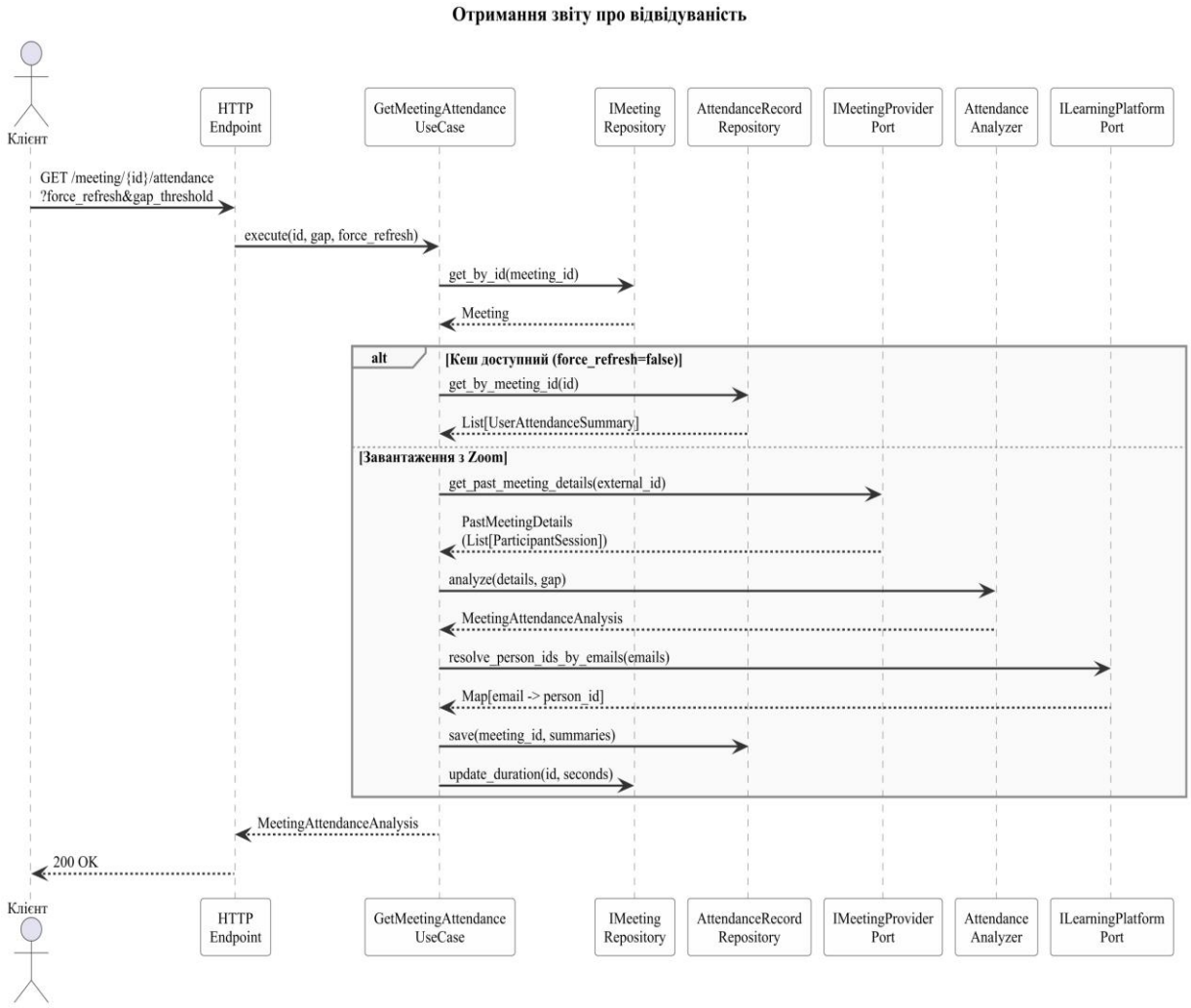

Рисунок 2.4 – Діаграма послідовностей: сценарій обробки завершеного заняття

Висновки до розділу 2

У підрозділах 2.1-2.5 розроблено повне проєктне рішення системи автоматизованого обліку відвідуваності. Спроєктовано архітектуру модуля із застосуванням адаптованого патерну «Порти та адаптери»: стабільні обчислювальні алгоритми зосереджено в доменних сервісах (*AttendanceAnalyzer*, *MeetingConfigResolver*), тоді як оркестрацію виконують прикладні сценарії (юз-кейси). Визначено шість абстрактних інтерфейсів (портів), що забезпечують незалежність доменної логіки від конкретних реалізацій взаємодії із Zoom API, базою даних та навчальною платформою.

Розроблено алгоритм ідентифікації користувачів на основі електронної адреси з підтримкою SSO, що дозволяє автоматично зіставляти учасників конференції з профілями здобувачів освіти. Для випадків неуспішного зіставлення передбачено механізм маркування неверифікованих записів. Обґрунтовано принцип ізоляції «одна подія розкладу – одна конференція», що забезпечує достовірність зібраних даних.

Спроєктовано алгоритм консолідації фрагментованих сесій з параметризованим пороговим значенням розриву (за замовчуванням 60 секунд), що дозволяє нівелювати вплив мікророзривів зв’язку. Розроблено реляційну схему бази даних із чотирьох таблиць для персистентного зберігання конференцій, конфігурацій, результатів аналізу та списків запрошених. UML-діаграми (підрозділ 2.5) формалізують сценарії взаємодії між акторами та компонентами системи.

Отримані результати проєктування створюють необхідний фундамент для переходу до третього розділу, а саме безпосередньої програмної реалізації та тестування модуля.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ АВТОМАТИЗОВАНОГО ОБЛІКУ НАВЧАЛЬНОЇ АКТИВНОСТІ

Передмова до розділу. Третій розділ присвячено безпосередній програмній реалізації системи, спроєктованої у другому розділі. Виклад матеріалу побудовано за принципом «від ядра до периферії»: спочатку описуються доменні сервіси та юз-кейси, що реалізують бізнес-логіку (підрозділ 3.1), потім – REST API як вхідний адаптер для звітності (підрозділ 3.2), далі – інтеграція з Zoom API та базою даних KSU24 через вихідні адаптери (підрозділ 3.3), після чого – автоматизоване тестування та демонстрація інтерфейсу системи (підрозділ 3.4), і завершується розділ апробацією модуля на тестовій конференції (підрозділ 3.5).

3.1 Реалізація доменної логіки та обчислювальних сервісів

Бізнес-логіка модуля зосереджена на рівні юз-кейсів та доменних сервісів. Доменні сервіси інкапсулюють обчислювальні алгоритми без зовнішніх залежностей, тоді як юз-кейси оркеструють взаємодію між сервісами, портами та адаптерами. Усі зовнішні залежності (репозиторії, адаптери провайдера) передаються через конструктор, що забезпечує тестованість без запуску реальної інфраструктури.

Алгоритм консолідації сесій. Центральним доменним сервісом є *AttendanceAnalyzer*. Метод mergesessions() реалізує алгоритм злиття часових інтервалів, описаний у підрозділі 2.3: сесії сортуються за часом входу, після чого ітеративно перевіряється розрив між суміжними інтервалами відносно порогу Δt_{tol}. Повний код методу наведено у лістингу 3.1.

```
@staticmethod
def _merge_sessions(
    raw_sessions: list,
```

```
    gap_threshold_seconds: int,
) -> list[MergedSession]:
    sorted_sessions = sorted(raw_sessions, key=lambda s: s.join_time)
    if not sorted_sessions:
        return []

    merged: list[MergedSession] = []
    current_join = sorted_sessions[0].join_time
    current_leave = sorted_sessions[0].leave_time

    for session in sorted_sessions[1:]:
        gap = (session.join_time - current_leave).total_seconds()
        if gap <= gap_threshold_seconds:
            if session.leave_time > current_leave:
                current_leave = session.leave_time
        else:
            duration = int((current_leave - current_join).total_seconds())
            merged.append(MergedSession(
                join_time=current_join,
                leave_time=current_leave,
                duration_seconds=duration,
            ))
            current_join = session.join_time
            current_leave = session.leave_time

    duration = int((current_leave - current_join).total_seconds())
    merged.append(MergedSession(
        join_time=current_join,
        leave_time=current_leave,
        duration_seconds=duration,
    ))
    return merged
```

Лістинг 3.1 – Метод AttendanceAnalyzer._merge_sessions() – злиття суміжних сесій

Метод analyze() виконує групування всіх записів підключень за email та викликає mergesessions() для кожної групи. Тривалість конференції обчислюється як різниця між часом завершення та початку, отриманими з архівного звіту Zoom. Результат інкапсулюється у доменну сутність *MeetingAttendanceAnalysis*. Код методу наведено у лістингу 3.2.

```
def analyze(
    self,
    details: PastMeetingDetails,
    gap_threshold_seconds: int = DEFAULT_SESSION_MERGE_GAP_SECONDS,
) -> Meeting\allowbreak Attendance\allowbreak Analysis:
    meeting_duration = int(
        (details.end_time - details.start_time).total_seconds()
    )

    sessions_by_email: dict[str, list] = {}
    for session in details.participant_sessions:
        sessions_by_email.setdefault(session.user_email, []).append(session)
```

```
    attendees: list[User\allowbreak Attendance\allowbreak Summary] = []
    for email, raw_sessions in sorted(sessions_by_email.items()):
        merged = self._merge_sessions(raw_sessions, gap_threshold_seconds)
        total = sum(s.duration_seconds for s in merged)
        attendees.append(User\allowbreak Attendance\allowbreak Summary(
            user_email=email,
            total_seconds=total,
        ))

    return Meeting\allowbreak Attendance\allowbreak Analysis(
        meeting_duration_seconds=meeting_duration,
        attendees=attendees,
    )
```

Лістинг 3.2 – Метод AttendanceAnalyzer.analyze() – аналіз відвідуваності

Юз-кейс отримання звіту. Клас *GetMeetingAttendanceUseCase* оркеструє повний сценарій формування звіту: перевірку кешу, завантаження даних з провайдера, аналіз, зіставлення email з personid та збереження результату. Фільтрація організатора конференції виконується на рівні збагачення даних. Код методу execute() наведено у лістингу 3.3.

```
def execute(
    self,
    meeting_id: UUID,
    gap_threshold_seconds: int = DEFAULT_SESSION_MERGE_GAP_SECONDS,
    force_refresh: bool = False,
) -> Meeting\allowbreak Attendance\allowbreak Analysis:

    meeting = self._meeting_repository.get_by_id(meeting_id)
    if meeting is None:
        raise Business\allowbreak Rule\allowbreak Error("meetings.E205 |
Meeting not found")

    if not force_refresh:
        cached =
self._attendance_record_repository.get_by_meeting_id(meeting_id)
        if cached and meeting.duration_seconds is not None:
            return Meeting\allowbreak Attendance\allowbreak Analysis(
                meeting_duration_seconds=meeting.duration_seconds,
                attendees=cached,
            )

    details = self._meeting_provider_adapter.get_past_meeting_details(
        meeting_external_id=meeting.meeting_external_id,
    )
    if details is None:
        raise Business\allowbreak Rule\allowbreak Error("meetings.E204 | Data
not yet available")

    analysis = self._attendance_analyzer.analyze(
        details=details,
```

```
        gap_threshold_seconds=gap_threshold_seconds,
    )

    emails = {a.user_email for a in analysis.attendees}
    email_to_person =
self._learning_platform_adapter.resolve_person_ids_by_emails(emails)

    enriched_attendees = [
        User\allowbreak Attendance\allowbreak Summary(
            person_id=email_to_person.get(a.user_email),
            user_email=a.user_email,
            total_seconds=a.total_seconds,
            unverified=a.user_email not in email_to_person,
        )
        for a in analysis.attendees
        if email_to_person.get(a.user_email) != meeting.organizer_person_id
    ]

    final_analysis = Meeting\allowbreak Attendance\allowbreak Analysis(
        meeting_duration_seconds=analysis.meeting_duration_seconds,
        attendees=enriched_attendees,
    )

    self._attendance_record_repository.save(meeting_id, enriched_attendees)
    self._meeting_repository.update_duration(meeting_id,
analysis.meeting_duration_seconds)
    return final_analysis
```

Лістинг 3.3 – Метод GetMeetingAttendanceUseCase.execute() – оркестрація сценарію

Сервіс конфігурації. Доменний сервіс *MeetingConfigResolver* реалізує каскадну стратегію пошуку налаштувань: для кожного рівня ієрархії (заняття, журнал, особа) здійснюється запит до репозиторію; за відсутності будь-яких налаштувань повертається конфігурація з константами за замовчуванням. Це дозволяє НПП задавати глобальні параметри один раз, переозначуючи їх лише при потребі для окремих занять. Код методу наведено у лістингу 3.4.

```
def resolve(
    self,
    lesson_id: Optional[str],
    gradebook_id: Optional[str],
    person_id: Optional[str],
    default_provider_settings: dict[str, Any],
) -> MeetingConfig:
    if lesson_id:
        config = self._config_repository.get_config(ResourceType.LESSON,
lesson_id)
        if config:
            return config

    if gradebook_id:
        config = self._config_repository.get_config(ResourceType.GRADEBOOK,
```

```
gradebook_id)
        if config:
            return config

    if person_id:
        config = self._config_repository.get_config(ResourceType.PERSON,
person_id)
        if config:
            return config

    return MeetingConfig(
        resource_type=ResourceType.PERSON if person_id else
ResourceType.GRADEBOOK,
        resource_id=person_id or gradebook_id or lesson_id or "",
        min_attendance_percentage=DEFAULT_MIN_ATTENDANCE_PERCENTAGE,
        provider_settings=default_provider_settings,
    )
```

Лістинг 3.4 – Метод MeetingConfigResolver.resolve() – каскадний пошук конфігурації

3.2 Реалізація вхідного адаптера та програмного інтерфейсу

Вхідним адаптером модуля є набір HTTP-ендпоінтів, реалізованих за допомогою бібліотеки django-ninja. Вона надає декларативний інтерфейс для опису REST API безпосередньо над Python-функціями, забезпечує автоматичну валідацію параметрів, серіалізацію відповідей через Pydantic-схеми та генерацію документації у форматі OpenAPI. Авторизація здійснюється через JWT-токен відповідно до рольової моделі KSU24.

Ендпоінт отримання звіту про відвідуваність приймає два необов’язкових параметри: gapthresholdseconds для задання порогу консолідації сесій та forcerefresh для примусового оновлення кешу. Результатом є DTO об’єкт *MeetingAttendanceResponseDto*, що містить тривалість конференції та перелік записів про відвідуваність. Код ендпоінту наведено у лістингу 3.5.

```
@api.get(
    "/meeting/{meeting_id}/attendance",

auth=jwt_auth_handler(permission=MeetingsCapabilities.MeetingAttendanceScope.
read),
    response=Meeting\allowbreak Attendance\allowbreak Response\allowbreak
Dto,
    tags=["Attendance"],
    summary="Отримати аналіз відвідування мітингу",
)
```

```
@log_db_queries
@handle_exceptions
def get_meeting_attendance(
    request,
    meeting_id: UUID = Path(..., description="Ідентифікатор відеозустрічі"),
    gap_threshold_seconds: int = DEFAULT_SESSION_MERGE_GAP_SECONDS,
    force_refresh: bool = False,
):
    container = meetings_container.get_meetings_container()
    analysis = container.get_meeting_attendance.execute(
        meeting_id=meeting_id,
        gap_threshold_seconds=gap_threshold_seconds,
        force_refresh=force_refresh,
    )
    response = Meeting\allowbreak Attendance\allowbreak Response\allowbreak
Dto(
        meeting_duration_seconds=analysis.meeting_duration_seconds,
        attendees=[
            AttendeeDto(
                person_id=a.person_id,
                user_email=a.user_email,
                total_seconds=a.total_seconds,
                unverified=a.unverified,
            )
            for a in analysis.attendees
        ],
    )
    return response
```

Лістинг 3.5 – HTTP-ендпоінт GET /meeting/{meeting_id}/attendance

Структуру об’єкта відповіді визначає DTO *MeetingAttendanceResponseDto*, поля якого наведено у табл. 3.1.

Таблиця 3.1 – Поля об’єкта відповіді MeetingAttendanceResponseDto

Поле	Тип	Опис
meetingdurationseconds	int	Загальна тривалість конференції, с
attendees	list[AttendeeDto]	Перелік записів про учасників
AttendeeDto.personid	UUID | null	Внутрішній ідентифікатор особи в KSU24
AttendeeDto.useremail	string	Email-адреса учасника
AttendeeDto.totalseconds	int	Сумарний час присутності після консолідації, с
AttendeeDto.unverified	bool	Прапорець нерозпізнаного учасника

Інтерфейс перегляду звіту відвідуваності для НПП інтегровано до клієнтської частини KSU24. Сторінка звіту відображає тривалість конференції, перелік учасників із загальним часом присутності у відсотках до тривалості заняття та прапорець верифікованості. Учасники,

для яких не вдалося встановити відповідність у базі, виводяться окремо з позначкою. Зовнішній вигляд сторінки звіту відвідуваності наведено на рис. 3.1.

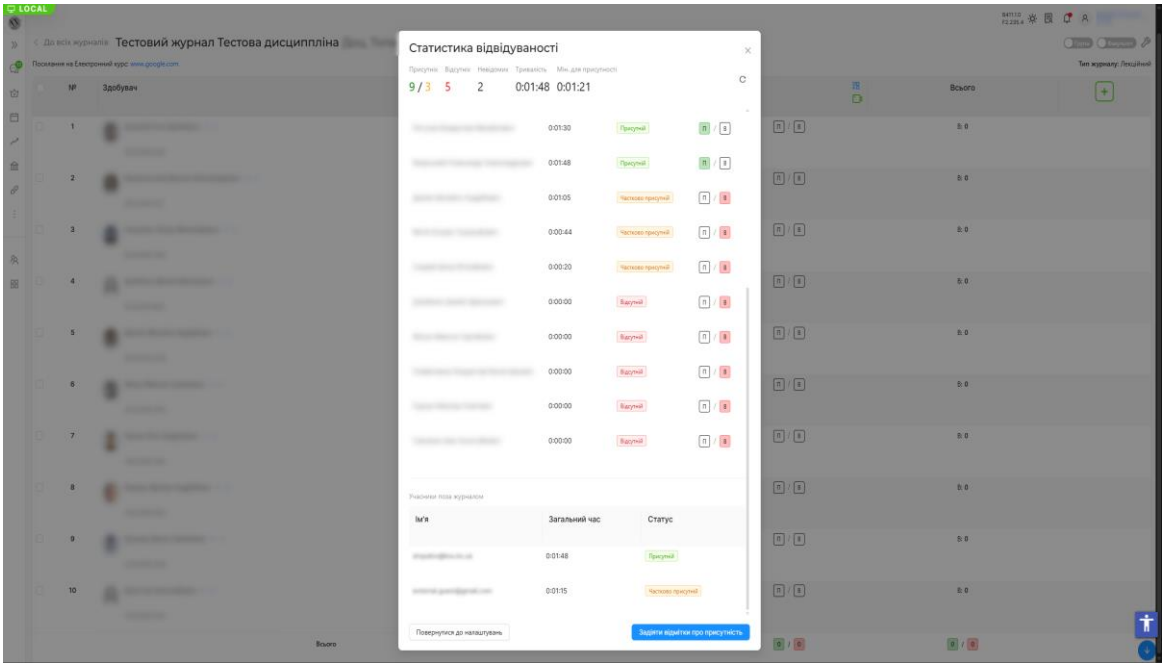

Рисунок 3.1 – Сторінка звіту відвідуваності у клієнтській частині KSU24

На рисунку 3.1 видно, що інтерфейс відображає назву конференції, загальну тривалість заняття та перелік учасників з індивідуальним часом присутності. Для кожного учасника вказується відсоток від загальної тривалості та значення прапорця верифікованості: учасники, чий email не знайдено у базі KSU24 (наприклад, підключились з особистого акаунта), позначаються окремо. Такий формат звіту дозволяє НПП отримати готову рекомендацію щодо виставлення оцінок за відвідуваністю, виконуючи таким чином вимогу 2 підрозділу 1.5. Окрім користувацького інтерфейсу, django-ninja автоматично генерує інтерактивну OpenAPI-документацію для всіх ендпоінтів модуля. Документацію REST API модуля (Swagger UI) наведено на рис. 3.2.

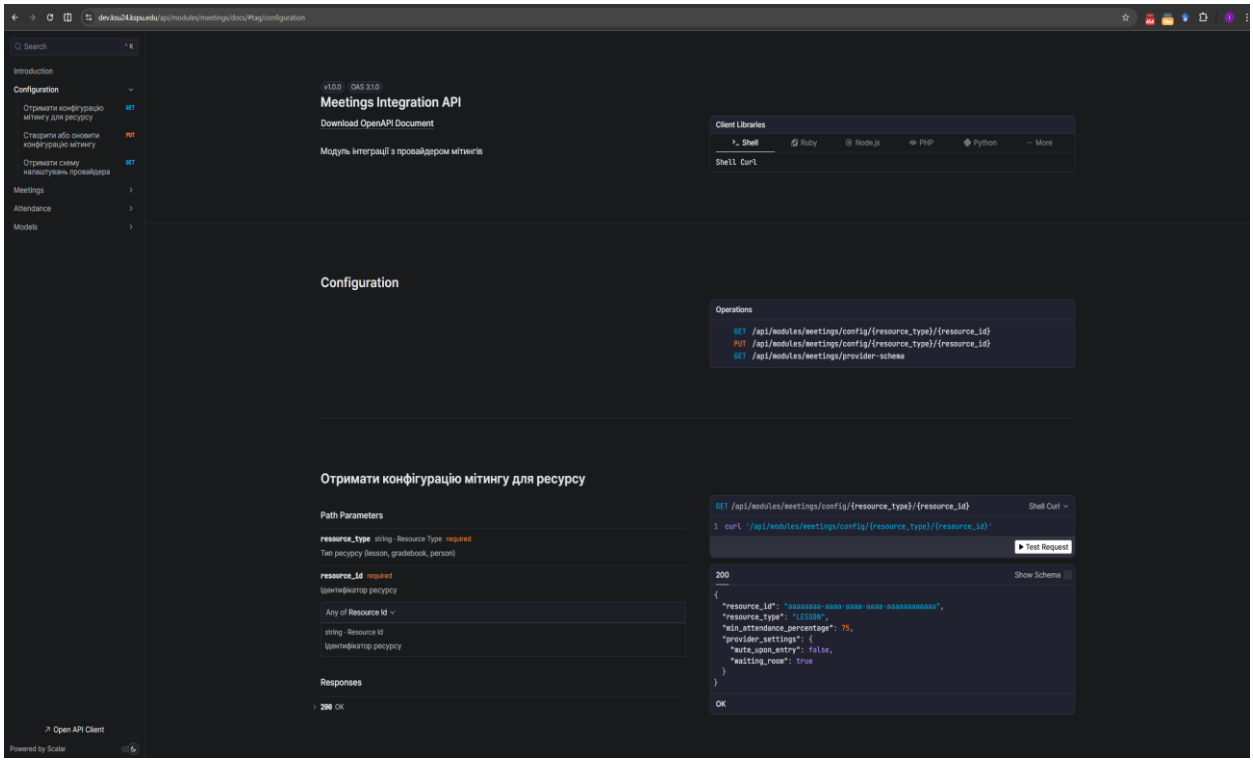

Рисунок 3.2 – OpenAPI-документація (Scalar) REST API модуля meetings

Рисунок 3.2 демонструє автоматично сформовану Scalar-документацію REST API модуля. Scalar генерується на основі OpenAPI-схеми django-ninja і відображає повний перелік ендпоінтів, схеми запитів і відповідей, механізм JWT-авторизації та інтерактивні форми для надсилання тестових запитів безпосередньо зі сторінки документації. Наявність такої документації дозволяє стороннім розробникам інтегруватися з API модуля без необхідності вивчення вихідного коду.

3.3 Реалізація вихідних адаптерів зовнішніх та внутрішніх систем

Вихідні адаптери реалізують порти, визначені в доменному ядрі, та забезпечують фактичну взаємодію з зовнішніми системами. Це дозволяє замінити конкретний адаптер (наприклад, перейти від Zoom до Microsoft Teams) без модифікації будь-якого коду доменної логіки.

Адаптер Zoom API. Клас *ZoomClient* є низькорівневим HTTP-клієнтом на базі бібліотеки httpx. Для отримання OAuth-токена

реалізовано метод getoauthtoken(), який кешує токен в пам’яті процесу до закінчення його терміну дії з урахуванням поправки на мережеві затримки (tokenleewayseconds). Метод getpastmeetingparticipants() реалізує пагінацію через поле nextpagetoken у відповіді Zoom API з розміром сторінки 300 записів. Повний код обох методів наведено у Додатку Б.1. [23]

Для верифікації коректності роботи ланцюжка «конфігураційний адаптер – MeetingConfigResolver – ендпоінт» виконано тестовий запит до API за допомогою інструменту Bruno. Приклад фактичної JSON-відповіді ендпоінта GET /config/lesson/{lessonid}, що повертає каскадно розв’язану конфігурацію мітингу для заняття, наведено на рис. 3.3.

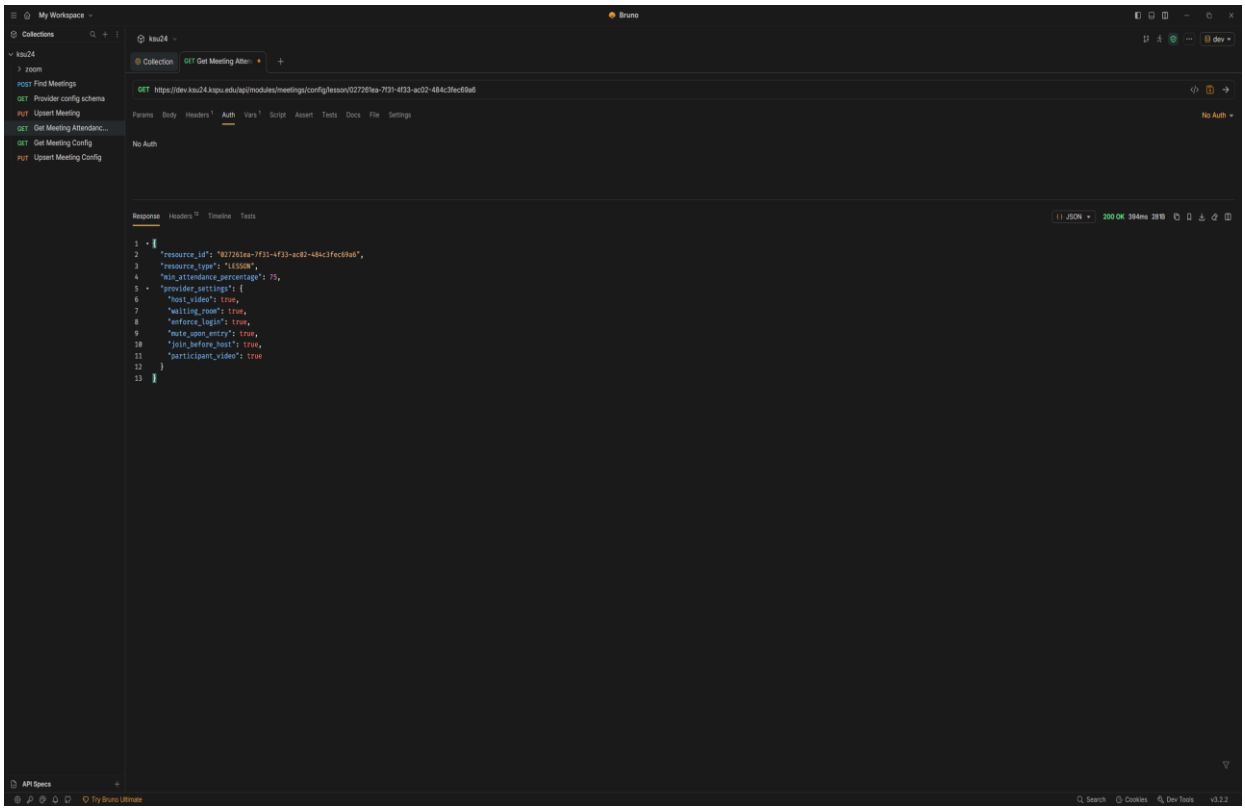

Рисунок 3.3 – Приклад JSON-відповіді ендпоінта GET /config/lesson/{lessonid} (Bruno)

На рис. 3.3 показано структуровану відповідь конфігураційного ендпоінта: поле resourcetype вказує рівень ієрархії, на якому знайдено конфігурацію (заняття, журнал або персона), поле minattendancepercentage містить мінімальний поріг присутності, а

providersettings – JSON-об’єкт з параметрами провайдера відеоконференцзв’язку (зал очікування, мікрофони тощо). Успішне виконання запиту підтверджує коректну роботу ланцюжка: HTTP-ендпоінт → юз-кейс → *MeetingConfigResolver* → *ConfigRepository* → база KSU24.

Адаптер KSU24. Клас *Ksu24Adapter* реалізує порт *ILearningPlatformPort* з використанням Django ORM для доступу до внутрішніх моделей KSU24. Метод resolvepersonidsbyemails() виконує одиничний SQL-запит для зіставлення набору email-адрес з UUID-ідентифікаторами персон. Нормалізація адрес до нижнього регістру запобігає розбіжностям форматування. Повний код методу наведено у Додатку Б.2.

Контейнер впровадження залежностей. Збірка всього об’єктного графа модуля здійснюється функцією getmeetingscontainer(), задекорованою @lrucache(maxsize=1), що забезпечує патерн «синглтон» на рівні процесу. Контейнер є замороженим датакласом (frozen=True), тому його поля доступні лише для читання після ініціалізації. Для ізольованого тестування передбачено функцію resetmeetingscontainerfortests(), що скидає всі кеші. Повний код функції наведено у Додатку Б.3.

3.4 Тестування та демонстрація роботи системи

Для підтвердження коректності бізнес-логіки та поведінки HTTP-інтерфейсу розроблено набір автоматизованих тестів на базі django.test.SimpleTestCase та unittest.mock. Тести покривають всі шість юз-кейсів модуля. Порти замінюються mock-об’єктами, що дозволяє виконувати тести без підключення до реальних баз даних або Zoom API. Перелік тестових файлів наведено у табл. 3.2. [22]

Таблиця 3.2 – Тестові файли модуля meetings

Файл	Об'єкт тестування
testgetmeetingattendance.py	GetMeetingAttendanceUseCase
testfindmeetings.py	FindMeetingsUseCase
testgetmeetingconfig.py	GetMeetingConfigUseCase
testgetproviderschema.py	GetProviderSchemaUseCase
testupsertmeetingconfig.py	UpsertMeetingConfigUseCase
testupsertmeetingforlesson.py	UpsertMeetingForLesson

Тести юз-кейсу *GetMeetingAttendanceUseCase* перевіряють: повернення кешованих даних при forcerefresh=False, коректне завантаження та аналіз даних при кеш-промаху, фільтрацію організатора зі списку учасників, а також генерацію виключення *BusinessRuleError* при відсутності мітингу або недоступності даних Zoom. Тести алгоритму консолідації перевіряють злиття інтервалів при розриві менше порогу та збереження окремих інтервалів при перевищенні порогу.

Результат прогону повного набору автоматизованих тестів модуля наведено на рис. 3.4.

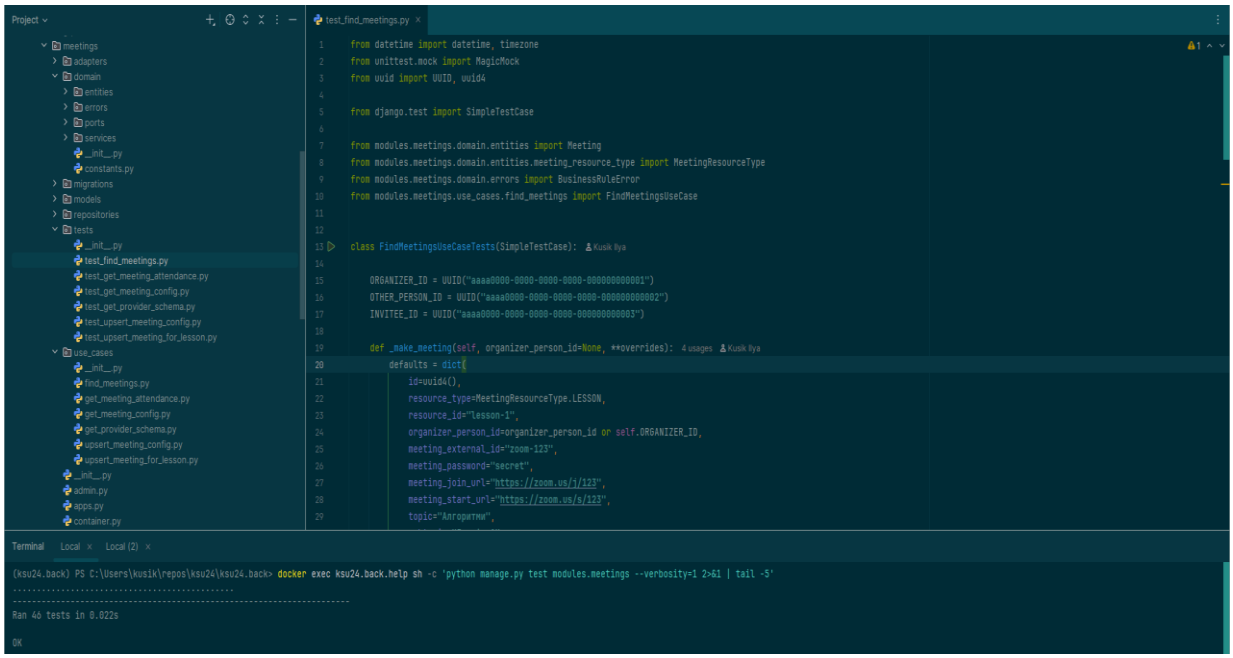

Рисунок 3.4 – Результат виконання автоматизованих тестів модуля meetings

Рисунок 3.4 демонструє вивід команди python manage.py test modules.meetings у термінальному середовищі. На виводі відображається

кількість виконаних тестових випадків, час виконання та підсумковий статус. Успішний прогін усіх тестових файлів (табл. 3.2) без єдиної помилки підтверджує коректність реалізації бізнес-логіки: алгоритмів консолідації сесій, ідентифікації учасників та каскадного пошуку конфігурації. Практичну апробацію системи на реальній конференції описано у підрозділі 3.5.

3.5 Апробація модуля на тестовій конференції

Для підтвердження практичної придатності модуля проведено апробацію на тестовій конференції за участю 4 учасників: одного організатора (науково-педагогічний працівник) та трьох слухачів (здобувачі освіти). Тривалість конференції склала 45 хвилин. Для імітації умов нестабільного зв'язку, характерних для сучасного освітнього середовища [2] , два учасники навмисно здійснювали повторні перепідключення, що призвело до появи у даних Zoom по 4–5 окремих записів сесій для кожного з них.

API-запит GET /meeting/{meetingid}/attendance з параметром за замовчуванням gapthresholdseconds=60 було виконано після завершення конференції. Роботу алгоритму mergesessions() проілюстровано на рис. 3.5 у вигляді таблиці «до та після консолідації» для одного з учасників тестової конференції.

До консолідації – 5 записів із журналу Zoom API:

#	Час входу	Час виходу	Тривалість	Розрив до наст.
1	10:00:00	10:08:00	480 с	40 с < 60 с → злиття
2	10:08:40	10:17:30	530 с	40 с < 60 с → злиття
3	10:18:10	10:28:00	590 с	50 с < 60 с → злиття
4	10:28:50	10:35:00	370 с	40 с < 60 с → злиття
5	10:35:40	10:38:00	140 с	—

⇓ *AttendanceAnalyzer._merge_sessions(Δt_tol = 60 с)*

Після консолідації – 1 запис:

#	Час входу	Час виходу	Тривалість	% від конф.
1	10:00:00	10:38:00	2280 с (38 хв)	84%

Рисунок 3.5 – Приклад роботи алгоритму консолідації сесій (Δt = 60 с, тестова конференція 45 хв)

На рис. 3.5 проілюстровано ключовий результат алгоритму: 5 фрагментованих інтервалів одного учасника об’єднано в 1 консолідовану сесію тривалістю 38 хвилин. Це становить 84% від загальної тривалості конференції та перевищує мінімальний поріг присутності 75%, визначений у підрозділі 1.5. Без консолідації (при простому підрахунку кожного інтервалу окремо) зазначений учасник міг би бути помилково класифікований як частково присутній через короткі інтервали записів. Апробація підтвердила, що алгоритм коректно обробляє реальні дані фрагментованих сесій та формує звіт, придатний для використання як рекомендація для НПП щодо виставлення оцінок за відвідуваністю в електронному журналі KSU24.

Висновки до розділу 3

У третьому розділі здійснено програмну реалізацію всіх компонентів системи автоматизованого обліку відвідуваності відповідно до архітектурного проєкту другого розділу. В основному тексті представлено п’ять ключових лістингів коду: алгоритм консолідації фрагментованих сесій (лістинги 3.1–3.2), юз-кейс оркестрації звіту (лістинг 3.3), сервіс каскадного пошуку конфігурації (лістинг 3.4) та REST API ендпоінт (лістинг 3.5). Лістинги адаптерів та DI-контейнера винесено до Додатку Б.

Працездатність системи підтверджена трьома способами. По-перше, п’ятьма ілюстраціями: скріншот інтерфейсу звіту KSU24 (рис. 3.1), OpenAPI-документація (рис. 3.2), приклад JSON-відповіді (рис. 3.3), вивід автоматизованих тестів (рис. 3.4) та результат алгоритму

консолідації на тестовій конференції (рис. 3.5). По-друге, автоматизованими тестами, що покривають усі шість юз-кейсів модуля через mock-порти. По-третє, апробацією на тестовій конференції (підрозділ 3.5), яка підтвердила коректність алгоритму злиття: 5 фрагментованих записів учасника об’єднано в одну сесію тривалістю 84% від конференції, що відповідає вимогам академічного обліку відвідуваності.

ВИСНОВКИ

У кваліфікаційній роботі вирішено науково-прикладне завдання – розроблено програмний модуль автоматизованого обліку відвідуваності здобувачів вищої освіти на платформі KSU24 Херсонського державного університету на основі інтеграції з сервісом відеоконференцзв'язку Zoom. За результатами виконання кожного із семи завдань дослідження отримано такі висновки.

1. Аналіз проблематики дистанційної освіти підтвердив, що ізольованість аналітичних даних у хмарних сервісах та фрагментація сесій через нестабільний зв'язок є системними проблемами, що унеможливлюють ручний облік без значних часових витрат. Встановлено, що записи одного учасника можуть розпадатися на десятки окремих інтервалів підключення, що потребує алгоритмічної обробки.

2. Огляд існуючих рішень – плагінів LTI-сумісних платформ (Moodle, Canvas) та мікросервісу ConferenceAPI – виявив спільне архітектурне обмеження: ці рішення функціонують як транзитні шлюзи та не реалізують консолідацію фрагментованих сесій. Мікросервіс ConferenceAPI додатково створював технологічну неоднорідність (.NET проти Python), що призвело до його фактичного виходу з ладу після зміни інфраструктури.

3. Порівняльний аналіз методів Webhooks та Past Meeting API за критеріями стійкості до розривів зв'язку, цілісності даних та серверного навантаження обґрунтував вибір асинхронної моделі через Past Meeting API як єдиного підходу, що гарантує повноту даних в умовах нестабільної мережі.

4. Спроєктовано архітектуру модуля на основі патерну «Порти та адаптери» з елементами предметно-орієнтованого проєктування. Доменне ядро містить 6 сутностей та 6 абстрактних портів; прикладний

рівень складається з 6 юз-кейсів та 2 доменних сервісів. UML-діаграми (компонентна, ER, прецедентів, послідовностей) формалізують статичну та динамічну структуру системи.

5. Математично формалізовано алгоритм консолідації фрагментованих сесій з параметризованим порогом $\Delta t_{tol} = 60$ секунд та двоетапний алгоритм ідентифікації учасників через SSO-зіставлення email-адрес. Спроєктовано реляційну схему з 4 таблиць (конференції, конфігурації, записи відвідуваності, запрошені) з унікальними обмеженнями на парах ключових полів.

6. Реалізовано програмний модуль мовою Python у фреймворку Django. Адаптер Zoom API забезпечує OAuth-аутентифікацію з кешуванням токена та пагінацію через nextpagetoken; адаптер KSU24 зіставляє email-адреси учасників з UUID-ідентифікаторами осіб через Django ORM. REST API побудовано за допомогою django-ninja з JWT-авторизацією. DI-контейнер збирає об'єктний граф єдиний раз на процес через @lrucache(maxsize=1).

7. Розроблено 6 тестових файлів, що покривають усі юз-кейси модуля з використанням mock-портів без підключення до реальної інфраструктури. Проведено апробацію на тестовій конференції (4 учасники, 45 хв.), яка підтвердила коректну консолідацію фрагментованих сесій: 5 окремих записів підключення об'єднано в 1 консолідовану сесію тривалістю 38 хвилин (84% присутності), що перевищує встановлений мінімальний поріг. Результати апробації та підтвердження працездатності системи наведено на рис. 3.1–3.5 та у підрозділі 3.5.

Перспективи подальших досліджень. У напрямку практичного впровадження передбачається апробація модуля на масиві реальних конференцій KSU24 з кількісним порівнянням результатів автоматичного та ручного обліку. В архітектурному напрямку патерн «Порти та адаптери» відкриває можливість підключення альтернативних

провайдерів відеоконференцзв’язку (Microsoft Teams, Google Meet) без модифікації доменної логіки. Перспективним є також розширення аналітики за рахунок кореляції даних відвідуваності з показниками академічної успішності здобувачів освіти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Torres Martín C., Acal C., El Homrani M., Mingorance Estrada Á. C. Impact on the Virtual Learning Environment Due to COVID-19. *Sustainability*. 2021. Vol. 13. Art. 582. DOI: https://doi.org/10.3390/su13020582

2. Shuliak I., Ostapchuk I. Online education in Ukraine in extreme conditions: constraints and challenges. *Computer Assisted Language Learning Electronic Journal (CALL-EJ)*. 2024. Vol. 25, no. 1. P. 208–227.

3. Shevchuk B. V. Virtual educational environment of the institution of higher education: realities and perspectives. *Science and Technology Today*. 2023. Vol. 14, no. 28. P. 493–504. DOI: https://doi.org/10.52058/2786-6025-2023-14(28)-493-504

4. Grandinetti J. «From the classroom to the cloud»: Zoom and the platformization of higher education. *First Monday*. 2022. Vol. 27, no. 2. DOI: https://doi.org/10.5210/fm.v27i2.11655

5. Spivakovsky A., Petukhova L., Poltoratskyi M., Lemeshchuk O., Volianiuk A., Kazannikova O., Voropay N., Chepurna S. Theoretical Principles of Measuring and Interpreting Levels of Attention, Involvement and Organizing Feedback of Students to the Educational Process Using Automated Software Products. *ICTERI 2023, CCIS 1980*. 2023. P. 144–159.

6. Vandenberg S., Magnuson M. A comparison of student and faculty attitudes on the use of Zoom, a video conferencing platform: A mixed-methods study. *Nurse Education in Practice*. 2021. Vol. 54. Art. 103138. DOI: https://doi.org/10.1016/j.nepr.2021.103138

7. Spiliopoulou M., Mobasher B., Berendt B., Nakagawa M. A framework for the evaluation of session reconstruction heuristics in web-usage analysis. *INFORMS Journal on Computing*. 2003. Vol. 15, no. 2. P. 171–190. DOI: https://doi.org/10.1287/ijoc.15.2.171.14445

8. KSU24 – віртуальне освітнє середовище Херсонського державного університету. URL: https://ksu24.kspu.edu/ (дата звернення: 08.04.2026).

9. Gladson S. A. An Analysis of Video Conferencing Tools for Learning Management Systems. *International Journal for Multidisciplinary Research (IJFMR)*. 2023. URL: https://www.ijfmr.com/papers/2023/6/42930.pdf (дата звернення: 08.04.2026).

10. Zoom meeting – Moodle Plugins directory. URL: https://moodle.org/plugins/modzoom (дата звернення: 08.04.2026).

11. Canvas and Zoom Integration – Zoom App Marketplace. URL: https://marketplace.zoom.us/apps/1a2P6PVR5qW6l0lK7BmA (дата звернення: 08.04.2026).

12. Blackboard and Zoom Integration – Zoom App Marketplace. URL: https://marketplace.zoom.us/apps/z7U8109SJibZgKx64DCDw (дата звернення: 08.04.2026).

13. Lemeshchuk O. I., Senchishen D. O. University virtual educational environment integration with the zoom service. *Scientific Bulletin of Ivano-Frankivsk National Technical University of Oil and Gas*. 2025. No. 1(58). P. 97–107. DOI: https://doi.org/10.31471/1993-9965-2025-1(58)-97-107

14. Novikov M. M. Implementation of interaction between the Zoom video conferencing platform and the KSU24 platform : кваліфікаційна робота магістра. Херсонський державний університет. Херсон, 2024.

15. Martin R. C. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. Prentice Hall, 2017. ISBN: 978-0134494166.

16. Palermo J. The Onion Architecture: Part 1. URL: https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/ (дата звернення: 08.04.2026).

17. Cockburn A., Garrido de Paz J. M. Hexagonal Architecture Explained: How the Ports & Adapters architecture simplifies your life, and how to implement it. Humans and Technology Press, 2025.

18. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, 2003.

19. Kapitanenko N. P. Legal support for electronic document management. *Scientific Herald of Uzhhorod University. Series LAW*. 2024. Vol. 84, no. 3. P. 130–138. DOI: https://doi.org/10.24144/2307-3322.2024.84.3.20

20. Meena B. S., Ranjan R., Anand S. K. Virtual meetings under attack: Assessing the legal and security risks of zoom bombing in the digital era. *ShodhKosh: Journal of Visual and Performing Arts*. 2023. Vol. 4, no. 2. P. 1256–1263. DOI: https://doi.org/10.29121/shodhkosh.v4.i2.2023.3047

ДОДАТОК А

Вихідний код PlantUML-діаграм

У цьому додатку наведено вихідні файли UML-діаграм у форматі PlantUML, що слугували основою для рисунків 2.1–2.4 основного тексту роботи.

А.1. Компонентна діаграма архітектури модуля (рис. 2.1)

```
@startuml
skinparam componentStyle rectangle
skinparam defaultFontName Times New Roman
skinparam defaultFontSize 12
skinparam shadowing false
skinparam packageStyle frame

title Компонентна діаграма архітектури модуля meetings (Порти та адаптери)

package "Вхідні адаптери" as input_adapters {
    [HTTP API\n(REST endpoints)] as HTTP
}

package "Домен" as domain {
    package "Use Cases" as usecases {
        [UpsertMeetingForLesson] as UC1
        [GetMeetingAttendance] as UC2
        [FindMeetings] as UC3
        [GetMeetingConfig] as UC4
        [UpsertMeetingConfig] as UC5
        [GetProviderSchema] as UC6
    }

    package "Доменні сервіси" as services {
        [AttendanceAnalyzer\n(консолідація сесій)] as Analyzer
        [MeetingConfigResolver\n(каскадний пошук)] as ConfigResolver
    }

    package "Порти (інтерфейси)" as ports {
        interface "IMeetingProviderPort" as IProvider
        interface "IMeetingRepository" as IRepo
        interface "ILearningPlatformPort" as ILearning
        interface "IMeetingAttendanceRecordRepository" as IAttendance
        interface "IMeetingConfigRepository" as IConfig
        interface "IMeetingInviteeRepository" as IInvitee
    }
}

package "Вихідні адаптери" as output_adapters {
    [Zoom API Adapter\n(httpx client)] as ZoomAdapter
    [KSU24 Adapter\n(Django ORM)] as KSU24Adapter
    [Meeting Repository\n(Django ORM)] as RepoAdapter
    [Attendance Repository\n(Django ORM)] as AttendanceAdapter
    [Config Repository\n(Django ORM)] as ConfigAdapter
    [Invitee Repository\n(Django ORM)] as InviteeAdapter
}
```

```
database "БД модуля meetings\n(PostgreSQL)" as ModuleDB
database "БД навчальної\nплатформи KSU24\n(PostgreSQL)" as KSU24DB
cloud "Zoom Cloud API" as ZoomCloud

' Input connections
HTTP --> UC1
HTTP --> UC2
HTTP --> UC3
HTTP --> UC4
HTTP --> UC5
HTTP --> UC6

' Use case to service connections
UC2 --> Analyzer
UC4 --> ConfigResolver
UC1 --> ConfigResolver

' Use case to port connections
UC1 ..> IProvider
UC1 ..> IRepo
UC1 ..> ILearning
UC1 ..> IInvitee
UC2 ..> IProvider
UC2 ..> ILearning
UC2 ..> IAttendance
UC3 ..> IRepo
UC4 ..> IConfig
UC5 ..> IConfig
UC5 ..> IProvider
UC5 ..> ILearning
UC6 ..> IProvider

' Port to adapter connections
IProvider <|.. ZoomAdapter
IRepo <|.. RepoAdapter
ILearning <|.. KSU24Adapter
IAttendance <|.. AttendanceAdapter
IConfig <|.. ConfigAdapter
IInvitee <|.. InviteeAdapter

' External connections
ZoomAdapter --> ZoomCloud
RepoAdapter --> ModuleDB
AttendanceAdapter --> ModuleDB
ConfigAdapter --> ModuleDB
InviteeAdapter --> ModuleDB
KSU24Adapter --> KSU24DB

@enduml
```

Лістинг А.1 – PlantUML-опис компонентної діаграми архітектури модуля meetings

А.2. ER-діаграма бази даних (рис. 2.2)

```
@startuml chapter2_er_diagram

hide empty methods
skinparam defaultFontName Times New Roman
skinparam defaultFontSize 11
skinparam classBorderColor #555555
```

```
skinparam classBackgroundColor #FEFEFE
skinparam classHeaderBackgroundColor #D6E8F7
skinparam ArrowColor #333333
skinparam linetype ortho
skinparam nodesep 60
skinparam ranksep 80

entity "meetings_meetings" as meetings {
  * **id** : UUID «PK»
  --
  * external_id : VARCHAR(32)
  * resource_type : VARCHAR(32)
  * resource_id : VARCHAR(64)
  * organizer_person_id : UUID
  * join_url : VARCHAR(1024)
  * start_url : VARCHAR(1024)
  password : VARCHAR(128)
  topic : VARCHAR(512)
  subtopic : VARCHAR(512)
  start_time : TIMESTAMPTZ
  duration_seconds : INTEGER
  * created_at : TIMESTAMPTZ
  * updated_at : TIMESTAMPTZ
  --
  UNIQUE(resource_type, resource_id)
}

entity "meetings_configurations" as configs {
  * **id** : UUID «PK»
  --
  * resource_type : VARCHAR(20)
  * resource_id : VARCHAR(64)
  * min_attendance_percentage : INTEGER
  * provider_settings : JSONB
  * created_at : TIMESTAMPTZ
  * updated_at : TIMESTAMPTZ
  --
  UNIQUE(resource_type, resource_id)
}

entity "meetings_attendance_records" as attendance {
  * **id** : UUID «PK»
  --
  * **meeting_id** : UUID «FK»
  * user_email : VARCHAR(254)
  person_id : UUID [NULL]
  * total_seconds : INTEGER
  * unverified : BOOLEAN
  * created_at : TIMESTAMPTZ
  * updated_at : TIMESTAMPTZ
  --
  UNIQUE(meeting_id, user_email)
}

entity "meetings_invitees" as invitees {
  * **id** : UUID «PK»
  --
  * **meeting_id** : UUID «FK»
  * person_id : UUID
  * created_at : TIMESTAMPTZ
  --
```

```
  UNIQUE(meeting_id, person_id)
}

meetings ||--o{ attendance : "CASCADE DELETE"
meetings ||--o{ invitees : "CASCADE DELETE"

@enduml
```

Лістинг А.2 – PlantUML-опис ER-діаграми реляційної структури бази даних

А.3. Діаграма прецедентів (рис. 2.3)

```
@startuml chapter2_usecase_diagram

skinparam defaultFontName Times New Roman
skinparam defaultFontSize 12
skinparam shadowing false
skinparam actorStyle awesome
skinparam usecaseBorderColor #555555
skinparam usecaseBackgroundColor #FEFEFE
skinparam actorBorderColor #333333
skinparam packageBorderColor #888888
skinparam packageBackgroundColor #F5F5F5
skinparam ArrowColor #333333

left to right direction

actor "Науково-педагогічний\nпрацівник" as NP
actor "Адміністратор ЗВО" as Admin

rectangle "Модуль автоматизованого обліку відвідуваності" {
    usecase "Створити конференцію\nдля заняття" as UC1
    usecase "Отримати звіт про\nвідвідуваність (з аналізом)" as UC2
    usecase "Налаштувати параметри\nобліку (поріг, мін. %)" as UC3
    usecase "Переглянути список\nконференцій" as UC4
    usecase "Ідентифікувати учасників\n(SSO / email → person_id)" as UC5
    usecase "Консолідувати\nфрагментовані сесії" as UC6
    usecase "Сформувати звіт-\nрекомендацію для журналу" as UC7
}

NP --> UC1
NP --> UC2
NP --> UC3
NP --> UC4

Admin --> UC2
Admin --> UC3

UC2 ..> UC5 : «include»
UC2 ..> UC6 : «include»
UC2 ..> UC7 : «include»

@enduml
```

Лістинг А.3 – PlantUML-опис діаграми прецедентів модуля

А.4. Діаграма послідовностей (рис. 2.4)

```
@startuml chapter2_sequence_diagram
```

```
skinparam defaultFontName Times New Roman
skinparam defaultFontSize 11
skinparam shadowing false
skinparam sequenceArrowThickness 1.5
skinparam sequenceParticipantBorderColor #555555
skinparam sequenceParticipantBackgroundColor #FEFEFE
skinparam sequenceActorBorderColor #333333
skinparam sequenceLifeLineBorderColor #888888
skinparam ArrowColor #333333
skinparam noteBorderColor #888888
skinparam noteBackgroundColor #FFFFF0
skinparam SequenceGroupBorderColor #888888
skinparam SequenceGroupBackgroundColor #F8F8F8
skinparam SequenceGroupBodyBackgroundColor #FAFAFA

title Отримання звіту про відвідуваність

actor "Клієнт" as C
participant "HTTP\nEndpoint" as E
participant "GetMeetingAttendance\nUseCase" as UC
participant "IMeeting\nRepository" as R
participant "AttendanceRecord\nRepository" as AR
participant "IMeetingProvider\nPort" as P
participant "Attendance\nAnalyzer" as A
participant "ILearningPlatform\nPort" as L

C -> E : GET /meeting/{id}/attendance\n?force_refresh&gap_threshold
E -> UC : execute(id, gap, force_refresh)
UC -> R : get_by_id(meeting_id)
R --> UC : Meeting

alt Кеш доступний (force_refresh=false)
    UC -> AR : get_by_meeting_id(id)
    AR --> UC : List[UserAttendanceSummary]
else Завантаження з Zoom
    UC -> P : get_past_meeting_details(external_id)
    P --> UC : PastMeetingDetails\n(List[ParticipantSession])
    UC -> A : analyze(details, gap)
    A --> UC : MeetingAttendanceAnalysis
    UC -> L : resolve_person_ids_by_emails(emails)
    L --> UC : Map[email -> person_id]
    UC -> AR : save(meeting_id, summaries)
    UC -> R : update_duration(id, seconds)
end

UC --> E : MeetingAttendanceAnalysis
E --> C : 200 OK

@enduml
```

Лістинг А.4 – PlantUML-опис діаграми послідовностей: сценарій обробки завершеного заняття

ДОДАТОК Б

Код адаптерів та контейнера впровадження залежностей

У цьому додатку наведено повний вихідний код вихідних адаптерів та DI-контейнера модуля, що описані у підрозділі 3.3 основного тексту.

Б.1. ZoomClient: отримання OAuth-токена та завантаження учасників (підрозділ 3.3)

```
def _get_oauth_token(self) -> str:
    now = time.time()
    if self._cached_token and now < self._token_expiry:
        return self._cached_token
    response = httpx.post(
        self._options.base_token_url,
        params={"grant_type": "account_credentials",
                "account_id": self._options.account_id},
        auth=(self._options.client_id, self._options.client_secret),
        timeout=10,
    )
    response.raise_for_status()
    data = response.json()
    self._cached_token = data["access_token"]
    self._token_expiry = (
        now + data["expires_in"] - self._options.token_leeway_seconds
    )
    return self._cached_token

def get_past_meeting_participants(self, meeting_id: str) -> list[dict]:
    url = self._url(f"past_meetings/{meeting_id}/participants")
    params: dict = {"page_size": 300}
    participants: list[dict] = []
    while True:
        response = httpx.get(
            url, headers=self._get_headers(), params=params, timeout=10
        )
        response.raise_for_status()
        data = response.json()
        participants.extend(data.get("participants", []))
        next_page_token = data.get("next_page_token", "")
        if not next_page_token:
            break
        params["next_page_token"] = next_page_token
    return participants
```

Лістинг Б.1 – ZoomClient: OAuth-токен та пагінація учасників

Б.2. Ksu24Adapter.resolvepersonidsbyemails() (підрозділ 3.3)

```
def resolve_person_ids_by_emails(
    self, emails: set[str]
) -> dict[str, UUID]:
    if not emails:
        return {}
    persons = Person.objects.filter(
        user__email__in=emails,
```

```
    ).select_related("user")
    return {
        p.user.email.lower(): p.id
        for p in persons
        if p.user and p.user.email
    }
```

Лістинг Б.2 – Ksu24Adapter: зіставлення email з UUID-ідентифікаторами персон

Б.3. Функція getmeetingscontainer() – DI-контейнер модуля (підрозділ 3.3)

```
@lru_cache(maxsize=1)
def get_meetings_container() -> MeetingContainer:
    learning_platform_adapter = _build_learning_platform_adapter()
    meeting_repository = _build_meeting_repository()
    invitee_repository = _build_invitee_repository()
    meeting_provider = _build_meeting_provider_adapter()
    config_repository = _build_config_repository()
    attendance_record_repository = _build_attendance_record_repository()
    config_resolver = _build_config_resolver()

    get_provider_schema_uc = GetProviderSchemaUseCase(
        meeting_provider=meeting_provider
    )
    get_meeting_config_uc = Get\allowbreak Meeting\allowbreak
Config\allowbreak Use\allowbreak Case(
        learning_platform_adapter=learning_platform_adapter,
        meeting_provider=meeting_provider,
        config_resolver=config_resolver,
    )
    upsert_meeting_config_uc = Upsert\allowbreak Meeting\allowbreak
Config\allowbreak Use\allowbreak Case(
        learning_platform_adapter=learning_platform_adapter,
        meeting_provider=meeting_provider,
        config_repository=config_repository,
    )
    upsert_meeting_for_lesson_uc = Upsert\allowbreak Meeting\allowbreak
For\allowbreak Lesson(
        learning_platform_adapter=learning_platform_adapter,
        meeting_provider_adapter=meeting_provider,
        meeting_repository=meeting_repository,
        invitee_repository=invitee_repository,
        config_resolver=config_resolver,
    )
    find_meetings_uc = Find\allowbreak Meetings\allowbreak Use\allowbreak
Case(
        meeting_repository=meeting_repository,
    )
    get_meeting_attendance_uc = Get\allowbreak Meeting\allowbreak
Attendance\allowbreak Use\allowbreak Case(
        meeting_repository=meeting_repository,
        meeting_provider_adapter=meeting_provider,
        learning_platform_adapter=learning_platform_adapter,
        attendance_analyzer=Attendance\allowbreak Analyzer(),
        attendance_record_repository=attendance_record_repository,
    )
    return MeetingContainer(
        learning_platform_adapter=learning_platform_adapter,
```

```
        meeting_provider_client=_build_meeting_provider_client(),
        meeting_provider=meeting_provider,
        meeting_repository=meeting_repository,
        invitee_repository=invitee_repository,
        config_repository=config_repository,
        attendance_record_repository=attendance_record_repository,
        config_resolver=config_resolver,
        find_meetings=find_meetings_uc,
        get_meeting_attendance=get_meeting_attendance_uc,
        get_provider_schema=get_provider_schema_uc,
        get_meeting_config=get_meeting_config_uc,
        upsert_meeting_config=upsert_meeting_config_uc,
        upsert_meeting_for_lesson=upsert_meeting_for_lesson_uc,
    )

def reset_meetings_container_for_tests() -> None:
    """Скинути всі lru_cache-побудови контейнера."""
    get_meetings_container.cache_clear()
    _build_meeting_provider_client.cache_clear()
    _build_learning_platform_adapter.cache_clear()
    _build_meeting_repository.cache_clear()
    _build_meeting_provider_adapter.cache_clear()
    _build_config_repository.cache_clear()
    _build_config_resolver.cache_clear()
    _build_attendance_record_repository.cache_clear()
    _build_invitee_repository.cache_clear()
```

Лістинг Б.3 – Функція get_meetings_container() – збірка об'єктного графа

ДОДАТОК В

Порти (інтерфейси) доменного ядра

У цьому додатку наведено абстрактні інтерфейси (порти), що визначають контракти взаємодії доменного ядра із зовнішніми системами.

В.1. Порт провайдера відеоконференцій IMeetingProviderPort

```
from abc import ABC, abstractmethod
from typing import List, Optional

from ..entities import Meeting, MeetingConfig, ProviderSettingDefinition
from ..entities.past_meeting_details import PastMeetingDetails


class IMeeting\allowbreak Provider\allowbreak Port(ABC):
    """Інтерфейс для взаємодії із зовнішнім провайдером мітингів."""

    @abstractmethod
    def get_provider_schema(self) -> List[ProviderSettingDefinition]:
        """Повертає список доступних налаштувань провайдера."""
        ...

    @abstractmethod
    def create_meeting(
        self,
        meeting: Meeting,
        config: MeetingConfig,
        host_email: str,
        invitee_emails: set[str],
    ) -> Meeting:
        """
        Створити мітинг у зовнішньому провайдері.
        Повертає Meeting із заповненими external_id, URLs та password.
        """
        ...

    @abstractmethod
    def get_past_meeting_details(
        self,
        meeting_external_id: str,
    ) -> Optional[PastMeetingDetails]:
        """Отримати детальну інформацію про завершений мітинг."""
        ...
```

Лістинг В.1 – Інтерфейс IMeetingProviderPort – контракт провайдера мітингів

В.2. Порт навчальної платформи ILearningPlatformPort

```
from abc import ABC, abstractmethod
from typing import Optional
from uuid import UUID

from ..entities import GradebookInfo, Lesson
```

```
class ILearning\allowbreak Platform\allowbreak Port(ABC):
    """Інтерфейс для взаємодії з платформою навчання (VLE)."""

    @abstractmethod
    def get_lesson(self, lesson_id: UUID) -> Optional[Lesson]: ...

    @abstractmethod
    def get_gradebook_info(
        self, gradebook_id: UUID
    ) -> Optional[GradebookInfo]: ...

    @abstractmethod
    def get_person_id(self, user_id: int) -> Optional[UUID]: ...

    @abstractmethod
    def resolve_person_ids_by_emails(
        self, emails: set[str]
    ) -> dict[str, UUID]: ...
```

Лістинг В.2 – Інтерфейс ILearningPlatformPort – контракт навчальної платформи

В.3. Порт репозиторію мітингів IMeetingRepository

```
from abc import ABC, abstractmethod
from datetime import datetime
from typing import Optional
from uuid import UUID

from ..entities import Meeting
from ..entities.meeting_resource_type import MeetingResourceType


class IMeeting\allowbreak Repository(ABC):
    """Порт репозиторію для доступу до записів мітингів."""

    @abstractmethod
    def get_by_id(self, meeting_id: UUID) -> Optional[Meeting]: ...

    @abstractmethod
    def save(self, meeting: Meeting) -> Meeting: ...

    @abstractmethod
    def get_by_resource(
        self,
        resource_type: MeetingResourceType,
        resource_id: str,
    ) -> Optional[Meeting]: ...

    @abstractmethod
    def find_by_resources(
        self,
        resource_type: MeetingResourceType,
        resource_ids: list[str],
    ) -> list[Meeting]: ...

    @abstractmethod
    def update_duration(
        self, meeting_id: UUID, duration_seconds: int
```

```
    ) -> None: ...

    @abstractmethod
    def find_by_filters(
        self,
        meeting_ids: list[UUID] | None = None,
        resource_type: MeetingResourceType | None = None,
        resource_ids: list[str] | None = None,
        organizer_person_id: UUID | None = None,
        start_time_from: datetime | None = None,
        start_time_to: datetime | None = None,
        participant_person_id: UUID | None = None,
    ) -> list[Meeting]:
        """
        Гнучкий пошук мітингів за набором опціональних фільтрів.
        participant_person_id -- фільтрує мітинги, де персона є
        організатором або запрошеним учасником.
        """
        ...
```

Лістинг В.3 – Інтерфейс IMeetingRepository – контракт персистентності мітингів

ДОДАТОК Г

Доменна сутність Meeting та адаптер Zoom API

Г.1. Доменна сутність Meeting

```
from datetime import datetime
from uuid import UUID

from pydantic import BaseModel, Field

from .meeting_resource_type import MeetingResourceType


class Meeting(BaseModel):
    """Доменна сутність мітингу, прив'язаного до ресурсу."""

    id: UUID = Field(..., description="UUID внутрішнього запису.")
    resource_type: MeetingResourceType = Field(
        ..., description="Тип ресурсу (LESSON тощо)."
    )
    resource_id: str = Field(
        ..., description="Ідентифікатор ресурсу."
    )
    organizer_person_id: UUID = Field(
        ..., description="UUID персони організатора."
    )
    meeting_external_id: str = Field(
        ..., description="Зовнішній ID (Zoom Meeting ID)."
    )
    meeting_password: str = Field(default="")
    meeting_join_url: str = Field(default="")
    meeting_start_url: str = Field(default="")
    topic: str = Field(default="", description="Назва конференції.")
    subtopic: str = Field(default="", description="Підтема заняття.")
    meeting_start_time: datetime | None = Field(default=None)
    duration_seconds: int | None = Field(
        default=None,
        description="Фактична тривалість (заповнюється після завершення).",
    )
    invitee_person_ids: list[UUID] = Field(default_factory=list)
```

Лістинг Г.1 – Доменна сутність Meeting – центральна сутність модуля

Г.2. Адаптер ZoomMeetingProviderAdapter

```
class ZoomMeetingProviderAdapter(IMeeting\allowbreak Provider\allowbreak
Port):
    """Адаптер Zoom як зовнішнього провайдера мітингів."""

    DEFAULT_DURATION_MINUTES = 80

    def __init__(self, client: ZoomClient) -> None:
        self._client = client

    def get_provider_schema(self) -> List[ProviderSettingDefinition]:
        return ZOOM_PROVIDER_SCHEMA

    def create_meeting(
```

```
        self,
        meeting: Meeting,
        config: MeetingConfig,
        host_email: str,
        invitee_emails: set[str],
    ) -> Meeting:
        ps = config.provider_settings
        settings = ZoomMeetingSettingsDto(
            meeting_invitees=[
                ZoomMeetingInviteeDto(email=e) for e in invitee_emails
            ],
            provider_settings=ps,
        )
        request_dto = ZoomCreateMeetingRequestDto(
            topic=meeting.topic,
            start_time=meeting.meeting_start_time.isoformat(),
            duration=int(ps.get("duration_minutes",
                                self.DEFAULT_DURATION_MINUTES)),
            password=ps.get("password"),
            settings=settings,
        )
        try:
            data = self._client.create_meeting(
                user_email=host_email,
                data=request_dto.to_zoom_payload(),
            )
        except httpx.HTTPStatusError as exc:
            raise ZoomMeetingException(
                f"meetings.E301 | Failed to create meeting: "
                f"{exc.response.status_code}"
            ) from exc
        return meeting.model_copy(update={
            "meeting_external_id": str(data["id"]),
            "meeting_join_url": data["join_url"],
            "meeting_start_url": data["start_url"],
            "meeting_password": data.get("password", ""),
        })

    def get_past_meeting_details(
        self, meeting_external_id: str
    ) -> Optional[PastMeetingDetails]:
        try:
            meeting_raw = self._client.get_past_meeting(
                meeting_external_id
            )
        except httpx.HTTPStatusError:
            return None

        start_dt = self._parse_zoom_datetime(meeting_raw.get("start_time"))
        end_dt = self._parse_zoom_datetime(meeting_raw.get("end_time"))
        if start_dt is None or end_dt is None:
            return None

        try:
            participants_raw = self._client.get_past_meeting_participants(
                meeting_external_id
            )
        except httpx.HTTPStatusError:
            participants_raw = []

        sessions: list[Participant\allowbreak Session] = []
```

```
        for item in participants_raw:
            if not isinstance(item, dict):
                continue
            email = item.get("user_email")
            if not isinstance(email, str) or not email.strip():
                continue
            join_t = self._parse_zoom_datetime(item.get("join_time"))
            leave_t = self._parse_zoom_datetime(item.get("leave_time"))
            if join_t is None or leave_t is None:
                continue
            sessions.append(Participant\allowbreak Session(
                user_email=email.strip().lower(),
                join_time=join_t,
                leave_time=leave_t,
            ))
        return PastMeetingDetails(
            start_time=start_dt,
            end_time=end_dt,
            participant_sessions=sessions,
        )

    @staticmethod
    def _parse_zoom_datetime(value) -> datetime | None:
        if not isinstance(value, str) or not value.strip():
            return None
        value = value.strip()
        if value.endswith("Z"):
            value = value[:-1] + "+00:00"
        try:
            parsed = datetime.fromisoformat(value)
            if parsed.tzinfo is None:
                parsed = parsed.replace(tzinfo=timezone.utc)
            return parsed.astimezone(timezone.utc)
        except ValueError:
            return None
```

Лістинг Г.2 – ZoomMeetingProviderAdapter – реалізація IMeetingProviderPort

ДОДАТОК Д

Репозиторії та повний код адаптера KSU24

Д.1. Репозиторій відвідуваності MeetingAttendanceRepository

```
class MeetingAttendanceRepository(IMeeting\allowbreak Attendance\allowbreak
Record\allowbreak Repository):

    def save(
        self, meeting_id: UUID, attendees: list[User\allowbreak
Attendance\allowbreak Summary]
    ) -> None:
        records = [
            MeetingAttendanceRecord(
                meeting_id=meeting_id,
                user_email=attendee.user_email,
                person_id=attendee.person_id,
                total_seconds=attendee.total_seconds,
                unverified=attendee.unverified,
            )
            for attendee in attendees
        ]
        MeetingAttendanceRecord.objects.bulk_create(
            records,
            update_conflicts=True,
            unique_fields=["meeting_id", "user_email"],
            update_fields=[
                "person_id", "total_seconds",
                "unverified", "updated_at",
            ],
        )

    def get_by_meeting_id(
        self, meeting_id: UUID
    ) -> list[User\allowbreak Attendance\allowbreak Summary]:
        records = MeetingAttendanceRecord.objects.filter(
            meeting_id=meeting_id
        )
        return [MeetingAttendanceMapper.to_domain(r) for r in records]
```

Лістинг Д.1 – MeetingAttendanceRepository – збереження та читання записів відвідуваності

Д.2. Репозиторій конфігурацій MeetingConfigRepository

```
class MeetingConfig\allowbreak Repository(IMeeting\allowbreak
Config\allowbreak Repository):

    def get_config(
        self, resource_type: ResourceType, resource_id: str
    ) -> Optional[MeetingConfig]:
        obj = MeetingConfiguration.objects.filter(
            resource_type=resource_type.value,
            resource_id=resource_id,
        ).first()
        if obj is None:
            return None
        return MeetingConfigMapper.to_domain(obj)
```

```
    def upsert_config(self, config: MeetingConfig) -> MeetingConfig:
        obj, _ = MeetingConfiguration.objects.update_or_create(
            resource_type=config.resource_type.value,
            resource_id=config.resource_id,
            defaults={
                "min_attendance_percentage":
                    config.min_attendance_percentage,
                "provider_settings": config.provider_settings,
            },
        )
        return MeetingConfigMapper.to_domain(obj)
```

Лістинг Д.2 – MeetingConfigRepository – зберігання та каскадне читання конфігурацій

Д.3. Метод Ksu24Adapter.getlesson() – отримання даних заняття

```
def get_lesson(self, lesson_id: UUID) -> Optional[DomainLesson]:
    """Отримати доменну сутність заняття за UUID з бази KSU24."""
    try:
        lesson = (
            GradebookLesson.objects
            .select_related(
                "gradebook__teacher__worker__person__user"
            )
            .prefetch_related(
                "gradebook__studentgradebook_set"
                "__student__person__user"
            )
            .get(id=lesson_id)
        )
    except GradebookLesson.DoesNotExist:
        logger.warning(
            "Lesson %s not found for auto-sync, skipping", lesson_id
        )
        return None

    gradebook = lesson.gradebook
    teacher = gradebook.teacher

    if teacher is None:
        raise Business\allowbreak Rule\allowbreak Error(
            "meetings.E104 | Для журналу не призначено викладача"
        )

    person = teacher.worker.person
    teacher_user = person.user
    teacher_display = getattr(
        teacher, "teacher_info_short", person.full_name
    )

    student_emails: set[str] = set()
    for sg in gradebook.studentgradebook_set.select_related(
        "student__person__user"
    ):
        user = getattr(sg.student.person, "user", None)
        email = getattr(user, "email", None)
        if email:
            student_emails.add(email.lower())
```

```
    return DomainLesson(
        id=lesson.id,
        discipline_name=gradebook.discipline_name,
        start_date=lesson.date,
        teacher_email=teacher_user.email,
        teacher_display=teacher_display,
        teacher_person_id=person.id,
        student_emails=student_emails,
        gradebook_id=str(gradebook.id),
    )
```

Лістинг Д.3 – Ksu24Adapter.get_lesson() – завантаження заняття з Django ORM KSU24