МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет	Інформаційних технологій
Кафедра	Інформатики і прикладного програмного забезпечення
Спеціальність	Інженерія програмного забезпечення
Форма навчання	Денна

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Покутній Даниїл Миколайович
(прізвище, ім'я, по батькові здобувача)

на тему «Розробка додатку для контролю відвідування занять студентами»
(повна назва теми)

за матеріалами праць провідних спеціалістів з розробки ПЗ та проектування БД
(повна назва бази дослідження)

науковий керівник _к.е.н., доцент_ _____ __Баран С.В.__
(наук. ступінь, вчене звання) *(підпис)* *(прізвище, ініціали)*

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 11.06.2025 р. № 12
Завідувач кафедри _____
(підпис)
д.т.н., професор Зеленський О.С.
Наук. ступень, вчене звання *Ініціали, прізвище*

Кривий Ріг – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет	Інформаційних технологій
Кафедра	Інформатики і прикладного програмного забезпечення
Спеціальність	Інженерія програмного забезпечення
Форма навчання	Денна

«ЗАТВЕРДЖУЮ»
Завідувач кафедри _____ Зеленський О.С.
(підпис) (Прізвище, ініціали)
« 11 » червня 2025 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ

1. Тема роботи «Розробка додатку для контролю відвідування занять студентами»

Керівник роботи к.е.н., доцент Баран С.В.
затверджені наказом закладу вищої освіти від «04» квітня 2025 р. № 222-ст

2. Строк подання здобувачем роботи до «09» червня 2025 р.

3. Зміст кваліфікаційної роботи, об'єкт, предмет та мета дослідження:

Розділ 1. Постановка задачі

Розділ 2. Розробка алгоритму розв'язання задачі

Розділ 3. Організація інформаційного забезпечення

Розділ 4. Розробка програмного забезпечення

Об'єкт дослідження: процес контролю відвідуваності занять студентами в закладах вищої освіти.

Предмет дослідження: м*етоди, моделі та програмні засоби для автоматизації контролю відвідуваності студентів з використанням веб-технологій та десктопних додатків.*

Мета кваліфікаційної роботи: *розробка програмного додатку для ефективного контролю відвідування занять студентами, що включає веб-сервіс на PHP Laravel, клієнтську веб-частину на Next.js та десктопний додаток на C++, для збору, обробки та візуалізації даних про відвідуваність.*

5. Дата видачі завдання «04» квітня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів МДР	Строк виконання етапів роботи	Відмітка керівника про виконання етапів (дата, підпис)
1	Підготовка розділу 1	04.04.2025-13.04.2025	
2	Підготовка розділу 2	14.04.2025-26.04.2025	
3.	Підготовка розділу 3	27.04.2025-15.05.2025	
4	Підготовка розділу 4	16.05.2025-08.06.2025	
5	Реєстрація завершеної кваліфікаційної роботи	09.06.2025	Реєстраційний №___ «09»червня 2025 р.
6	Отримання відгуку від наукового керівника	10.06.2025	
7	Подання кваліфікаційної роботи на перегляд завідувачу кафедри	11.06.2025	
8	Отримання зовнішньої рецензії	12.06.2025	
9	Попередній захист кваліфікаційної роботи на кафедрі	13.06.2025	
10	Підготовка до захисту в ЕК	16.06.2025-21.06.2025	

Завдання підготував науковий керівник _____ Баран С.В.
(підпис) (прізвище та ініціали)

Завдання одержав _____ Покутній Д.М
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

на кваліфікаційну бакалаврську роботу

«Розробка додатку для контролю відвідування занять студентами»

Покутній Даниїл Миколайович

Кваліфікаційна бакалаврська робота на здобуття освітньо-кваліфікаційного рівня бакалавра зі спеціальності 121 «Інженерія програмного забезпечення» – Державний університет економіки і технологій – Кривий Ріг, 2025.

У бакалаврській дипломній роботі були розроблені додаток для контролю відвідуваності студентів. Система складається з декількох частин, десктопного додатку створеного на мові C++ , клієнтського веб-компоненту побудованого за допомогою бібліотеки Next.js, та серверної частини реалізованої на мові програмування PHP з використанням фреймворку Laravel. Інформація була розміщена в реляційній базі даних PostgreSQL.

Ключові слова: Контроль відвідуваності; студенти; PHP Laravel; NEXT.JS; C++; база даних; програмне забезпечення; автоматизація.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД(база даних)	Впорядкований набір логічно взаємопов’язаних даних, що використовуються спільно та призначені для задоволення інформаційних потреб користувачів.
СУБД	Система управління базами даних.
ПЗ	Програмне забезпечення.

ЗМІСТ

Вступ

Сучасний етап розвитку вищої освіти нерозривно пов'язаний із процесами глобальної цифровізації, що висувають нові вимоги до організації навчального процесу, ефективності управління освітніми установами та якості підготовки майбутніх фахівців. У цьому контексті, питання контролю та аналізу відвідуваності занять студентами набуває особливої ваги, оскільки регулярна участь в освітньому процесі є одним із ключових чинників успішного засвоєння знань та формування професійних компетенцій. Однак, як показує практика багатьох закладів вищої освіти, існуючі підходи до обліку присутності студентів часто виявляються недостатньо ефективними та не відповідають викликам сьогодення.

Під час спостережень за навчальним процесом та аналізу існуючої ситуації, було відзначено тривожну тенденцію до поступового зниження загального рівня відвідуваності занять студентами. Однією з причин такого явища, на мою думку, є відсутність прозорої, чіткої та зручної системи моніторингу та регуляції відвідуваності, яка б однаково ефективно працювала як для традиційних аудиторних занять, так і для занять, що проводяться у дистанційному або змішаному форматах. Недостатня оперативність отримання даних про пропуски, складність їх аналізу та відсутність зручних інструментів для взаємодії між студентами, викладачами та адміністрацією щодо питань відвідуваності призводять до того, що проблема часто залишається поза належною увагою до моменту виникнення суттєвих академічних заборгованостей.

Паралельно з цим, спостерігається значне зростання навантаження як на викладацький склад, так і на самих студентів. Викладачі змушені витрачати дорогоцінний час на рутинні процедури ведення паперових журналів, підрахунку пропусків та підготовки відповідних звітів, що відволікає їх від основної методичної та науково-педагогічної діяльності. Для студентів, особливо в умовах гнучких графіків або змішаних форм навчання, відсутність

єдиного інформаційного простору для швидкого доступу до розкладу, навчальних матеріалів, завдань та інформації про власну відвідуваність створює додаткові труднощі в плануванні навчальної діяльності та самоконтролі. Необхідність вручну уточнювати інформацію, відстежувати зміни в розкладі та контролювати власну присутність на численних заняттях також збільшує організаційне навантаження на здобувачів освіти.

Вищезазначені проблеми зумовлюють високу актуальність розробки та впровадження сучасної автоматизованої системи, яка б не лише забезпечувала точний облік відвідуваності, але й слугувала б зручним інструментом для всіх учасників освітнього процесу. Така система повинна сприяти оптимізації робочого часу викладачів, підвищенню рівня самоорганізації студентів та наданню адміністрації об'єктивних даних для прийняття управлінських рішень. Саме на вирішення цих нагальних проблем і спрямована дана кваліфікаційна бакалаврська робота.

Метою даної кваліфікаційної бакалаврської роботи є розробка програмного додатку для контролю відвідування занять студентами, який, перш за все, спрямований на спрощення та оптимізацію взаємодії студентів та викладачів з навчальним процесом. Ключова ідея проєкту полягає у створенні єдиного, інтуїтивно зрозумілого середовища, що допоможе студентам швидше та простіше отримувати доступ до актуального розкладу, навчальних матеріалів, завдань та оперативно приєднуватися до онлайн-занять, а викладачам – легко та ефективно фіксувати та переглядати дані про відвідуваність, зменшуючи паперову роботу та часові витрати. Досягнення цієї мети передбачає створення комплексного рішення, що включає серверний веб-сервіс на PHP Laravel для обробки даних та логіки, сучасну клієнтську веб-частину на Next.js для зручного доступу через браузер, а також спеціалізований десктопний додаток на C++ для розширених можливостей аналізу та візуалізації даних про відвідуваність.

Для досягнення поставленої мети було сформульовано наступні завдання дослідження:

1. Провести поглиблений аналіз існуючих проблем у сфері контролю відвідуваності студентів, зокрема, дослідити фактори, що впливають на зниження відвідуваності та зростання навантаження на учасників навчального процесу.
2. Здійснити огляд та критичний аналіз доступних програмних рішень та систем, призначених для автоматизації обліку відвідуваності, з метою виявлення їхніх переваг, недоліків та можливостей для створення більш ефективного та користувацько-орієнтованого продукту.
3. На основі виявлених потреб студентів та викладачів, сформулювати деталізовані функціональні та нефункціональні вимоги до розроблюваного програмного додатку, акцентуючи увагу на зручності використання, швидкості доступу до інформації та надійності системи.
4. Обґрунтувати вибір конкретного стеку технологій (PHP Laravel, Next.js (React), C++ з ImGui, PostgreSQL) для реалізації кожного компонента системи, враховуючи їхню відповідність поставленим завданням, потенціал для масштабування та наявний досвід розробника.
5. Розробити комплексну архітектуру програмного додатку, що забезпечує ефективну взаємодію між серверною частиною, веб-клієнтом та десктопним додатком, а також спроектувати оптимальну структуру бази даних PostgreSQL для зберігання та управління великими обсягами даних.
6. Реалізувати основний функціонал серверної частини (API) на PHP Laravel, включаючи механізми автентифікації, управління даними про користувачів, розклад, навчальні активності та фіксацію відвідуваності.
7. Розробити інтерактивну та адаптивну клієнтську веб-частину на Next.js, що забезпечує студентам та викладачам зручний доступ до розкладу, матеріалів, завдань, інформації про відвідуваність та можливість швидкого приєднання до онлайн-занять.
8. Створити спеціалізований десктопний додаток на C++, який надасть викладачам та адміністрації інструменти для поглибленого аналізу та візуалізації даних відвідуваності, зокрема, на часовій шкалі.

9. Розробити методику та провести комплексне ручне тестування всіх компонентів системи для забезпечення їхньої коректної роботи, стабільності та відповідності функціональним вимогам.
10. Проаналізувати результати розробки, описати ключові аспекти реалізованих користувацьких інтерфейсів та сценаріїв їх використання, а також оцінити практичне значення створеного програмного продукту для полегшення навчального процесу.

Об’єктом дослідження є процес взаємодії студентів та викладачів з системою обліку та контролю відвідуваності занять у закладах вищої освіти.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ РОЗРОБКИ

1.1. Актуальність задачі автоматизації контролю відвідуваності

В контексті модернізації системи вищої освіти та посилення вимог до якості підготовки фахівців, питання ефективного управління навчальним процесом набуває першочергового значення. Одним із фундаментальних аспектів, що безпосередньо корелює з академічною успішністю студентів та загальним рівнем освітніх стандартів, є систематичний та об'єктивний контроль відвідуваності аудиторних та дистанційних занять. Регулярна присутність здобувачів вищої освіти на запланованих навчальних заходах є невід'ємною умовою для повноцінного сприйняття та засвоєння програмного матеріалу, активної участі в дискусіях, виконання практичних завдань та формування стійких професійних компетенцій. Недостатня увага до цього аспекту може призводити до зниження рівня знань, накопичення академічної заборгованості та, як наслідок, до погіршення загальних показників успішності навчального закладу.

Аналіз сучасної практики функціонування закладів вищої освіти свідчить про те, що традиційні методи обліку відвідуваності, які здебільшого покладаються на ручне ведення паперових журналів викладачами, демонструють низку суттєвих системних недоліків. По-перше, такі методи характеризуються значною трудомісткістю, вимагаючи від викладацького складу регулярних витрат робочого часу на механічну фіксацію присутніх та відсутніх, а також на подальший підрахунок пропусків та підготовку відповідної звітності для адміністративних підрозділів. Це відволікає викладачів від їхніх безпосередніх науково-педагогічних обов'язків та методичної роботи. По-друге, людський фактор неминуче призводить до виникнення помилок, неточностей та описок при заповненні журналів, що ставить під сумнів об'єктивність та достовірність зібраних даних. По-третє, паперові носії інформації є вразливими до фізичного

зносу, втрати або пошкодження, а також не забезпечують належного рівня захисту від несанкціонованих виправлень чи підробок.

Особливої гостроти проблема набуває в умовах зростаючої популярності змішаних та дистанційних форм навчання, де традиційні підходи до контролю присутності стають практично неможливими або надзвичайно ускладненими. Відсутність єдиної, прозорої та оперативної системи моніторингу відвідуваності онлайн-занять та активності студентів на освітніх платформах створює сприятливе підґрунтя для зниження рівня залученості та самодисципліни здобувачів освіти.

Спостереження за динамікою навчального процесу в сучасних університетах, зокрема і дослідження попередніх етапів підготовки даної кваліфікаційної роботи, виявило тривожну тенденцію до поступового зниження середнього рівня відвідуваності занять студентами різних спеціальностей та форм навчання. Це явище, на мою думку, частково зумовлене саме відсутністю дієвого, зручного та мотивуючого механізму контролю, який би не сприймався студентами як суто каральний захід, а радше як інструмент самоорганізації та відповідального ставлення до навчання. Більше того, відсутність чіткої системи регуляції та оперативного інформування про пропуски призводить до збільшення адміністративного та організаційного навантаження як на викладачів, так і на самих студентів. Викладачі змушені витрачати додатковий час на з'ясування причин відсутності, індивідуальні консультації для тих, хто пропустив значний обсяг матеріалу, та підготовку численних довідок і звітів. Студенти, у свою чергу, стикаються з труднощами в отриманні актуальної інформації про розклад, пропущені заняття, навчальні матеріали та завдання, що дезорганізує їхню навчальну діяльність.

Таким чином, об'єктивна необхідність автоматизації процесу контролю відвідуваності студентів є беззаперечною та продиктована як потребою підвищення ефективності управління навчальним процесом, так і завданням покращення якості освітніх послуг. Розробка та впровадження сучасних програмних комплексів, здатних забезпечити точний, оперативний та

багатоаспектний облік присутності, є актуальним науково-практичним завданням, вирішення якого сприятиме оптимізації роботи всіх учасників освітнього процесу та створенню більш сприятливих умов для успішного навчання.

1.2. Аналіз існуючих рішень та обґрунтування вимог до системи

Вирішення завдання автоматизації контролю відвідуваності студентів вимагає попереднього ретельного аналізу існуючих на ринку програмних продуктів та систем, що пропонують схожий або дотичний функціонал. Такий аналіз дозволяє не лише оцінити поточний рівень розвитку технологій у даній сфері, але й виявити сильні та слабкі сторони наявних рішень, що, у свою чергу, є основою для формування обґрунтованих та всебічних вимог до проектованої системи. Дослідження ринку програмного забезпечення для освітніх установ показує наявність широкого спектру продуктів, починаючи від окремих модулів у складі великих систем управління навчанням (LMS) і закінчуючи спеціалізованими автономними додатками. Однією з найпоширеніших категорій є інтегровані модулі відвідуваності в рамках таких відомих LMS, як, наприклад, Moodle (Рис 1.1). Дана платформа, завдяки своїй відкритості та гнучкості, дозволяє викладачам створювати навчальні сесії та фіксувати присутність студентів з різними статусами (присутній, відсутній, запізнився, поважна причина). Перевагою такого підходу є тісна інтеграція з іншими освітніми інструментами платформи, такими як курси, завдання, оцінки, що створює єдине інформаційне середовище. Однак, функціонал стандартних модулів відвідуваності в Moodle може виявитися недостатнім для комплексного аналізу даних або для специфічних потреб навчального закладу, а їхній інтерфейс не завжди відповідає сучасним вимогам до ергономіки та інтуїтивності.

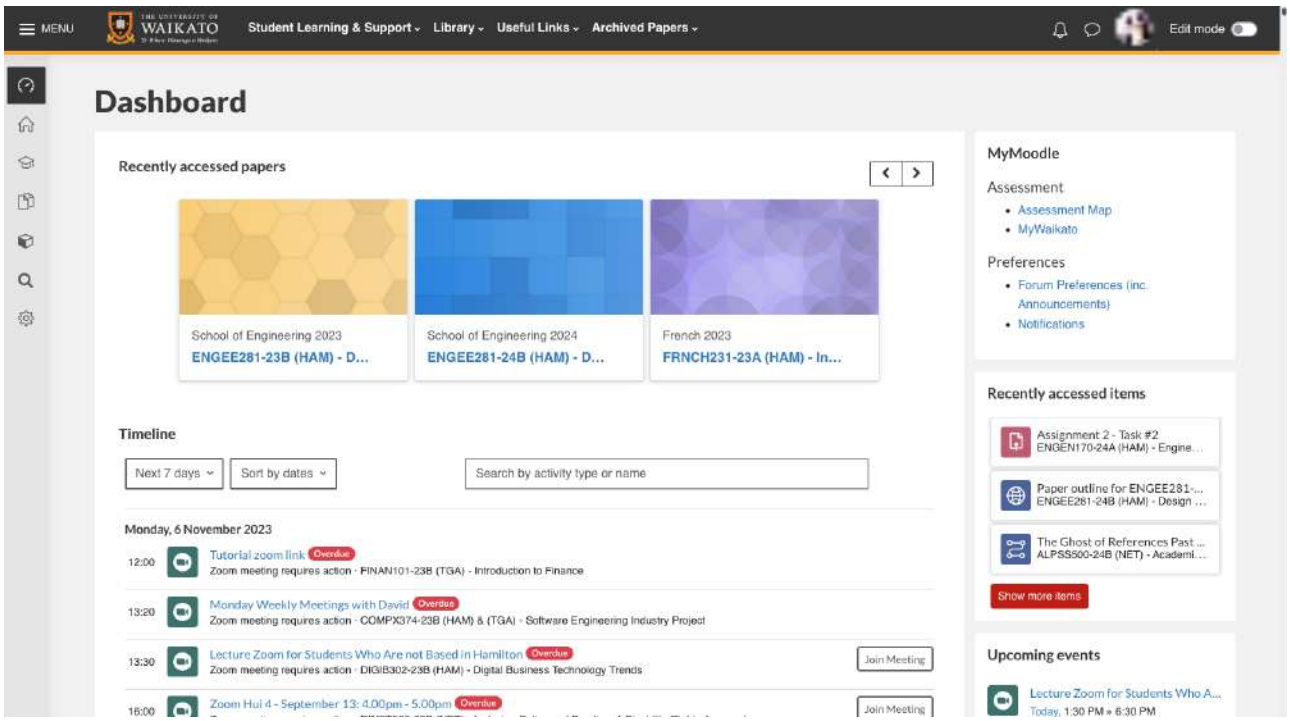

Рис. 1.1. Головна сторінка університету з Moodle

Іншу групу складають комерційні хмарні системи управління навчальним закладом (School Management Systems / ERP), такі як Classter (Рис.1.2) або Fedena (Рис. 1.3) (хоча остання має і версію з відкритим кодом). Ці комплексні рішення зазвичай пропонують значно ширший набір функцій, включаючи не лише деталізований контроль відвідуваності в режимі реального часу, але й можливість інтеграції з системами контролю доступу (наприклад, RFID-картки, біометричні сканери), автоматичні сповіщення батькам або опікунам про пропуски, розширені інструменти аналітики та формування звітності, а також мобільні додатки для всіх категорій користувачів. Безперечною перевагою таких систем є їхня багатофункціональність, сучасний дизайн та професійна технічна підтримка. Проте, їхня вартість може бути значною, а надлишковість функціоналу – невиправданою для закладів, що потребують лише автоматизації конкретного процесу обліку відвідуваності. Крім того, залежність від хмарної інфраструктури та зовнішнього провайдера може викликати певні занепокоєння щодо безпеки даних та стабільності роботи.

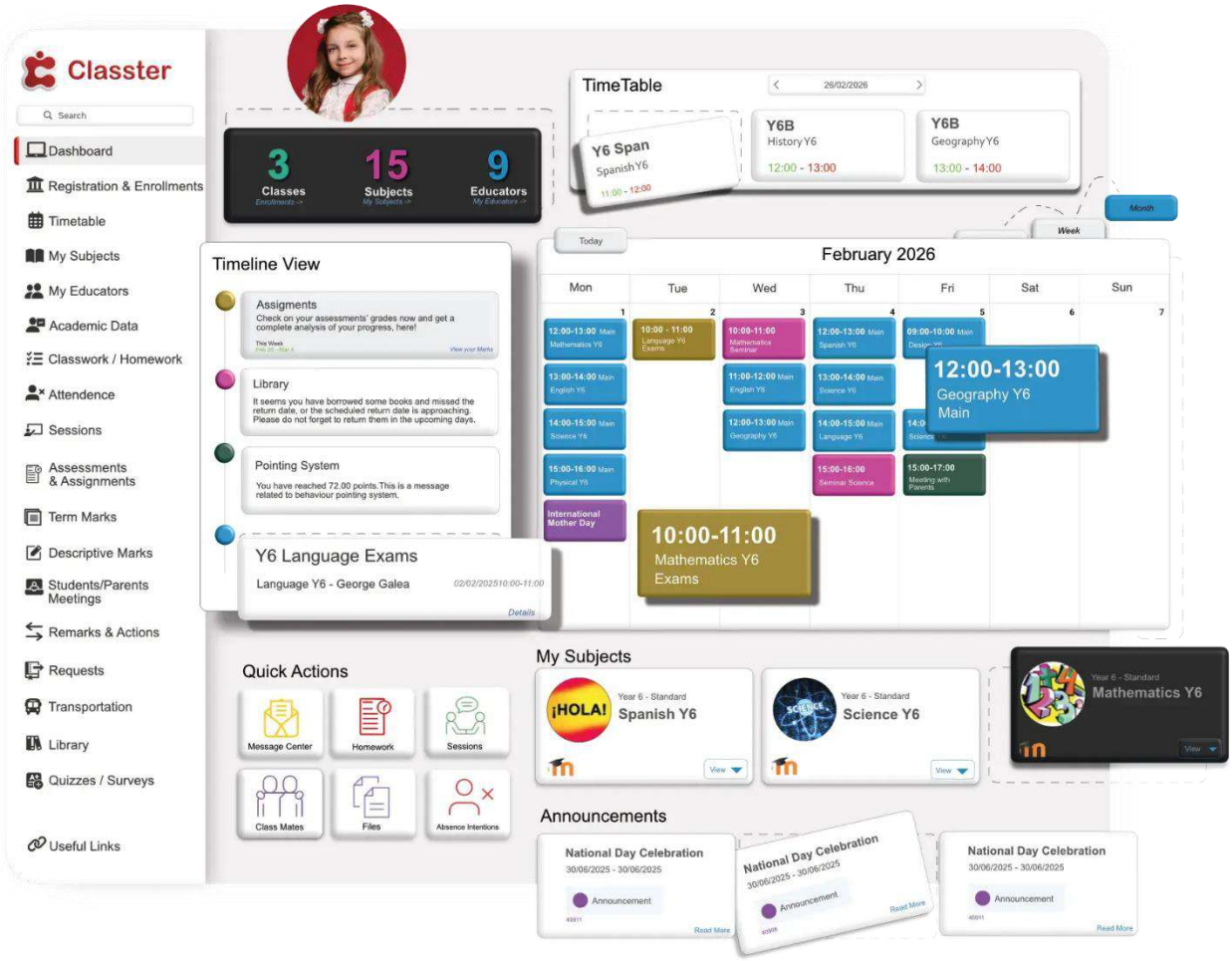

Рис. 1.2 Сайт Classter

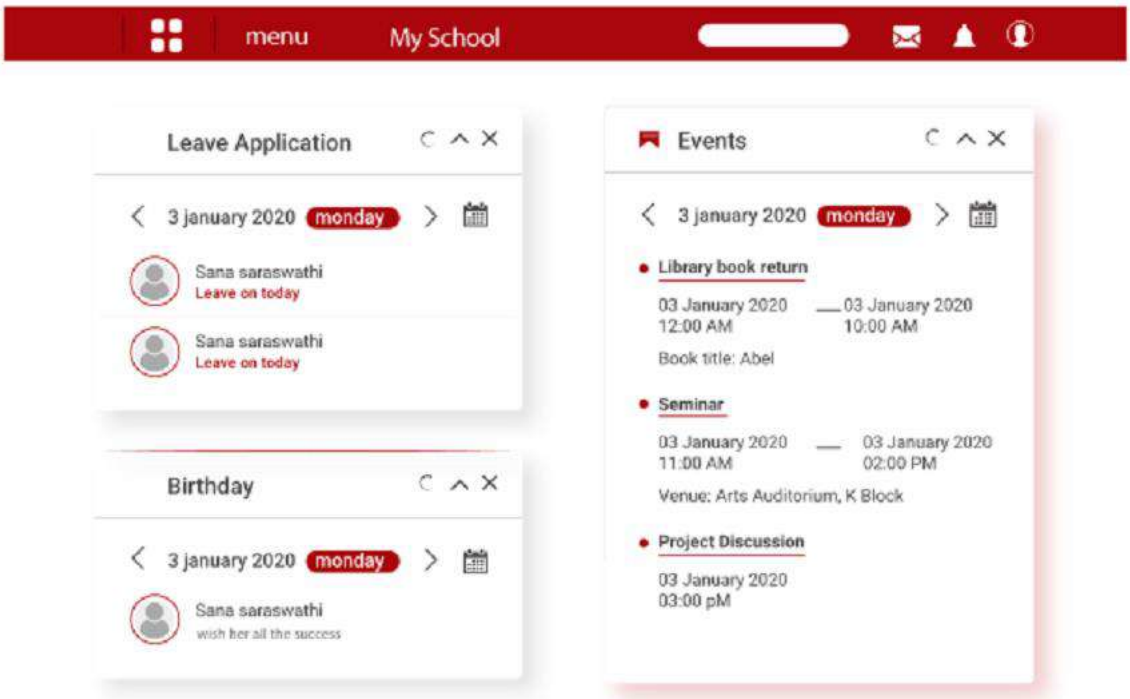

Рис. 1.3. Сайт Fedena

Варто також відзначити численні спеціалізовані розробки та наукові проєкти, які описуються у фахових публікаціях та кваліфікаційних роботах. Такі системи часто створюються з урахуванням унікальних потреб конкретного навчального закладу або навіть окремого факультету. Вони можуть використовувати найсучасніші технологічні стеки та пропонувати інноваційні підходи до збору даних (наприклад, розпізнавання облич, геолокація). Перевагою таких індивідуальних розробок є максимальна адаптація під специфіку замовника. Однак, вони можуть поступатися комерційним продуктам у надійності, рівні безпеки, масштабованості та якості документації, а також вимагають значних ресурсів на розробку, впровадження та подальшу підтримку.

Проведений аналіз існуючих рішень дозволяє констатувати, що, незважаючи на різноманітність пропозицій, існує об'єктивна потреба у створенні гнучкої, функціональної та водночас доступної системи контролю відвідуваності, яка б поєднувала переваги сучасних технологій та була б орієнтована на реальні потреби студентів і викладачів. Зокрема, важливим аспектом, якому не завжди приділяється належна увага в існуючих системах, є створення максимально простого та інтуїтивного інтерфейсу для швидкого доступу студентів до актуального розкладу, навчальних матеріалів, завдань та можливості миттєвого приєднання до онлайн-занять, а також надання викладачам зручних інструментів для фіксації присутності та оперативного аналізу даних без значних часових витрат.

На основі вищезазначеного аналізу та з урахуванням виявлених проблем зниження відвідуваності та зростання навантаження на учасників освітнього процесу, формулюються наступні ключові вимоги до проектованої системи:

Функціональні вимоги до системи повинні охоплювати повний цикл управління процесом контролю відвідуваності. По-перше, система має забезпечувати надійний механізм управління користувачами, включаючи їхню реєстрацію, автентифікацію та авторизацію з чітким розмежуванням прав доступу для різних ролей (адміністратор, викладач, студент). По-друге, необхідним є функціонал для управління основними довідниками навчального

процесу, такими як перелік навчальних груп, дисциплін (курсів) та формування детального розкладу занять із зазначенням всіх необхідних атрибутів (час, місце, викладач, тип заняття). По-третє, ядром системи має стати модуль безпосереднього контролю відвідуваності, що дозволяє викладачам фіксувати присутність студентів як через веб-інтерфейс, так і за допомогою спеціалізованого десктопного додатку, із збереженням відповідних статусів та часових міток. По-четверте, система повинна надавати гнучкі можливості для перегляду інформації та формування звітності: студенти повинні мати доступ до свого розкладу та персональної статистики відвідуваності; викладачі – до даних по своїх групах та дисциплінах; адміністрація – до зведеної аналітичної інформації. Важливою є також реалізація інтерфейсу прикладного програмування (API) для забезпечення взаємодії між серверною частиною та різними клієнтськими додатками.

Нефункціональні вимоги визначають якісні характеристики системи. До них належать, перш за все, висока продуктивність, що забезпечує швидкий відгук системи на дії користувачів та ефективну обробку запитів навіть за умов значного навантаження. Надійність системи повинна гарантувати безперебійну роботу ключових функцій та цілісність збережених даних. Критично важливою є безпека, що передбачає захист від несанкціонованого доступу, шифрування конфіденційної інформації (зокрема, паролів), розмежування прав доступу та захист від поширених веб-вразливостей. Система повинна бути масштабованою, тобто мати потенціал для розширення функціоналу та збільшення кількості користувачів без суттєвого погіршення продуктивності. Однією з пріоритетних нефункціональних вимог є зручність використання (usability), що передбачає розробку інтуїтивно зрозумілих, ергономічних та візуально привабливих інтерфейсів для всіх категорій користувачів. Нарешті, підтримуваність системи, що забезпечується якістю програмного коду, його структурованістю та наявністю документації, є важливою для подальшого розвитку та модифікації продукту.

Формулювання цих вимог слугує основою для подальшого обґрунтування вибору технологічного стеку та проектування архітектури програмного додатку, спрямованого на створення ефективного інструменту для вирішення актуальних проблем контролю відвідуваності в сучасному закладі вищої освіти.

1.3. Вибір та обґрунтування технологічного стеку для розробки

Вибір адекватного та ефективного стеку технологій є одним із визначальних факторів успішності розробки будь-якого програмного продукту, особливо такого комплексного, як система контролю відвідуваності, що передбачає наявність серверної логіки, інтерактивних клієнтських інтерфейсів та системи управління базами даних. Процес вибору інструментальних засобів для даної кваліфікаційної роботи ґрунтувався на ретельному аналізі сформульованих функціональних та нефункціональних вимог, врахуванні специфіки предметної області, а також на оцінці переваг та недоліків потенційних технологій-кандидатів. Основною метою було формування такого технологічного стеку, який би забезпечив оптимальний баланс між швидкістю розробки, продуктивністю, надійністю, масштабованістю та зручністю подальшої підтримки системи.

Серверна частина (Backend Application Programming Interface – API)**:** Для реалізації серверної частини, що відповідає за всю бізнес-логіку, обробку запитів від клієнтів та взаємодію з базою даних, було прийнято рішення використовувати мову програмування PHP у поєднанні з фреймворком Laravel. PHP є однією з найпоширеніших та найзатребуваніших мов для веб-розробки, що підтверджується її широким використанням у мільйонах веб-проєктів різного масштабу. Величезна спільнота розробників, велика кількість доступних бібліотек, пакетів та детальної документації значно спрощують та прискорюють процес розробки, а також полегшують пошук рішень для типових інженерних завдань. Альтернативними варіантами могли б бути Python з фреймворком Django або Ruby з Ruby on Rails, які також є потужними інструментами для

створення веб-додатків. Однак, PHP, завдяки своїй простоті розгортання на більшості хостинг-платформ та великій кількості готових рішень для інтеграції, часто є більш доступним вибором, особливо для проєктів з обмеженими ресурсами.

Вибір фреймворку Laravel останньої 12 версії був зумовлений його багатою функціональністю “з коробки”, елегантною архітектурою MVC (Model-View-Controller) та активною підтримкою спільнотою. Laravel надає розробнику потужні інструменти, такі як Eloquent ORM для зручної та безпечної роботи з базами даних, систему маршрутизації для визначення API ендпоінтів, вбудовані механізми автентифікації та авторизації, систему шаблонізації Blade (хоча для API-орієнтованого підходу вона використовується менше, але може бути корисною для адміністративних панелей), а також консольний інструмент Artisan для автоматизації рутинних завдань (генерація коду, управління міграціями тощо). У порівнянні з іншими PHP-фреймворками, такими як Symfony або CodeIgniter, Laravel часто вирізняється нижчим порогом входження та швидшим циклом розробки, що було важливим фактором для даної кваліфікаційної роботи. Забезпечення безпеки, зокрема захист від CSRF-атак та SQL-ін'єкцій, є однією з сильних сторін Laravel, що мінімізує ризики, пов'язані з веб-вразливостями.

Клієнтська веб-частина (Frontend Web Application):
Для розробки інтерактивного та сучасного користувацького інтерфейсу веб-додатку було обрано бібліотеку React у поєднанні з фреймворком Next.js. JavaScript, як мова програмування, є де-факто стандартом для фронтенд-розробки. React, розроблений компанією Facebook (Meta), є однією з найпопулярніших JavaScript-бібліотек для створення користувацьких інтерфейсів, що базується на компонентному підході. Такий підхід дозволяє розбивати складний інтерфейс на незалежні, повторно використовувані компоненти, що значно спрощує розробку, тестування та підтримку коду. Альтернативами React могли б бути Angular від Google або Vue.js. Angular є більш комплексним фреймворком з жорсткішою структурою, тоді як Vue.js часто

вважається легшим для вивчення. Однак, React має надзвичайно велику екосистему, величезну кількість готових UI-компонентів та бібліотек для управління станом, що робить його привабливим вибором для багатьох проєктів.

Фреймворк Next.js, побудований на базі React, надає низку суттєвих переваг для розробки сучасних веб-додатків. До них належать оптимізація для продуктивності та SEO (завдяки можливостям рендерингу на сервері – SSR, та генерації статичних сайтів – SSG), зручна маршрутизація на основі файлової системи, вбудовані механізми оптимізації зображень та коду. Використання Next.js дозволяє створювати швидкі, масштабовані та добре структуровані React-додатки. Для управління станом та взаємодії з API було обрано Redux Toolkit Query (RTK Query), що є частиною Redux Toolkit та надає потужні інструменти для декларативного отримання, кешування та синхронізації даних з сервером, значно спрощуючи роботу з асинхронними операціями. Для стилізації інтерфейсу було використано Tailwind CSS та UI-бібліотеку Rizzui, що дозволило швидко створювати адаптивні та візуально привабливі компоненти.

Десктопний додаток (Desktop Application):
Для реалізації десктопного клієнта було обрано мову програмування **C++**. Цей вибір зумовлений потребою у створенні нативного додатку, що має високу продуктивність та потенціал для тісної взаємодії з операційною системою, хоча в даному проєкті основний акцент робився на візуалізації даних та взаємодії з API. C++ є потужною мовою, що дозволяє контролювати ресурси системи та досягати високої швидкодії. Альтернативами для створення десктопних додатків могли б бути Java з фреймворками Swing або JavaFX, або C# з .NET (WPF, Windows Forms). Однак, C++ у поєднанні з легкою та гнучкою бібліотекою для побудови графічного інтерфейсу ImGui (Dear ImGui) надав можливість швидко розробити функціональний прототип з кастомним інтерфейсом без значних накладних витрат, характерних для більш важковагових GUI-фреймворків. ImGui, хоч і не є традиційним фреймворком для створення складних корпоративних додатків, ідеально підходить для інструментальних програм, панелей управління та додатків з інтенсивною візуалізацією даних, де важлива

швидкість розробки UI та можливість низькорівневого контролю над рендерингом. Для мережевої взаємодії з API було використано легку однофайлову бібліотеку httplib.h, що забезпечує реалізацію HTTP-клієнта, а для роботи з JSON-даними – бібліотеку nlohmann/json.hpp, відому своєю простотою використання та ефективністю.

Система управління базами даних (СУБД):
Як система управління базами даних для зберігання всієї інформації системи було обрано PostgreSQL. PostgreSQL є потужною, об'єктно-реляційною СУБД з відкритим кодом, що вирізняється високою надійністю, відповідністю стандартам SQL, розширюваністю та підтримкою складних запитів і типів даних, включаючи ефективну роботу з JSON/JSONB. У порівнянні з іншою популярною СУБД MySQL, PostgreSQL часто вважається більш придатною для складних додатків з великими обсягами даних та високими вимогами до цілісності та консистентності даних завдяки своїй строгій реалізації ACID-транзакцій та розширеним можливостям. Для даного проєкту, де передбачається зберігання значної кількості пов'язаних даних про студентів, розклад, відвідуваність та навчальні активності, надійність та гнучкість PostgreSQL є суттєвими перевагами. Фреймворк Laravel має відмінну вбудовану підтримку PostgreSQL через свій Eloquent ORM, що спрощує взаємодію з базою даних на рівні програмного коду.

Отже, обраний технологічний стек, що включає PHP/Laravel для серверної частини, Next.js/React для веб-клієнта, C++/ImGui для десктопного додатку та PostgreSQL для управління даними, є збалансованим та сучасним рішенням, яке дозволяє ефективно реалізувати всі заплановані функціональні можливості системи контролю відвідуваності та забезпечити її подальший розвиток. Кожен обраний інструмент має значну підтримку спільноти, велику кількість документації та доведену ефективність у реальних проєктах, що мінімізує ризики, пов'язані з розробкою.

Висновки до розділу 1

У рамках першого розділу даної кваліфікаційної бакалаврської роботи було проведено комплексне дослідження предметної області, пов'язаної з автоматизацією процесу контролю відвідуваності студентів у закладах вищої освіти. Результати цього дослідження дозволили сформулювати теоретико-методологічне підґрунтя для подальшого проектування та розробки спеціалізованого програмного додатку.

Насамперед, було детально проаналізовано та обґрунтовано актуальність обраної теми. Визначено, що проблема ефективного обліку та аналізу відвідуваності є нагальною для сучасних освітніх установ, оскільки традиційні паперові методи не відповідають сучасним вимогам до оперативності, точності та аналітичних можливостей. Особливу увагу було приділено виявленим тенденціям зниження рівня відвідуваності та зростання адміністративного навантаження на викладацький склад та студентів, що підкреслює необхідність впровадження інноваційних цифрових рішень. Автоматизація даного процесу розглядається як важливий крок на шляху до підвищення якості навчального процесу, оптимізації управлінських функцій та покращення загальної академічної дисципліни.

Проведений огляд існуючих програмних рішень та систем дозволив оцінити поточний стан ринку програмного забезпечення для контролю відвідуваності. Було розглянуто як інтегровані модулі у складі популярних систем управління навчанням (LMS), так і комплексні комерційні системи управління навчальним закладом (ERP), а також спеціалізовані індивідуальні розробки. Аналіз їхніх функціональних можливостей, переваг та недоліків уможливив виявлення незадоволених потреб користувачів та визначення потенційних напрямків для створення більш ефективного та користувацько-орієнтованого програмного продукту, зокрема з акцентом на зручність використання та інтеграцію різних клієнтських платформ.

На основі аналізу предметної області та існуючих рішень було сформульовано детальні функціональні та нефункціональні вимоги до проектованої системи. Функціональні вимоги охоплюють весь спектр необхідних операцій: від управління користувачами та довідниками навчального процесу до безпосередньої фіксації відвідуваності, перегляду інформації та формування звітності. Нефункціональні вимоги акцентують увагу на таких якісних характеристиках системи, як продуктивність, надійність, безпека, масштабованість, зручність використання та підтримуваність, що є критично важливими для забезпечення довгострокової ефективності та життєздатності програмного продукту.

Завершальним етапом першого розділу стало обґрунтування вибору інструментальних засобів та технологічного стеку для розробки. Було прийнято рішення про використання мови PHP з фреймворком Laravel для реалізації серверної частини (API), бібліотеки Next.js (React) для створення клієнтської веб-частини, мови C++ з бібліотекою ImGui для розробки десктопного додатку та системи управління базами даних PostgreSQL. Детально проаналізовано переваги кожної обраної технології в контексті поставлених завдань, проведено порівняння з можливими альтернативами та підкреслено їхню відповідність сформульованим вимогам щодо функціональності, продуктивності та безпеки.

Таким чином, у першому розділі закладено міцний фундамент для подальших етапів роботи. Визначено проблематику, обґрунтовано актуальність, сформульовано чіткі вимоги та обрано оптимальний набір технологій, що дозволяє перейти до безпосереднього проектування архітектури та бази даних розроблюваної системи контролю відвідуваності.

РОЗДІЛ 2
ПРОЕКТУВАННЯ СИСТЕМИ

2.1. Розробка архітектури програмного комплексу

Проектування архітектури програмного комплексу є фундаментальним етапом розробки, що визначає його структуру, взаємозв'язки між компонентами та загальну стратегію реалізації. Для системи контролю відвідуваності студентів, з огляду на її багатокомпонентність та необхідність забезпечення взаємодії між різними клієнтськими додатками та централізованим сховищем даних, було обрано класичну трирівневу архітектурну модель. Ця модель, що включає рівень представлення, рівень бізнес-логіки та рівень доступу до даних, дозволяє досягти чіткого розподілу відповідальностей, підвищити модульність системи, спростити її розробку, тестування та подальшу підтримку й масштабування. Такий підхід сприяє незалежності окремих рівнів, що уможливлює їхню паралельну розробку та модифікацію без значного впливу на інші частини системи.

Рівень представлення (Presentation Layer) відповідає за взаємодію з кінцевим користувачем, збір вхідних даних та візуалізацію результатів роботи системи. У розроблюваному програмному комплексі цей рівень реалізовано двома основними клієнтськими додатками. По-перше, це клієнтська веб-частина, розроблена на базі сучасного JavaScript-фреймворку Next.js, що використовує бібліотеку React для побудови динамічних та інтерактивних користувацьких інтерфейсів. Цей веб-додаток забезпечує кросплатформний доступ до функціоналу системи через стандартні веб-браузери, надаючи студентам, викладачам та адміністраторам зручні інструменти для перегляду розкладу, управління навчальними активностями, фіксації відвідуваності та аналізу даних. По-друге, розроблено спеціалізований десктопний додаток на мові програмування C++ з використанням бібліотеки ImGui для створення графічного інтерфейсу. Цей додаток призначений переважно для категорій користувачів, що

потребують розширених можливостей візуалізації даних, таких як викладачі та адміністративний персонал, або для роботи в умовах, де використання веб-інтерфейсу може бути менш оптимальним.

Рівень бізнес-логіки (Business Logic Layer / Application Layer) є центральним компонентом системи, де інкапсульовано всю основну логіку обробки даних, реалізацію бізнес-правил та управління взаємодією між рівнем представлення та рівнем даних. Цей рівень реалізовано на серверній стороні у вигляді Application Programming Interface (API), розробленого на мові програмування PHP з використанням потужного фреймворку Laravel. Серверна частина відповідає за автентифікацію та авторизацію користувачів, валідацію вхідних даних, виконання операцій з базою даних, обробку специфічних для предметної області алгоритмів та надання уніфікованого інтерфейсу для всіх клієнтських додатків. Взаємодія між клієнтами та сервером відбувається за протоколом HTTP з використанням формату JSON для обміну даними, що відповідає принципам RESTful архітектури.

Рівень доступу до даних (Data Access Layer) забезпечує зберігання, управління та вибірку всієї інформації, необхідної для функціонування системи. Як система управління базами даних (СУБД) для даного проєкту обрано PostgreSQL – об'єктно-реляційну СУБД з відкритим кодом, відому своєю надійністю, продуктивністю та відповідністю стандартам SQL. Взаємодія з базою даних на рівні бізнес-логіки здійснюється через об'єктно-реляційний мапер (ORM) Eloquent, що є складовою фреймворку Laravel. Eloquent ORM абстрагує роботу з SQL-запитами, дозволяючи розробникам взаємодіяти з даними на рівні об'єктних моделей, що значно спрощує та прискорює розробку, а також підвищує безпеку за рахунок захисту від SQL-ін'єкцій.

Детальніше розглянемо архітектурні особливості кожного з ключових компонентів.

Архітектура серверної частини (API на PHP Laravel), як видно зі структури директорій наданого проєкту (Рис. 2.1) чітко відповідає архітектурному шаблону Model-View-Controller (MVC), адаптованому для розробки API.

- Моделі (Models) розташовані в директорії app/Models (наприклад, User.php, Group.php, ClassModel.php, Schedule.php, Attendance.php та інші, що відповідають таблицям бази даних, описаним у підрозділі 2.2). Вони інкапсулюють логіку взаємодії з відповідними таблицями бази даних через Eloquent ORM, визначають зв'язки між сутностями (наприклад, hasMany, belongsTo, belongsToMany) та можуть містити специфічну для сутності бізнес-логіку.
- Контролери (Controllers) знаходяться в app/Http/Controllers. Вони відповідають за обробку вхідних HTTP-запитів, що надходять від клієнтських додатків. Контролери (наприклад, AuthController.php, AttendanceController.php, ScheduleController.php, ActivityController.php, FilesController.php) взаємодіють з моделями для отримання або збереження даних, застосовують необхідну логіку валідації (часто через Form Requests) та повертають відповідь клієнту, як правило, у форматі JSON.
- Маршрути (Routes) визначаються у файлах директорії routes (зокрема, routes/api.php для API-ендпоінтів, як видно з parts/api/activity.php, attendance.php, files.php у вашій структурі). Вони пов'язують конкретні URL-адреси та HTTP-методи (GET, POST, PUT, DELETE тощо) з відповідними методами контролерів, формуючи таким чином публічний інтерфейс API.
- Запити (Requests), розташовані в app/Http/Requests, використовуються для інкапсуляції логіки валідації вхідних даних та, можливо, авторизації перед тим, як запит досягне методу контролера. Це сприяє очищенню коду контролерів та централізації правил валідації.
- Проміжне програмне забезпечення (Middleware), що знаходиться в app/Http/Middleware (наприклад, Authenticate.php, TrimStrings.php), дозволяє виконувати певну логіку на етапі обробки запиту до або після його потрапляння до контролера. Middleware використовується для таких завдань, як автентифікація користувача, перевірка CSRF-токенів, логування запитів, модифікація заголовків тощо.

- Сервіси (Services), як видно з наявності директорії app/Services (наприклад, Search/Filter.php, HasFilters.php), можуть використовуватися для винесення складної або повторно використовуваної бізнес-логіки з контролерів, що сприяє кращій структуризації коду та його тестуванню.
- Перелічення (Enums), такі як app/Enums/UserRole.php, використовуються для визначення набору константних значень (наприклад, ролей користувачів), що підвищує читабельність та підтримуваність коду.

Застосування вищеописаної трирівневої архітектурної парадигми є ключовим для забезпечення належного рівня організації та керованості програмного комплексу. Розподіл системи на чітко визначені логічні рівні – представлення, бізнес-логіки та доступу до даних – дозволяє не лише спростити процес розробки та тестування окремих модулів, але й створює передумови для підвищення загальної надійності та гнучкості системи. Зокрема, рівень бізнес-логіки, реалізований у вигляді серверного API, стає центральною точкою обробки всієї функціональності, забезпечуючи уніфікований інтерфейс для різнорідних клієнтських додатків. Такий підхід мінімізує дублювання коду та полегшує внесення змін до бізнес-правил системи, оскільки модифікації на серверному рівні автоматично відображаються на поведінці всіх підключених клієнтів.

Обрана архітектура також сприяє кращій масштабованості системи. За потреби, окремі рівні або компоненти можуть бути оптимізовані або масштабовані незалежно один від одного.

Така структурована організація коду в Laravel-додатку, що базується на принципах MVC та додаткових патернах, дозволяє створювати добре організовані, тестовані та масштабовані API.

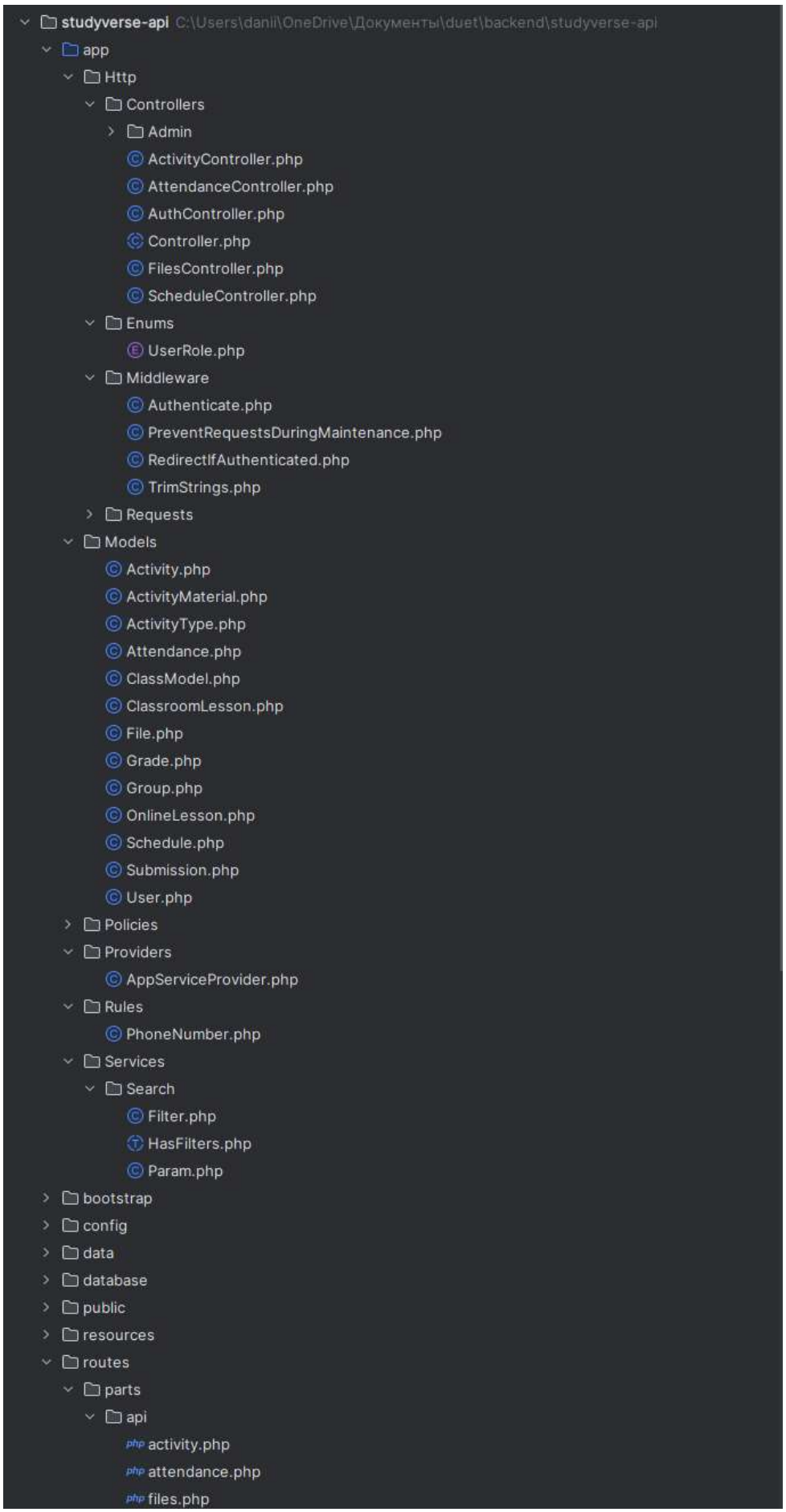

Рис. 2.1. Архітектура бекенду

2.2. Проектування інтерфейсу взаємодії (API) компонентів системи

Ефективна та надійна взаємодія між різними компонентами розробленого програмного комплексу, зокрема між серверною частиною та клієнтськими додатками (веб-інтерфейсом та десктопним клієнтом), є критично важливою для забезпечення цілісності функціонування системи. Ця взаємодія реалізується через ретельно спроектований інтерфейс прикладного програмування (Application Programming Interface – API), який надається серверною частиною, розробленою на базі фреймворку Laravel. Проектування API здійснювалося з дотриманням принципів RESTful архітектури, що передбачає використання стандартних HTTP-методів для виконання операцій над ресурсами, представленими у вигляді URL-адрес, та обмін даними у форматі JSON. Такий підхід забезпечує універсальність, простоту інтеграції та масштабованість взаємодії.

Ключовим аспектом проектування API було визначення набору ендпоінтів, що відповідають основним функціональним можливостям системи. Для управління автентифікацією користувачів було розроблено ендпоінти, відповідальні за вхід користувача в систему (/api/login), його реєстрацію (/api/register) та вихід (/api/logout). Процес автентифікації передбачає обмін обліковими даними та управління сесійними механізмами, включаючи роботу з CSRF-токенами (через стандартний ендпоінт Laravel /sanctum/csrf-cookie) для забезпечення безпеки запитів, що модифікують дані.

Для забезпечення функціоналу, пов'язаного безпосередньо з контролем відвідуваності, було спроектовано ендпоінти в рамках ресурсу AttendanceController. Зокрема, передбачено можливість отримання списку записів відвідуваності для автентифікованого користувача через GET-запит до /api/attendance, з підтримкою пагінації та можливістю застосування фільтрів для уточнення вибірки. Надсилання даних про відвідуваність, наприклад, фіксація факту приєднання студента до заняття, здійснюється через POST-запит до цього

ж ендпоінту /api/attendance, де тіло запиту містить необхідну інформацію, таку як ідентифікатор заняття та статус відвідування.

Управління розкладом занять реалізовано через ендпоінти, що обслуговуються ScheduleController. GET-запит до /api/schedules дозволяє клієнтським додаткам отримувати актуальну інформацію про заплановані заняття. Даний ендпоінт підтримує параметризацію для фільтрації розкладу за різними критеріями, наприклад, за ідентифікатором дисципліни або діапазоном дат, що є особливо важливим для коректного відображення календаря та списку занять на конкретний день у клієнтських інтерфейсах. Логіка серверної частини також враховує роль автентифікованого користувача, надаючи адміністраторам доступ до повного розкладу, тоді як студенти та викладачі отримують лише релевантну для них інформацію.

Аналогічним чином, для роботи з навчальними активностями (завданнями, матеріалами) передбачено ендпоінт /api/activity, що обробляється ActivityController. Цей інтерфейс дозволяє клієнтам отримувати структуровану інформацію про курси та пов'язані з ними навчальні активності, також з урахуванням прав доступу користувача. Для управління файлами, що можуть бути прикріплені до навчальних матеріалів чи зданих робіт, розроблено спеціалізований ендпоінт /api/files/{file_hash}/{download}/{filename}, який забезпечує безпечний доступ до файлових ресурсів.

Проектування структури запитів та відповідей для кожного ендпоінту здійснювалося з орієнтацією на мінімізацію надлишковості даних та забезпечення інформативності. Відповіді сервера, як правило, містять JSON-об'єкти з чітко визначеною структурою, що відповідає моделям даних, описаним у попередньому підрозділі. Валідація вхідних даних на стороні сервера, реалізована за допомогою Form Request класів Laravel, є невід'ємною частиною API, що гарантує отримання коректних даних та запобігає можливим помилкам обробки.

Особливу увагу при проектуванні API було приділено забезпеченню його консистентності та передбачуваності поведінки, що є важливим для розробки

надійних клієнтських додатків. Взаємодія десктопного C++ додатку з API передбачає коректну роботу з HTTP-заголовками, включаючи передачу CSRF-токенів та сесійних кук для підтримки автентифікованої сесії. Веб-клієнт на Next.js, використовуючи інструменти на кшталт Redux Toolkit Query, абстрагує безпосередню роботу з HTTP-запитами, надаючи розробнику більш високорівневий інтерфейс для взаємодії з серверними даними, проте ефективність цієї взаємодії напряму залежить від якості спроектованого API. Таким чином, розроблений інтерфейс прикладного програмування слугує надійним та стандартизованим мостом між потужною серверною логікою та різноманітними клієнтськими інтерфейсами системи контролю відвідуваності.

Висновки до розділу 2

У другому розділі було представлено комплексний підхід до проектування архітектури та ключових інтерфейсів взаємодії для розроблюваної системи контролю відвідуваності студентів. Результатом проведеної роботи стало формування міцного теоретичного та практичного фундаменту для подальшої реалізації програмних компонентів системи та розробки її інформаційного забезпечення.

Насамперед, було розроблено та обґрунтовано загальну архітектуру програмного комплексу, яка базується на класичній трирівневій моделі. Такий вибір дозволив забезпечити чітке розділення відповідальностей між рівнем представлення, що реалізується клієнтською веб-частиною на Next.js та десктопним додатком на C++, рівнем бізнес-логіки, представленим серверним API на PHP Laravel, та рівнем доступу до даних, що використовуватиме СУБД PostgreSQL. Цей архітектурний підхід сприяє підвищенню модульності, гнучкості та масштабованості системи, а також спрощує процеси її подальшої розробки, тестування та підтримки. Особливу увагу було приділено архітектурним особливостям серверної частини, побудованої згідно з патерном MVC та з використанням стандартних компонентів фреймворку Laravel, таких

як моделі, контролери, маршрути, Form Requests, middleware та сервіси, що забезпечують структуровану та логічно завершену обробку запитів (рис. 2.1).

Важливим аспектом проектних робіт стало детальне проектування інтерфейсу прикладного програмування (API), що слугує основним каналом взаємодії між серверною частиною та клієнтськими додатками. Дотримуючись принципів RESTful архітектури та використовуючи формат JSON для обміну даними, було визначено набір ключових ендпоінтів. Ці ендпоінти забезпечують весь спектр необхідного функціоналу, включаючи управління автентифікацією користувачів, доступ до даних про розклад занять, фіксацію та отримання інформації про відвідуваність, роботу з навчальними активностями та управління файловими ресурсами. При проектуванні API було ретельно опрацьовано структуру запитів та відповідей, а також механізми валідації вхідних даних, що є запорукою надійності та безпеки взаємодії між компонентами системи.

Таким чином, у другому розділі було виконано комплекс проектних робіт, що дозволили сформувати чітке бачення архітектурної побудови майбутньої системи та визначити фундаментальні принципи взаємодії її програмних компонентів. Обґрунтовані архітектурні рішення та детально спроектований API створюють міцну основу для переходу до наступних етапів, а саме – розробки структури бази даних та безпосередньої реалізації функціоналу системи.

РОЗДІЛ 3
РОЗРОБКА БАЗИ ДАНИХ

3.1. Проектування структури та логіки бази даних

Фундаментальною основою будь-якої інформаційної системи, що оперує значними обсягами взаємопов'язаних даних, є ретельно спроектована база даних. Для розроблюваного програмного комплексу контролю відвідуваності студентів, як система управління базами даних було обрано PostgreSQL. Вибір на користь цієї об'єктно-реляційної СУБД зумовлений її високою надійністю, відповідністю стандартам SQL, широкими можливостями розширюваності та ефективною підтримкою складних запитів і транзакцій, що є критично важливим для забезпечення цілісності та консистентності даних освітнього процесу. Уся структура бази даних була реалізована з використанням механізму міграцій фреймворку Laravel, що забезпечує версійність схеми, легкість її модифікації та розгортання на різних середовищах. Нижче наведено детальний опис ключових таблиць (сутностей), їх атрибутів та логічних зв'язків, що формують інформаційне ядро системи.

Таблиця users (Таблиця 3.1) призначена для зберігання вичерпної інформації про всіх зареєстрованих користувачів системи, включаючи студентів, викладачів та адміністративний персонал. Основним ідентифікатором користувача є поле id типу BIGSERIAL, що автоматично інкрементується. Поле group_id типу BIGINT встановлює зв'язок з таблицею groups та вказує на навчальну групу, до якої належить студент; для інших категорій користувачів це поле може мати значення NULL, а правило ON DELETE SET NULL забезпечує збереження запису про користувача у випадку видалення групи. Атрибути name (VARCHAR(255)) та email (VARCHAR(255) з унікальним обмеженням users_email_unique) зберігають відповідно повне ім'я та електронну адресу користувача, причому остання використовується як логін для входу в систему. Поле email_verified_at (TIMESTAMP) фіксує час підтвердження електронної

пошти, що є важливим елементом безпеки. Захешований пароль користувача зберігається в полі password (VARCHAR(255)). Атрибут remember_token (VARCHAR(100)) використовується для реалізації функціоналу “запам'ятати мене”, а стандартні поля created_at та updated_at (TIMESTAMP) фіксують час створення та останнього оновлення запису.

Таблиця 3.1

Структура таблиці users

Ім’я	Тип	Атрибути	Опис
id	bigserial	NOT_NULL	Ключ - ідентифікатор
group_id	bigint		Зовнішній ключ, посилання на id таблиці groups (ON DELETE SET NULL) - група студента
name	varchar(255)	NOT_NULL	Ім’я користувача
email	varchar(255)	NOT_NULL, UNIQUE	Електронна пошта користувача
email_verified_at	timestamp(0)		Дата підтвердження пошти
remember_token	varchar(100)		Тимчасовий токен
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Для створення таблиці users може бути використаний наступний SQL-запит:

CREATE TABLE users (

```
id BIGSERIAL PRIMARY KEY,
group_id BIGINT,
name VARCHAR(255) NOT NULL,
email VARCHAR(255) NOT NULL,
email_verified_at TIMESTAMP(0),
password VARCHAR(255) NOT NULL,
remember_token VARCHAR(100),
created_at TIMESTAMP(0),
updated_at TIMESTAMP(0),
CONSTRAINT users_email_unique UNIQUE (email),
CONSTRAINT users_group_id_foreign FOREIGN KEY (group_id)
REFERENCES groups ON DELETE SET NULL
);
```

Таблиця groups (табл. 3.2) слугує для ведення довідника навчальних груп, що існують в освітньому закладі. Унікальний ідентифікатор групи id типу BIGSERIAL є первинним ключем даної таблиці. Назва навчальної групи зберігається в полі title типу VARCHAR(50) і має обмеження унікальності groups_title_unique, що гарантує відсутність груп з однаковими назвами. Додаткова описова інформація про групу, наприклад, спеціалізація або рік набору, може міститися в текстовому полі description. Поля created_at та updated_at типу TIMESTAMP(0) мають стандартне призначення для відстеження часу створення та останньої модифікації запису про групу.

Таблиця 3.2

Структура таблиці навчальних груп “groups”

Ім’я	Тип	Атрибути	Опис
id	bigserial	NOT_NULL	Ключ - ідентифікатор
title	varchar(50)	NOT_NULL, UNIQUE	Назва групи
description	text		Опис групи

Продовження табл.3.2

1	2	3	4
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Для створення таблиці groups може бути використаний наступний SQL-запит:

```
CREATE TABLE groups (
   id BIGSERIAL PRIMARY KEY,
   title VARCHAR(50) NOT NULL,
   description TEXT,
   created_at TIMESTAMP(0),
   updated_at TIMESTAMP(0),
   CONSTRAINT groups_title_unique UNIQUE (title)
);
```

Таблиця classes (табл. 3.3) містить інформацію про навчальні дисципліни або курси, що викладаються в освітньому закладі. Поле id типу BIGSERIAL є унікальним ідентифікатором та первинним ключем дисципліни. Назва дисципліни представлена атрибутом title типу VARCHAR(100), який є обов'язковим для заповнення. Детальний опис курсу, його цілі, завдання або анотація можуть зберігатися у полі description типу TEXT. Передбачено можливість прив'язки файлу зображення до дисципліни (наприклад, обкладинка курсу) через поле image_file_id типу BIGINT, що є зовнішнім ключем, який посилається на id таблиці files. Правило ON DELETE SET NULL означає, що при видаленні відповідного файлу з таблиці files, значення image_file_id у таблиці classes буде встановлено в NULL, а не видалено сам запис про дисципліну. Для оптимізації запитів, що використовують поле image_file_id, створено індекс classes_image_file_id_index. Стандартні поля created_at та updated_at фіксують час створення та оновлення запису.

Таблиця 3.3

Структура таблиці навчальних дисциплін “classes”

Ім’я	Тип	Атрибути	Опис
id	bigserial	NOT_NULL	Ключ - ідентифікатор
title	varchar(50)	NOT_NULL, UNIQUE	Назва групи
image_file_id	bigint		Зовнішній ключ, посилання на id таблиці files (ON DELETE SET NULL) - файл зображення для дисципліни
description	text		Опис групи
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Для створення таблиці classes та відповідного індексу може бути використаний наступний SQL-запит:

```
CREATE TABLE classes (
    id BIGSERIAL PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    description TEXT,
    created_at TIMESTAMP(0),
    updated_at TIMESTAMP(0),
    image_file_id BIGINT,
    CONSTRAINT classes_image_file_id_foreign FOREIGN KEY (image_file_id)
        REFERENCES files (id) ON DELETE SET NULL
);
```

CREATE INDEX classes_image_file_id_index ON classes (image_file_id);

Таблиця classroom_lessons (табл. 3.4) призначена для зберігання специфічної інформації про заняття, що проводяться в аудиторному форматі. Первинним ключем є поле id типу BIGSERIAL. Обов'язкове поле room типу VARCHAR(50) містить номер або назву аудиторії, де проходить заняття. Поле building типу VARCHAR(100) дозволяє опціонально вказати назву навчального корпусу або будівлі. Стандартні поля created_at та updated_at фіксують часові мітки створення та модифікації запису. Дана таблиця використовується у поліморфному зв'язку з таблицею schedules для деталізації місця проведення офлайн-занять.

Таблиця 3.4

Структура таблиці аудиторних занять “classroom_lessons”

Ім’я	Тип	Атрибути	Опис
id	bigserial	NOT_NULL	Ключ - ідентифікатор
room	varchar(50)	NOT_NULL	Номер аудиторії
building	varchar(100)		Назва будівлі (корпусу)
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Для створення таблиці classroom_lessons може бути використаний наступний SQL-запит:

```
CREATE TABLE classroom_lessons (
    id BIGSERIAL PRIMARY KEY,
    room VARCHAR(50) NOT NULL,
    building VARCHAR(100),
    created_at TIMESTAMP(0),
    updated_at TIMESTAMP(0)
);
```

Таблиця online_lessons (табл. 3.5) зберігає інформацію, необхідну для підключення до занять, що проводяться у дистанційному (онлайн) форматі. Поле id типу BIGSERIAL є первинним ключем. Обов'язкове поле meeting_url типу VARCHAR(500) містить URL-адресу для доступу до віртуальної кімнати або конференції. Поля meeting_id (VARCHAR(100)) та passcode (VARCHAR(100)) є опціональними та можуть зберігати відповідно ідентифікатор зустрічі та пароль для підключення, якщо такі використовуються платформою для проведення онлайн-занять. Часові мітки created_at та updated_at відстежують час створення та оновлення запису. Ця таблиця також є частиною поліморфного зв'язку з таблицею schedules.

Таблиця 3.5

Структура таблиці онлайн-занять “online_lessons”

Ім’я	Тип	Атрибути	Опис
id	bigserial	NOT_NULL	Ключ - ідентифікатор
meeting_url	varchar(500)	NOT_NULL	Посилання на онлайн-зустріч (конференцію)
meeting_id	varchar(100)		Ідентифікатор онлайн-зустрічі
passcode	varchar(100)		Пароль для доступу до зустрічі
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Для створення таблиці online_lessons може бути використаний наступний SQL-запит:

```
CREATE TABLE online_lessons (
   id BIGSERIAL PRIMARY KEY,
```

```
    meeting_url VARCHAR(500) NOT NULL,
    meeting_id VARCHAR(100),
    passcode VARCHAR(100),
    created_at TIMESTAMP(0),
    updated_at TIMESTAMP(0)
);
```

Центральною таблицею для організації навчального процесу є schedules (табл. 3.6), що зберігає записи про заплановані заняття. Кожен запис має унікальний id (BIGSERIAL). Зовнішні ключі group_id (BIGINT, ON DELETE CASCADE) та class_id (BIGINT, ON DELETE RESTRICT) пов'язують заняття з відповідною групою та дисципліною. Поліморфний зв'язок, реалізований через поля lessonable_type (VARCHAR(255)) та lessonable_id (BIGINT), дозволяє асоціювати запис розкладу або з аудиторним заняттям (посилаючись на classroom_lessons), або з онлайн-заняттям (посилаючись на online_lessons). Час початку та закінчення заняття фіксується у полях started_at та ended_at (TIMESTAMP) відповідно. Для оптимізації запитів за поліморфним зв'язком створено індекси schedules_lessonable_type_lessonable_id_index та schedules_lessonable_id_lessonable_type_index.

Таблиця 3.6

Структура таблиці розкладу занять “schedules”

Ім’я	Тип	Атрибути	Опис
id	bigserial	NOT_NULL	Ключ - ідентифікатор
group_id	varchar(500)	NOT_NULL	Зовнішній ключ, посилання на id таблиці groups (ON DELETE CASCADE) - група, для якої заняття

Продовження табл. 3.6

1	2	3	4
class_id	varchar(100)	NOT_NULL	Зовнішній ключ, посилання на id таблиці classes (ON DELETE RESTRICT) - дисципліна
lessonable_type	varchar(100)	NOT_NULL	Тип пов'язаної моделі для поліморфного зв'язку
lessonable_id	timestamp(0)	NOT_NULL	Ідентифікатор пов'язаної моделі для поліморфного зв'язку
started_at	timestamp(0)	NOT_NULL	Час початку заняття
ended_at	timestamp(0)	NOT_NULL	Час закінчення заняття
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Для створення таблиці schedules та відповідних індексів може бути використаний наступний SQL-запит:

```
CREATE TABLE schedules (
    id BIGSERIAL PRIMARY KEY,
    group_id BIGINT NOT NULL,
    class_id BIGINT NOT NULL,
    lessonable_type VARCHAR(255) NOT NULL,
```

```
    lessonable_id BIGINT NOT NULL,
    started_at TIMESTAMP(0) NOT NULL,
    ended_at TIMESTAMP(0) NOT NULL,
    created_at TIMESTAMP(0),
    updated_at TIMESTAMP(0),
    CONSTRAINT schedules_group_id_foreign FOREIGN KEY (group_id)
      REFERENCES groups (id) ON DELETE CASCADE,
    CONSTRAINT schedules_class_id_foreign FOREIGN KEY (class_id)
      REFERENCES classes (id) ON DELETE RESTRICT
);
CREATE INDEX schedules_lessonable_type_lessonable_id_index
    ON schedules (lessonable_type, lessonable_id);
CREATE INDEX schedules_lessonable_id_lessonable_type_index
    ON schedules (lessonable_id, lessonable_type);
```

Управління навчальними активностями, такими як завдання, матеріали чи оголошення, реалізовано через декілька пов'язаних таблиць. Таблиця activity_types (табл. 3.7) є довідником типів активностей, де id (BIGSERIAL) – ідентифікатор, а type_name (VARCHAR(50), UNIQUE) – унікальна назва типу (наприклад, “Завдання”, “Матеріал”)

Таблиця 3.7

Структура таблиці типів навчальних активностей “activity_types”

Ім’я	Тип	Атрибути	Опис
id	bigserial	NOT_NULL, PRIMARY KEY	Ключ - ідентифікатор
type_name	varchar(50)	NOT_NULL, UNIQUE	Унікальна назва типу активності
description	text		Опис типу активності
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Для створення таблиці activity_types може бути використаний наступний SQL-запит:

```
CREATE TABLE activity_types (
   id BIGSERIAL PRIMARY KEY,
   type_name VARCHAR(50) NOT NULL,
   description TEXT,
   created_at TIMESTAMP(0),
   updated_at TIMESTAMP(0),
   CONSTRAINT activity_types_type_name_unique UNIQUE (type_name)
);
```

Таблиця activities (Табл. 3.8) зберігає детальну інформацію про кожну активність, включаючи її id (BIGSERIAL), зв'язки з дисципліною (class_id, ON DELETE RESTRICT), групою (group_id, ON DELETE CASCADE), типом активності (activity_type_id, ON DELETE RESTRICT) та користувачем, що її створив (created_by, ON DELETE RESTRICT). Також таблиця містить назву активності title (VARCHAR(200)), опис description (TEXT), термін виконання due_date (TIMESTAMP), максимальну кількість балів max_points (INTEGER), порядок сортування order (INTEGER, DEFAULT 0) та можливий зв'язок із завантаженим файлом через file_id (BIGINT, ON DELETE SET NULL).

Таблиця 3.8

Структура таблиці навчальних активностей “activities”

Ім’я	Тип	Атрибути	Опис
id	bigserial	NOT_NULL, PRIMARY KEY	Ключ - ідентифікатор
class_id	varchar(50)	NOT_NULL, UNIQUE	Зовнішній ключ, посилання на id таблиці classes

Продовження табл. 3.8

1	2	3	4
group_id	text	NOT_NULL	Зовнішній ключ, посилання на id таблиці groups
activity_type_id	bigint	NOT_NULL	Зовнішній ключ, посилання на id таблиці activity_types
created_by	bigint	NOT_NULL	Зовнішній ключ, посилання на id таблиці users
title	varchar(200)	NOT_NULL	Назва активності
description	text		Опис активності
due_date	timestamp(0)		Термін виконання (для завдань)
max_points	integer		Максимальна кількість балів
order	integer	DEFAULT 0, NOT_NULL	Порядок сортування активності
file_id	bigint		Зовнішній ключ, посилання на id таблиці files
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Для створення таблиці activities може бути використаний наступний SQL-запит:

CREATE TABLE activities (

```
    id BIGSERIAL PRIMARY KEY,
    class_id BIGINT NOT NULL,
    group_id BIGINT NOT NULL,
    activity_type_id BIGINT NOT NULL,
    created_by BIGINT NOT NULL,
    title VARCHAR(200) NOT NULL,
    description TEXT,
    due_date TIMESTAMP(0),
    max_points INTEGER,
    "order" INTEGER DEFAULT 0 NOT NULL,
    file_id BIGINT,
    created_at TIMESTAMP(0),
    updated_at TIMESTAMP(0),
    CONSTRAINT activities_class_id_foreign FOREIGN KEY (class_id)
        REFERENCES classes (id) ON DELETE RESTRICT,
    CONSTRAINT activities_group_id_foreign FOREIGN KEY (group_id)
        REFERENCES groups (id) ON DELETE CASCADE,
    CONSTRAINT activities_activity_type_id_foreign FOREIGN KEY
(activity_type_id)
        REFERENCES activity_types (id) ON DELETE RESTRICT,
    CONSTRAINT activities_created_by_foreign FOREIGN KEY (created_by)
        REFERENCES users (id) ON DELETE RESTRICT,
    CONSTRAINT activities_file_id_foreign FOREIGN KEY (file_id)
        REFERENCES files (id) ON DELETE SET NULL
);
```

Додаткові матеріали до активностей можуть зберігатися у таблиці activity_materials (табл. 3.9) яка пов'язана з activities через activity_id (ON DELETE CASCADE) та містить назву title, опис description та посилання на файл file_id.

Таблиця 3.9

Структура таблиці матеріалів до навчальних активностей “activity_materials”

Ім’я	Тип	Атрибути	Опис
id	bigserial	NOT_NULL, PRIMARY KEY	Ключ - ідентифікатор
activity_id	bigint	NOT_NULL, UNIQUE	Унікальна назва типу активності
title	varchar(200)	NOT_NULL	Опис типу активності
description	text		Опис матеріалу
file_id	bigint		Зовнішній ключ, посилання на id таблиці files
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Для створення таблиці activity_materials може бути використаний наступний SQL-запит:

```
CREATE TABLE activity_materials (
    id BIGSERIAL PRIMARY KEY,
    activity_id BIGINT NOT NULL,
    title VARCHAR(200) NOT NULL,
    description TEXT,
    file_id BIGINT,
    created_at TIMESTAMP(0),
    updated_at TIMESTAMP(0),
    CONSTRAINT activity_materials_activity_id_foreign FOREIGN KEY (activity_id)
```

```
    REFERENCES activities (id) ON DELETE CASCADE,
  CONSTRAINT activity_materials_file_id_foreign FOREIGN KEY (file_id)
    REFERENCES files (id) ON DELETE SET NULL
);
```

Для управління файлами, що завантажуються в систему, використовується таблиця files (Табл. 3.10). Вона зберігає метаінформацію про кожен файл, таку як id (BIGSERIAL), оригінальне ім'я original_name, розмір file_size, тип контенту content_type, шлях до файлу на сервері file_path, ім'я файлу на сервері file_name, розширення file_extension, загальний тип файлу file_type, унікальний хеш файлу file_hash (UNIQUE) для уникнення дублікатів та тип сховища storage_type.

Таблиця 3.10

Структура таблиці файлів “files”

Ім’я	Тип	Атрибути	Опис
id	bigserial	NOT_NULL, PRIMARY KEY	Ключ - ідентифікатор
original_name	varchar(255)	NOT_NULL	Оригінальна назва файлу
file_size	bigint	NOT_NULL	Розмір файлу в байтах
content_type	varchar(50)	NOT_NULL	MIME-тип файлу
file_path	varchar(500)	NOT_NULL	Шлях до файлу на сервері (без імені файлу)
file_name	varchar(255)	NOT_NULL	Ім'я файлу на сервері (згенероване)
file_extension	varchar(50)	NOT_NULL	Розширення файлу

Продовження табл. 3.10

1	2	3	4
file_type	varchar(50)	NOT_NULL	Загальний тип файлу (напр., 'document')
file_hash	varchar(255)	NOT_NULL, UNIQUE	Унікальний хеш вмісту файлу
storage_type	varchar(50)	NOT_NULL	Тип сховища (напр., 'local', 's3')
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Для створення таблиці files може бути використаний наступний SQL-запит:

```
CREATE TABLE files (
        id BIGSERIAL PRIMARY KEY,
        original_name VARCHAR(255) NOT NULL,
        file_size BIGINT NOT NULL,
        content_type VARCHAR(50) NOT NULL,
        file_path VARCHAR(500) NOT NULL,
        file_name VARCHAR(255) NOT NULL,
        file_extension VARCHAR(50) NOT NULL,
        file_type VARCHAR(50) NOT NULL,
        file_hash VARCHAR(255) NOT NULL,
        storage_type VARCHAR(50) NOT NULL,
        created_at TIMESTAMP(0),
        updated_at TIMESTAMP(0),
        CONSTRAINT files_file_hash_unique UNIQUE (file_hash)
);
```

Процес здачі студентами робіт по завданнях фіксується у таблиці submissions (Таблиця 3.11). Кожен запис пов'язаний з конкретною активністю (activity_id, ON DELETE CASCADE) та студентом (user_id, ON DELETE CASCADE), має унікальне обмеження на пару (activity_id, user_id). Таблиця також містить посилання на файл зданої роботи file_id (ON DELETE SET NULL), час здачі submitted_at (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP) та текстовий коментар студента content (TEXT).

Таблиця 3.11

Структура таблиці зданих робіт “submissions”

Ім’я	Тип	Атрибути	Опис
id	bigserial	NOT_NULL, PRIMARY KEY	Ключ - ідентифікатор
activity_id	bigint	NOT_NULL	Зовнішній ключ, посилання на id таблиці activities
user_id	bigint	NOT_NULL	Зовнішній ключ, посилання на id таблиці users
file_id	bigint		Зовнішній ключ, посилання на id таблиці files
submitted_at	timestamp(0)	DEFAULT CURRENT_TIMESTAMP, NOT_NULL	Час здачі роботи
content	text		Текстовий коментар

Продовження табл. 3.11

1	2	3	4
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Для створення таблиці submissions може бути використаний наступний SQL-запит:

```
CREATE TABLE submissions (
    id BIGSERIAL PRIMARY KEY,
    activity_id BIGINT NOT NULL,
    user_id BIGINT NOT NULL,
    file_id BIGINT,
    submitted_at TIMESTAMP(0) DEFAULT CURRENT_TIMESTAMP NOT NULL,
    content TEXT,
    created_at TIMESTAMP(0),
    updated_at TIMESTAMP(0),
    CONSTRAINT submissions_activity_id_foreign FOREIGN KEY (activity_id)
      REFERENCES activities (id) ON DELETE CASCADE,
    CONSTRAINT submissions_user_id_foreign FOREIGN KEY (user_id)
      REFERENCES users (id) ON DELETE CASCADE,
    CONSTRAINT submissions_file_id_foreign FOREIGN KEY (file_id)
      REFERENCES files (id) ON DELETE SET NULL,
    CONSTRAINT submissions_activity_id_user_id_unique UNIQUE (activity_id, user_id)
);
```

Оцінки за здані роботи зберігаються у таблиці grades (Таблиця 3.12), яка має унікальний зв'язок з таблицею submissions через submission_id (ON DELETE CASCADE, UNIQUE). Також фіксується ідентифікатор викладача, що поставив

оцінку (graded_by, ON DELETE RESTRICT), сама оцінка grade (INTEGER) та текстовий відгук feedback (TEXT).

Таблиця 3.12

Структура таблиці оцінок “grades”

Ім’я	Тип	Атрибути	Опис
id	bigserial	NOT_NULL, PRIMARY KEY	Ключ - ідентифікатор
submission_id	bigint	NOT_NULL	Зовнішній ключ, посилання на id таблиці submissions
graded_by	bigint	NOT_NULL	Зовнішній ключ, посилання на id таблиці users
grade	integer		Числова оцінка
feedback	text		Текстовий відгук викладача
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Для створення таблиці submissions може бути використаний наступний SQL-запит:

```
CREATE TABLE grades (
    id BIGSERIAL PRIMARY KEY,
    submission_id BIGINT NOT NULL,
    graded_by BIGINT NOT NULL,
    grade INTEGER,
    feedback TEXT,
    created_at TIMESTAMP(0),
    updated_at TIMESTAMP(0),
```

```
    CONSTRAINT grades_submission_id_unique UNIQUE (submission_id),
    CONSTRAINT grades_submission_id_foreign FOREIGN KEY (submission_id)
        REFERENCES submissions (id) ON DELETE CASCADE,
    CONSTRAINT grades_graded_by_foreign FOREIGN KEY (graded_by)
        REFERENCES users (id) ON DELETE RESTRICT
);
```

Зв'язки типу “багато-до-багатьох” реалізовані через проміжні таблиці. Таблиця class_group (Таблиця 3.13) пов'язує дисципліни (class_id, ON DELETE CASCADE) з навчальними групами (group_id, ON DELETE CASCADE), встановлюючи також період вивчення дисципліни групою через поля start_date (DATE) та end_date (DATE). Пара (class_id, group_id) є унікальною.

Таблиця 3.13

Структура таблиці зв'язку дисциплін та груп “class_group”

Ім’я	Тип	Атрибути	Опис
id	bigserial	NOT_NULL, PRIMARY KEY	Ключ - ідентифікатор
class_id	bigint	NOT_NULL	Зовнішній ключ, посилання на id таблиці classes
group_id	bigint	NOT_NULL	Зовнішній ключ, посилання на id таблиці groups
start_date	date	NOT_NULL	Дата початку вивчення дисципліни групою
end_date	date	NOT_NULL	Дата закінчення вивчення дисципліни групою

Продовження табл. 3.13

1	2	3	4
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Для створення таблиці class_group може бути використаний наступний SQL-запит:

```
CREATE TABLE class_group (
   id BIGSERIAL PRIMARY KEY,
   class_id BIGINT NOT NULL,
   group_id BIGINT NOT NULL,
   start_date DATE NOT NULL,
   end_date DATE NOT NULL,
   created_at TIMESTAMP(0),
   updated_at TIMESTAMP(0),
   CONSTRAINT class_group_class_id_foreign FOREIGN KEY (class_id)
      REFERENCES classes (id) ON DELETE CASCADE,
   CONSTRAINT class_group_group_id_foreign FOREIGN KEY (group_id)
      REFERENCES groups (id) ON DELETE CASCADE,
   CONSTRAINT class_group_class_id_group_id_unique UNIQUE (class_id, group_id)
);
```

Таблиця class_user (Таблиця 3.14) призначає користувачів (зазвичай викладачів) на певні дисципліни, встановлюючи їхню роль (наприклад, 'teacher') у полі role (VARCHAR(255), DEFAULT 'teacher'). Зв'язок здійснюється через class_id (ON DELETE CASCADE) та user_id (ON DELETE CASCADE), з унікальним обмеженням на цю пару та індексом по полю role.

Таблиця 3.14

Структура таблиці зв'язку користувачів та дисциплін “class_user”

Ім’я	Тип	Атрибути	Опис

Продовження табл. 3.14

1	2	3	4
id	bigserial	NOT_NULL, PRIMARY KEY	Ключ - ідентифікатор
class_id	bigint	NOT_NULL	Зовнішній ключ, посилання на id таблиці classes
user_id	bigint	NOT_NULL	Зовнішній ключ, посилання на id таблиці users
role	varchar(255)	DEFAULT 'teacher'::character varying, NOT_NULL	Роль користувача в рамках дисципліни (наприклад, 'teacher')
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Для створення таблиці class_user та відповідного індексу може бути використаний наступний SQL-запит:

```
CREATE TABLE class_user (
    id BIGSERIAL PRIMARY KEY,
    class_id BIGINT NOT NULL,
    user_id BIGINT NOT NULL,
    role VARCHAR(255) DEFAULT 'teacher'::character varying NOT NULL,
    created_at TIMESTAMP(0),
    updated_at TIMESTAMP(0),
    CONSTRAINT class_user_class_id_foreign FOREIGN KEY (class_id)
        REFERENCES classes (id) ON DELETE CASCADE,
    CONSTRAINT class_user_user_id_foreign FOREIGN KEY (user_id)
```

```
    REFERENCES users (id) ON DELETE CASCADE,
  CONSTRAINT class_user_class_id_user_id_unique UNIQUE (class_id, user_id)
);

CREATE INDEX class_user_role_index ON class_user (role);
```

Нарешті, ключова для даного проєкту таблиця attendances (Таблиця 3.15) фіксує факти відвідуваності. Кожен запис має унікальний id (BIGSERIAL) та пов'язує конкретне заняття з розкладу (schedule_id, ON DELETE CASCADE) зі студентом (user_id, ON DELETE CASCADE). Статус відвідування (наприклад, “join”, “absent”, “late”) зберігається у полі status (VARCHAR(255)), по якому створено індекс attendances_status_index для оптимізації вибірок.

Таблиця 3.15

Структура таблиці відвідуваності “attendances”

Ім’я	Тип	Атрибути	Опис
id	bigserial	NOT_NULL, PRIMARY KEY	Ключ - ідентифікатор
schedule_id	bigint	NOT_NULL	Зовнішній ключ, посилання на id таблиці schedules
user_id	bigint	NOT_NULL	Зовнішній ключ, посилання на id таблиці users
status	varchar(255)	NOT_NULL	Статус відвідування (наприклад, “join”, “absent”, “late”)
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Для створення таблиці attendances та відповідного індексу може бути використаний наступний SQL-запит:

```
CREATE TABLE attendances (
   id BIGSERIAL PRIMARY KEY,
   schedule_id BIGINT NOT NULL,
   user_id BIGINT NOT NULL,
   status VARCHAR(255) NOT NULL,
   created_at TIMESTAMP(0),
   updated_at TIMESTAMP(0),
   CONSTRAINT attendances_schedule_id_foreign FOREIGN KEY (schedule_id)
      REFERENCES schedules (id) ON DELETE CASCADE,
   CONSTRAINT attendances_user_id_foreign FOREIGN KEY (user_id)
      REFERENCES users (id) ON DELETE CASCADE
);

CREATE INDEX attendances_status_index ON attendances (status);
```

Окрім описаних таблиць, що безпосередньо стосуються бізнес-логіки додатку, система також включає низку службових таблиць, що використовуються фреймворком Laravel та іншими пакетами для забезпечення його функціонування. До них належать таблиці для управління ролями та дозволами (roles, permissions, model_has_permissions, model_has_roles, role_has_permissions), кешуванням (cache, cache_locks), обробкою черг завдань (jobs, failed_jobs, job_batches), управлінням міграціями бази даних (migrations), скиданням паролів (password_reset_tokens), API-токенами (personal_access_tokens) та сесіями користувачів (sessions). Ці таблиці, хоч і не є предметом детального опису в контексті бізнес-логіки, відіграють важливу роль у забезпеченні стабільної та безпечної роботи всього програмного комплексу.

3.2. Реляційні зв’язки між таблицями бази даних

Загальна логічна структура взаємозв'язків між усіма описаними сутностями наочно представлена на комплексній ER-діаграмі (Рис. 3.1). Ця візуалізація демонструє кардинальність встановлених зв'язків – один-до-одного, один-до-багатьох та багато-до-багатьох (реалізованих через проміжні таблиці) – а також ключові атрибути кожної сутності, що дозволяє отримати цілісне уявлення про спроектовану модель даних.

Зв'язки типу “один-до-багатьох” (One-to-Many) є найпоширенішими в даній схемі. Наприклад, таблиця groups (табл. 3.2) пов'язана з таблицею users (табл. 3.1) таким зв'язком: одна група може включати багато студентів, але кожен студент (якщо він належить до групи) належить лише до однієї групи. Це реалізовано через зовнішній ключ group_id у таблиці users (табл. 3.1), що посилається на id таблиці groups (табл. 3.2). Аналогічно, таблиця users (табл. 3.1) пов'язана з таблицею activities (табл. 3.9) через поле created_by (один користувач-викладач може створити багато активностей), з таблицею submissions (табл. 3.11) через поле user_id (один студент може здати багато робіт) та з таблицею grades (табл. 3.12) через поле graded_by (один викладач може виставити багато оцінок). Таблиця schedules (табл. 3.6) також має зв'язки "один-до-багатьох" з таблицею attendances (табл. 3.15) (одне заняття в розкладі може мати багато записів про відвідуваність) через зовнішній ключ schedule_id в таблиці attendances (табл. 3.15). Таблиця activities (табл. 3.9) має зв'язок "один-до-багатьох" з activity_materials (табл. 3.10) (одна активність може мати багато матеріалів) та з submissions (табл. 3.11) (одна активність може мати багато зданих робіт).

Зв'язки типу “багато-до-багатьох” (Many-to-Many) реалізовані за допомогою проміжних (зв'язуючих) таблиць. Так, взаємозв'язок між таблицями classes (табл. 3.3) та groups (табл. 3.2) встановлено через таблицю class_group (табл. 3.13). Одна дисципліна може викладатися для багатьох груп, і одна група може вивчати багато дисциплін. Проміжна таблиця class_group (табл. 3.13) містить зовнішні ключі class_id та group_id, а також додаткові атрибути, що характеризують цей зв'язок, наприклад, start_date та end_date періоду вивчення.

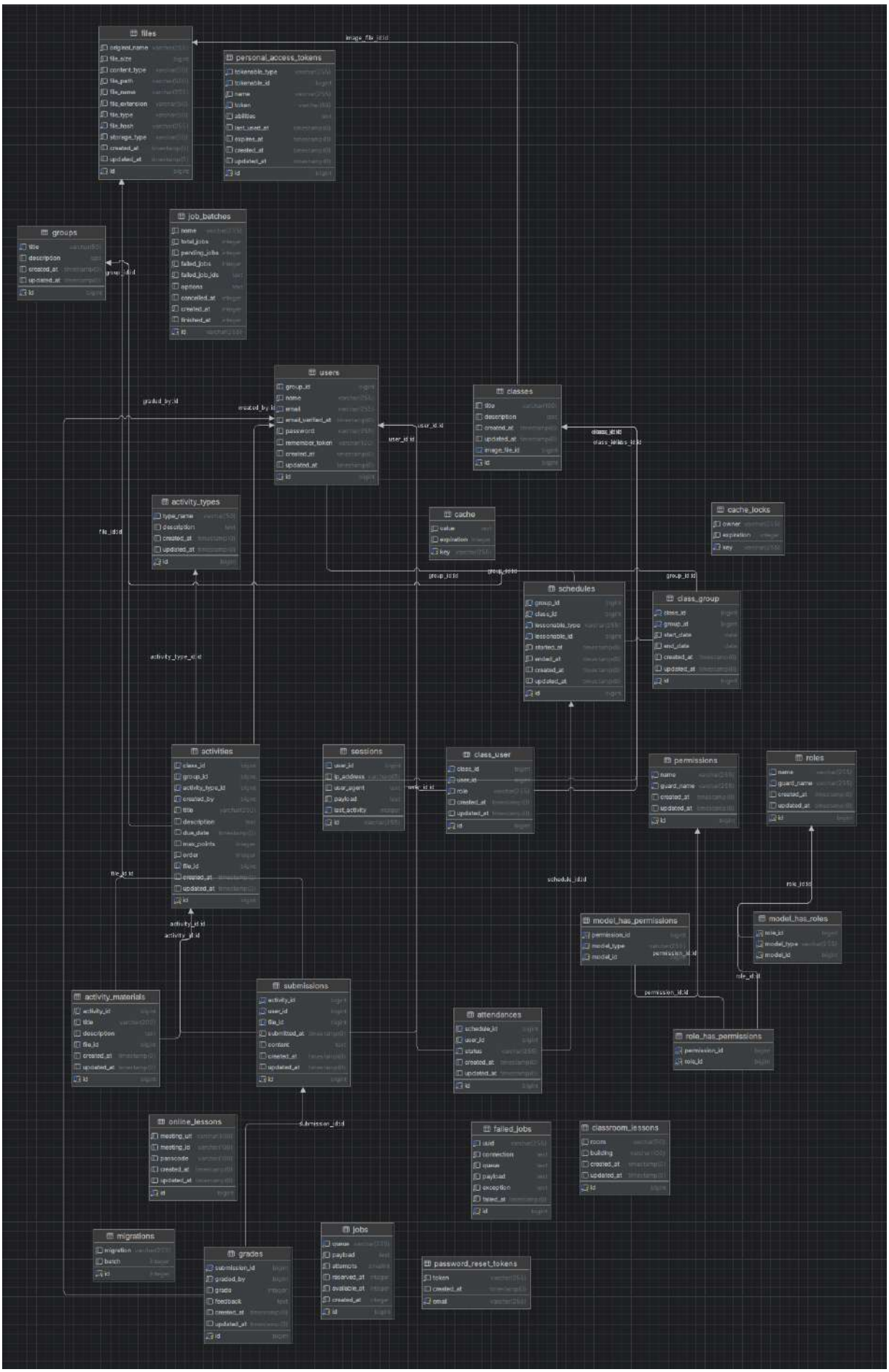

Рис. 3.1 Загальна ER-діаграма

Аналогічно, зв'язок між classes (табл. 3.3) та users (табл. 3.1) (для призначення викладачів на дисципліни) реалізовано через таблицю class_user (табл. 3.14), що містить class_id, user_id та атрибут role, який визначає роль користувача в рамках даної дисципліни. Система ролей та дозволів, що використовує таблиці roles, permissions, model_has_roles, model_has_permissions та role_has_permissions (опис яких буде наведено пізніше як службових), також ґрунтується на зв'язках "багато-до-багатьох" для гнучкого призначення дозволів ролям та ролей користувачам (моделям).

Особливий тип зв'язку – поліморфний зв'язок (Polymorphic Relationship) – використано в таблиці schedules (табл. 3.6) для полів lessonable_type та lessonable_id. Це дозволяє одному запису в розкладі (schedules) бути пов'язаним з різними типами “уроків” – або з записом у таблиці classroom_lessons (табл. 3.4) (для аудиторних занять), або з записом у таблиці online_lessons (табл. 3.5) (для дистанційних занять). Поле lessonable_type зберігає назву моделі, з якою встановлено зв'язок, а lessonable_id – ідентифікатор відповідного запису в цій моделі. Такий підхід забезпечує гнучкість структури даних та дозволяє легко додавати нові типи “уроків” у майбутньому без зміни основної таблиці розкладу.

Зв'язки типу “один-до-одного” (One-to-One) також присутні в схемі, хоча й менш поширені. Наприклад, зв'язок між таблицями submissions (табл. 3.11) (здані роботи) та grades (табл. 3.12) (оцінки) через унікальний зовнішній ключ submission_id у таблиці grades (табл. 3.12) фактично встановлює зв'язок “один-до-одного”, оскільки одна здана робота може мати лише одну підсумкову оцінку.

Для забезпечення цілісності даних при визначенні зовнішніх ключів використовуються правила каскадних операцій, такі як ON DELETE CASCADE (наприклад, при видаленні запису з groups (табл. 3.2) автоматично видаляються пов'язані записи з schedules (табл. 3.6) або class_group (табл. 3.13)) або ON DELETE SET NULL (наприклад, при видаленні файлу з files (табл. 3.8), відповідне поле image_file_id у classes (табл. 3.3) встановлюється в NULL). Вибір конкретного правила залежить від бізнес-логіки та вимог до збереження даних. Обмеження унікальності (UNIQUE constraints) застосовуються до полів або

комбінацій полів, де значення не повинні повторюватися (наприклад, email у users (табл. 3.1), title у groups (табл. 3.2), пара (activity_id, user_id) у submissions (табл. 3.11)).

Слід констатувати, що розроблена структура бази даних є достатньо гнучкою, комплексною та добре структурованою для ефективного зберігання, управління та вибірки всієї інформації, необхідної для реалізації функціоналу системи контролю відвідуваності та супутніх навчальних процесів, забезпечуючи при цьому цілісність та консистентність даних.

Висновки до розділу 3

У третьому розділі було представлено комплексний підхід до проектування та розробки інформаційного забезпечення для розроблюваної системи контролю відвідуваності студентів, що полягало у створенні детальної структури бази даних. Результатом проведеної роботи стало формування міцного фундаменту для надійного зберігання, ефективного управління та забезпечення цілісності всіх даних, необхідних для функціонування програмного комплексу.

Проектування бази даних на платформі PostgreSQL було здійснено з урахуванням всіх ключових сутностей предметної області. Для кожної з основних таблиць, таких як користувачі (users, табл. 3.1), навчальні групи (groups, табл. 3.2), дисципліни (classes, табл. 3.3), розклад занять (schedules, табл. 3.6), відвідуваність (attendances, табл. 3.15), а також для супутніх таблиць, що деталізують навчальні активності, файлові ресурси та специфіку проведення занять, було ретельно визначено їх призначення, розроблено перелік атрибутів із зазначенням відповідних типів даних та необхідних обмежень. Практична реалізація спроектованої схеми була продемонстрована через наведення SQL-коду для створення кожної таблиці. Важливим аспектом є те, що вся структура бази даних була реалізована з використанням механізму міграцій фреймворку

Laravel, що забезпечує контроль версій, легкість модифікації та портативність схеми даних на різні середовища розгортання.

Значну увагу було приділено аналізу та опису реляційних зв'язків між таблицями спроектованої бази даних. Було розглянуто та реалізовано основні типи зв'язків, що використовуються в системі для відображення логічних залежностей між даними, а саме: зв'язки типу "один-до-багатьох", "багато-до-багатьох" (з використанням проміжних асоціативних таблиць, таких як class_group (табл. 3.13) та class_user (табл. 3.14)), а також поліморфні зв'язки, зокрема, в таблиці schedules (табл. 3.6) для забезпечення гнучкості моделі. Детально описано застосування зовнішніх ключів, правил посилальної цілісності, включаючи каскадні операції (ON DELETE CASCADE, ON DELETE SET NULL), та обмежень унікальності, що є критично важливим для забезпечення цілісності та консистентності даних. Загальна логічна структура та встановлені взаємозв'язки між усіма сутностями бази даних були візуалізовані за допомогою комплексної ER-діаграми (рис. 3.16), яка наочно демонструє кардинальність зв'язків та ключові атрибути, дозволяючи отримати цілісне уявлення про розроблену модель даних.

Таким чином, у третьому розділі було всебічно описано процес розробки бази даних, починаючи від вибору системи управління базами даних та закінчуючи деталізацією структури таблиць та їхніх взаємозв'язків. Створена модель даних характеризується комплексністю, гнучкістю та логічною структурованістю, що забезпечує надійне підґрунтя для зберігання та ефективного управління всією інформацією, необхідною для функціонування системи контролю відвідуваності та пов'язаних з нею навчальних процесів. Це, у свою чергу, є необхідною передумовою для успішної реалізації програмних компонентів, детальний опис яких представлено у наступному розділі даної кваліфікаційної роботи.

РОЗДІЛ 4
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Розробка серверної частини (PHP Laravel)

Серверна частина програмного комплексу, що є його ядром та відповідає за обробку бізнес-логіки, управління даними та надання інтерфейсу прикладного програмування (API) для взаємодії з клієнтськими додатками, була реалізована з використанням мови програмування PHP та сучасного фреймворку Laravel. Вибір даного фреймворку, як було обґрунтовано у попередніх розділах, зумовлений його багатою екосистемою, наявністю потужних інструментів для швидкої та ефективної розробки, а також вбудованими механізмами для забезпечення безпеки та надійності додатку. Архітектура серверної частини базується на патерні Model-View-Controller (MVC), що забезпечує чітке розділення відповідальностей між компонентами системи.

Процес розробки серверної частини охоплював декілька ключових етапів, включаючи визначення маршрутизації, реалізацію контролерів для обробки запитів, створення моделей для взаємодії з базою даних PostgreSQL за допомогою Eloquent ORM, а також розробку механізмів автентифікації, авторизації та валідації даних.

Організація маршрутизації та визначення API ендпоінтів було здійснено у файлах директорії routes. Зокрема, основний файл маршрутизації API routes/api.php (як показано на рис. [номер скріншоту файлу api.php]) визначає публічні ендпоінти системи. Наприклад, для автентифікації користувачів визначено маршрути для входу (POST /login), реєстрації (POST /register) та виходу (POST /logout), які пов'язані з відповідними методами AuthController. Для забезпечення модульності та кращої організації, частина API маршрутів, що стосуються специфічних ресурсів, таких як активності, відвідуваність, файли та розклад, винесена в окремі файли у директорії routes/parts/api (наприклад, activity.php, attendance.php, files.php, schedules.php), які потім підключаються до

основного файлу api.php за допомогою конструкції require __DIR__.'/parts/api.php';. Такий підхід сприяє кращій читабельності та підтримуваності коду маршрутизації. Захист маршрутів, що вимагають автентифікації, реалізовано за допомогою middleware, наприклад, 'auth:sanctum' для ендпоінту /user, що повертає дані поточного автентифікованого користувача.

Реалізація контролерів (розташованих у app/Http/Controllers) є центральним елементом обробки запитів. Кожен контролер відповідає за певну групу функціональних можливостей. Наприклад, AuthController інкапсулює логіку автентифікації та реєстрації користувачів. Метод login даного контролера, після успішної валідації облікових даних за допомогою LoginRequest (який включає логіку перевірки спроб входу та захисту від перебору паролів – ensureIsNotRateLimited), регенерує сесію користувача та повертає JSON-відповідь з основними даними користувача. Метод logout забезпечує завершення сесії користувача та інвалідацію токена. Метод register створює нового користувача та ініціює відправку листа для верифікації електронної пошти.

Контролер AttendanceController відповідає за управління даними про відвідуваність. Його метод store приймає запит StoreAttendanceRequest, що валідує вхідні дані (schedule_id, status), та створює або оновлює запис про відвідуваність для поточного користувача, використовуючи метод Eloquent firstOrCreate для уникнення дублікатів. Метод index контролера AttendanceController використовує IndexAttendanceRequest для обробки запитів на отримання списку відвідувань, застосовуючи фільтри (через трейт HasFilters, що, ймовірно, використовується у IndexAttendanceRequest) та пагінацію результатів.

Аналогічно, ScheduleController обробляє запити, пов'язані з розкладом. Метод index цього контролера демонструє реалізацію розмежування доступу на основі ролі користувача (використовуючи перелічення UserRole та метод hasRole моделі User): адміністратори отримують повний доступ до розкладу, тоді як інші користувачі – лише до релевантної для них інформації. Запити також підтримують фільтрацію, наприклад, за class_id, та пагінацію. Для оптимізації

запитів активно використовується “жадібне завантаження” пов'язаних моделей за допомогою методу with(), наприклад, with(['lessonable', 'classModel.assignedUsers', 'group', 'attendances.user']). Це дозволяє уникнути проблеми “N+1 запит” та підвищити продуктивність API.

Контролер ActivityController надає доступ до навчальних активностей, також з урахуванням ролі користувача та завантаженням пов'язаних даних (матеріали, здані роботи). FilesController забезпечує безпечний доступ до завантажених файлів, обробляючи запити на їх відображення або завантаження з коректними HTTP-заголовками.

Розробка моделей Eloquent ORM (розташованих у app/Models) відіграє ключову роль у взаємодії з базою даних PostgreSQL. Кожна модель (наприклад, User.php, Schedule.php, Attendance.php, Group.php, ClassModel.php (для таблиці classes), Activity.php тощо) відповідає певній таблиці бази даних та надає об'єктно-орієнтований інтерфейс для виконання CRUD-операцій та визначення зв'язків між сутностями. Наприклад, модель User містить визначення зв'язків group() (BelongsTo), assignedClasses() (BelongsToMany), submissions() (HasMany), attendances() (HasMany), а також методи для перевірки ролей (isAdmin, isTeacher, isStudent). Модель Schedule визначає зв'язки group() (BelongsTo), classModel() (BelongsTo), lessonable() (MorphTo) та attendances() (HasMany), а також налаштування для автоматичного перетворення атрибутів (casts), наприклад, полів started_at та ended_at у тип datetime. Використання атрибутів $fillable та $hidden у моделях забезпечує контроль над масовим присвоєнням та серіалізацією даних.

Забезпечення безпеки та валідації даних є невід'ємною частиною розробки серверної частини. Валідація вхідних даних реалізована за допомогою спеціалізованих класів Form Requests (наприклад, LoginRequest, StoreAttendanceRequest, IndexAttendanceRequest, IndexScheduleRequest), які розташовані в директорії app/Http/Requests. Ці класи містять набір правил валідації для відповідних полів запиту та можуть інкапсулювати логіку авторизації для виконання конкретного запиту. Наприклад, LoginRequest не

лише валідує поля email та password, але й реалізує логіку автентифікації та захисту від перебору паролів за допомогою RateLimiter.

Механізми автентифікації та авторизації базуються на вбудованих можливостях Laravel. Для автентифікації, ймовірно, використовується Laravel Sanctum для API-токенів або стандартні веб-сесії, що підтверджується використанням Auth::guard('web')->logout() та захистом маршрутів через auth:sanctum. Розмежування прав доступу реалізовано на основі ролей, визначених у переліченні UserRole та, ймовірно, з використанням пакету spatie/laravel-permission (про що свідчить трейт HasRoles у моделі User).

На завершення етапу розробки серверної частини було проведено її функціональне тестування. Хоча написання автоматизованих юніт-тестів не входило до завдань даної кваліфікаційної роботи, перевірка працездатності API ендпоінтів здійснювалася опосередковано через взаємодію з ними розроблених клієнтських додатків. Коректність обробки запитів, валідації вхідних даних, взаємодії з базою даних PostgreSQL та формування JSON-відповідей ретельно перевірялася шляхом аналізу поведінки веб- та десктопного клієнтів. Додатково, для моніторингу та відладки HTTP-трафіку та аналізу логів сервера використовувалися відповідні інструментальні засоби. У ході тестування було перевірено сценарії автентифікації, отримання та модифікації даних розкладу, фіксації відвідуваності та роботи з навчальними активностями, що дозволило переконатися у відповідності реалізованого API проектним вимогам.

Отже, розробка серверної частини на PHP Laravel дозволила створити надійний, безпечний та добре структурований API, що забезпечує всю необхідну функціональність для системи контролю відвідуваності, ефективно взаємодіє з базою даних PostgreSQL та надає уніфікований інтерфейс для клієнтських додатків. Використання патерну MVC, можливостей Eloquent ORM, системи маршрутизації, middleware та Form Requests сприяло створенню чистого, підтримуваного та масштабованого коду.

4.2. Розробка інтерфейсу користувача сайту (Frontend на Next.js)

Клієнтська веб-частина програмного комплексу контролю відвідуваності, розроблена з використанням фреймворку Next.js та бібліотеки React, відіграє ключову роль у забезпеченні зручного та інтерактивного доступу користувачів до функціоналу системи через стандартні веб-браузери. Головними завданнями при її створенні були реалізація інтуїтивно зрозумілого інтерфейсу, забезпечення швидкого відгуку на дії користувача, ефективна взаємодія з серверним API та надання персоніфікованого досвіду для різних категорій користувачів, зокрема студентів та викладацького складу. Архітектура веб-додатку базується на компонентному підході, що є характерним для React, та використовує сучасні інструменти для управління станом та маршрутизації.

Основою для взаємодії з серверною частиною слугує набір спеціалізованих API-клієнтів, розроблених з використанням Redux Toolkit Query (RTK Query). Зокрема, було створено окремі модулі, такі як authApi.tsx, classroomApi.tsx, scheduleApi.tsx та attendanceApi.tsx, кожен з яких інкапсулює логіку запитів до відповідних ендпоінтів API Laravel. RTK Query дозволяє декларативно описувати операції отримання та модифікації даних, автоматично обробляючи стани завантаження, помилок, а також забезпечуючи ефективне кешування та автоматичне оновлення даних при їх зміні на сервері. Для виконання безпосередніх HTTP-запитів використовується кастомний axiosBaseQuery.

Ключовим елементом інтерфейсу є головна інформаційна панель (Dashboard), представлена компонентом AppointmentDashboard. Як видно з рис. 4.1, головна панель веб-додатку агрегує ключові інформаційні блоки: “My Classes” (список предметів), “Schedule Calendar” (календар та розклад на обрану дату) та “Recent Activities” (останні активності користувача). Це дозволяє користувачам швидко отримувати доступ до необхідної інформації.

Компонент ClassroomSection (My Classes) (Рис. 4.1) відповідає за відображення списку навчальних курсів. Дані про курси отримуються асинхронно. Кожен курс представлений інтерактивною карткою з назвою,

описом та кількістю активностей. На рис. 4.2 продемонстровано інтерфейс перегляду активностей в рамках обраного курсу, що включає оголошення, матеріали та завдання, з можливістю їх розгортання для отримання детальної інформації.

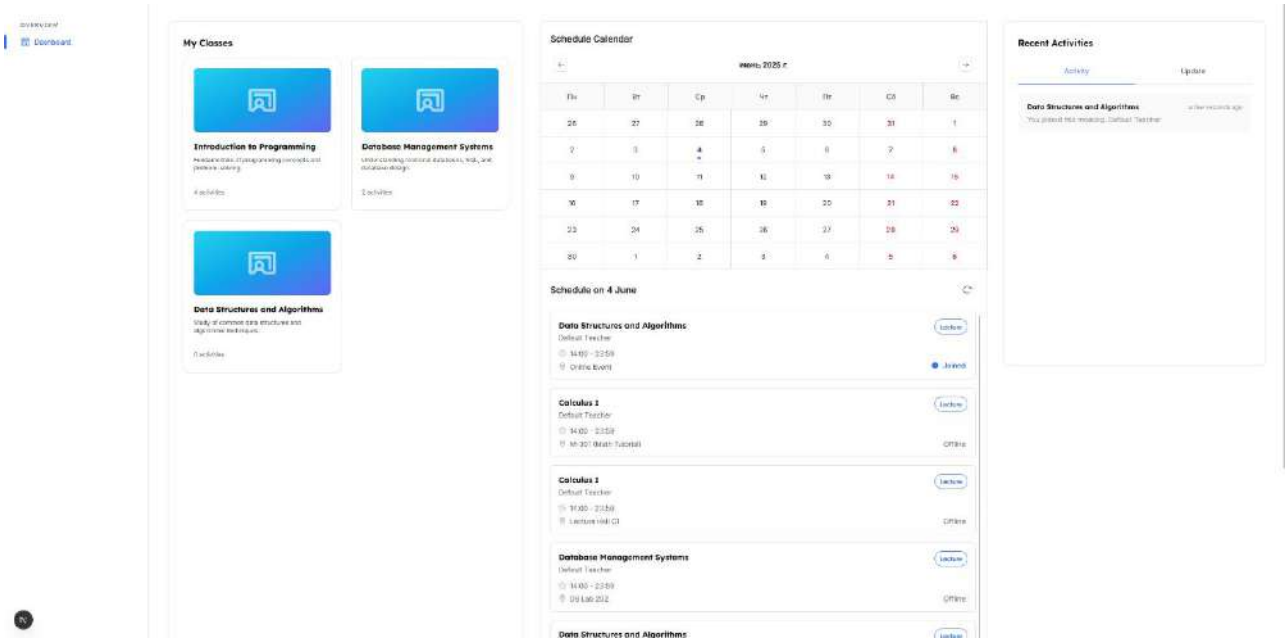

Рис. 4.1 Головна сторінка веб сайту

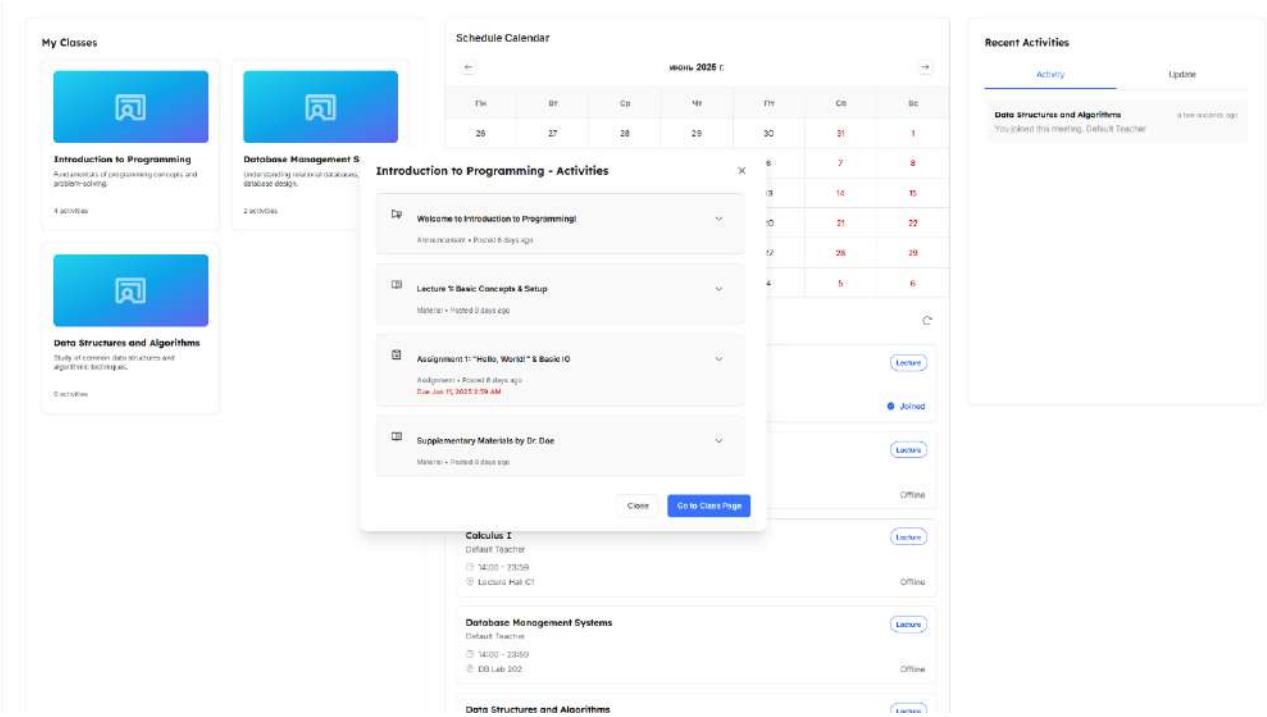

Рис. 4.2. Швидкий перегляд активностей

Компонент ScheduleList (Schedule Calendar) є центральним елементом для роботи з розкладом занять (див. рис. 4.1) Він інтегрує інтерактивний календар

для вибору дати та динамічно відображає список занять на обраний день. Кожне заняття відображається окремою карткою з детальною інформацією. Для онлайн-занять передбачена кнопка “Join”, що фіксує факт приєднання студента на сервері та, за наявності посилання, відкриває онлайн-зустріч.

Компонент RecentActivities (Recent Activities) відображає список останніх дій користувача, зокрема, приєднання до онлайн-занять. Управління автентифікацією реалізовано з використанням next-auth/react. Маршрутизація побудована на основі файлової системи Next.js. Для стилізації використовуються UI-бібліотека Rizzui та Tailwind CSS.

Інтерфейс перегляду стрічки активностей предмету, приклад якого наведено на (Рис. 4.3), слугує централізованим інформаційним хабом для студентів та викладачів, забезпечуючи структурований доступ до всіх актуальних оновлень, навчальних матеріалів, завдань та оголошень, що стосуються даної дисципліни. Такий підхід, подібний до концепції "стрічки новин" або "дошки оголошень" у соціальних мережах та системах управління навчанням, значно спрощує навігацію та пошук необхідної інформації, мінімізуючи ризик пропустити важливі повідомлення або терміни.

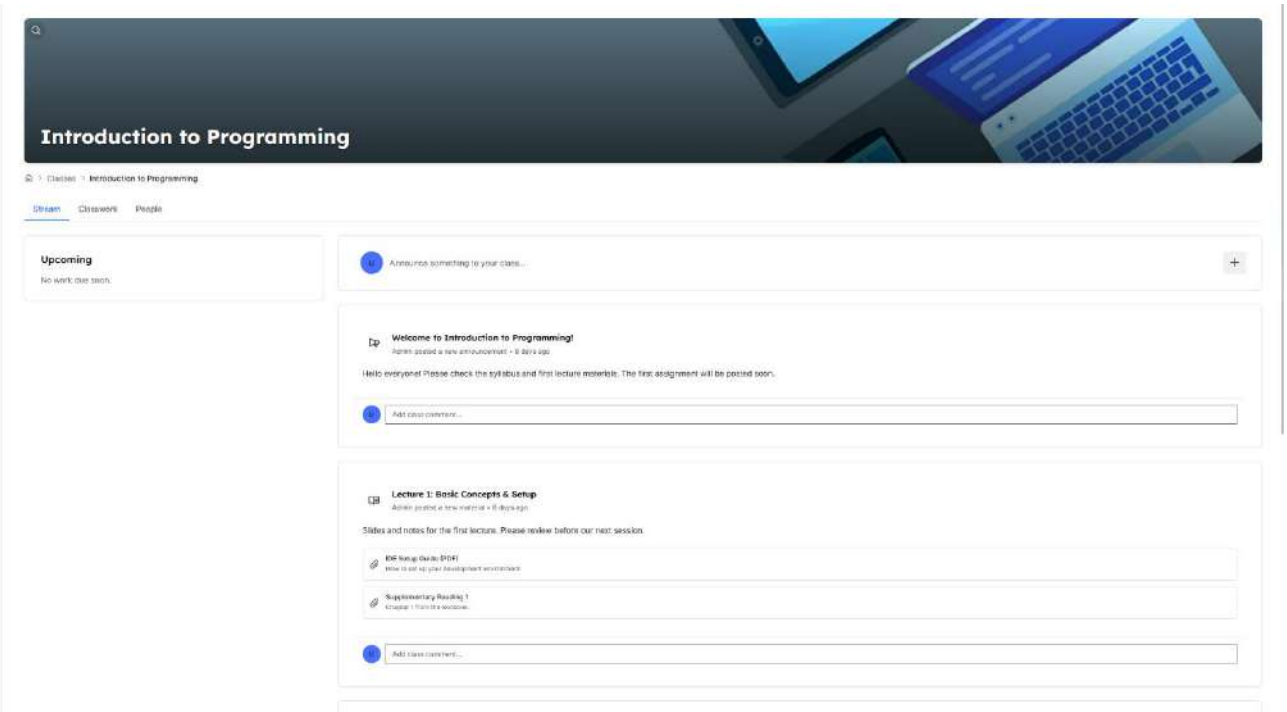

Рис. 4.3 Перегляд завдань предмету

На представленому (Рис. 4.3) можна бачити, що верхня частина сторінки зазвичай містить назву предмету (“Introduction to Programming”) та, можливо, навігаційні елементи (“хлібні крихти”, що показують шлях до поточної сторінки, наприклад, “Classes > Introduction to Programming”). Нижче розташовані вкладки для сегментації контенту, такі як “Stream” (Стрічка), “Classwork” (Завдання/Роботи) та “People” (Учасники), що дозволяє користувачеві легко перемикатися між різними аспектами курсу. Активна вкладка “Stream”(Стрічка) відображає хронологічно впорядкований потік інформаційних блоків.

У лівій частині може знаходитися блок “Upcoming” (Майбутні події/завдання), який інформує студентів про найближчі терміни здачі робіт або інші важливі дедлайни, сприяючи кращому плануванню навчальної діяльності. Основний простір стрічки займають інформаційні картки або пости, що представляють різні типи активностей. Наприклад, на рисунку видно оголошення (“Welcome to Introduction to Programming!”), яке було опубліковано адміністратором (“Admin posted a new announcement”) та містить вітальне повідомлення та загальну інформацію про курс. Нижче розташований блок, що стосується лекційного матеріалу (“Lecture 1: Basic Concepts & Setup”), який також був опублікований адміністратором (“Admin posted a new material”) і містить опис та, ймовірно, прикріплені файли (наприклад, “IDE Setup Guide (PDF)”, “Supplementary Reading 1”).

Кожен інформаційний блок у стрічці зазвичай містить заголовок, короткий опис або основний текст, інформацію про автора та час публікації. Важливою функціональною можливістю є поле для додавання коментарів ("Add class comment...") під кожним постом, що сприяє інтерактивній взаємодії між учасниками курсу, дозволяючи ставити запитання, обговорювати матеріали та отримувати зворотний зв'язок. У верхній частині стрічки може бути передбачено поле для створення нового оголошення або запису ("Announce something to your class..."), доступне, наприклад, викладачу або адміністратору курсу, а також кнопка "+" для швидкого додавання нових елементів (завдань, матеріалів тощо).

Тестування клієнтської веб-частини на Next.js включало ретельну перевірку всіх елементів користувацького інтерфейсу. Оцінювалася коректність відображення сторінок та компонентів, таких як календар розкладу, списки курсів та активностей, форми вводу даних. Особливу увагу було приділено перевірці інтерактивних елементів: роботи кнопок, навігаційних посилань, модальних вікон та коректності оновлення даних на сторінці після взаємодії з API. Також здійснювалася перевірка адаптивності веб-інтерфейсу та сумісності з основними сучасними веб-браузерами.

Загалом, розроблена система демонструє реалізацію ключового функціоналу, необхідного для автоматизації процесу обліку та аналізу відвідуваності. Серверне API забезпечує надійну основу для управління даними. Клієнтська веб-частина надає сучасний, інтерактивний та інтуїтивно зрозумілий інтерфейс. У ході тестування було підтверджено, що розроблений програмний комплекс загалом відповідає сформульованим функціональним та нефункціональним вимогам. Нефункціональні аспекти, такі як продуктивність веб-інтерфейсу, знаходяться на прийнятному рівні. Безпека забезпечується механізмами фреймворку Laravel.

4.3. Розробка інтерфейсу користувача додатку (C++ з ImGui)

Паралельно з розробкою веб-інтерфейсу, для забезпечення розширених можливостей аналізу даних та надання альтернативного доступу до функціоналу системи для певних категорій користувачів, було створено десктопний додаток. Для його реалізації було обрано мову програмування C++, відому своєю високою продуктивністю та можливістю низькорівневого контролю над системними ресурсами. Графічний користувацький інтерфейс (GUI) було побудовано з використанням легкої та гнучкої бібліотеки негайного режиму (immediate mode) Dear ImGui (ImGui), яка дозволяє швидко створювати кастомні та інтерактивні інтерфейси, особливо для інструментальних та аналітичних додатків. Взаємодія з серверною частиною здійснюється через HTTP-запити до раніше розробленого

API, для чого використовується бібліотека cpp-httplib, а обробка JSON-даних – за допомогою бібліотеки nlohmann/json.

Основою десктопного додатку є головний цикл, реалізований у файлі main.cpp, який керує ініціалізацією віконної системи (GLFW), графічного контексту (OpenGL через GLAD) та самої бібліотеки ImGui. У цьому циклі відбувається обробка подій вводу, формування нового кадру ImGui, рендеринг усіх активних вікон та елементів інтерфейсу, та подальше відображення згенерованого зображення на екрані. Для забезпечення коректного відображення кириличних символів в інтерфейсі було підключено відповідний шрифт (наприклад, NotoSans-Regular.ttf або системні шрифти Windows).

Одним із першочергових завдань була реалізація механізму автентифікації користувача. Цей функціонал інкапсульовано у функції ShowLoginPage (файл ui_windows.cpp) та логіці взаємодії з API, описаній у api_client.cpp. При запуску додатку, якщо користувач ще не автентифікований, відображається модальне вікно, що вимагає введення електронної пошти та пароля. Функція attemptLogin ініціює HTTP POST-запит до ендпоінту /api/login серверної частини. Перед цим обов'язково виконується запит на отримання CSRF-куки (getCsrfCookie), яка потім включається у заголовок запиту на вхід для забезпечення безпеки. У разі успішної автентифікації, сервер повертає сесійні куки, які зберігаються у глобальних змінних (g_csrfCookie, g_sessionCookie) для використання у подальших запитах, а статус g_isLoggedIn встановлюється у true, що відкриває доступ до основного функціоналу додатку. У випадку помилки автентифікації, користувачеві відображається відповідне повідомлення.

Після успішного входу користувачеві стає доступним головне меню (ShowMainMenuBar), через яке можна відкривати основні функціональні вікна додатку. Як продемонстровано на рис. 4.4, десктопний клієнт надає комплексний інтерфейс для роботи з даними. Одним з таких вікон є “Расписание с сервера (API)” (ShowApiScheduleTable, нижня частина рис. 4.4), призначене для відображення розкладу занять у табличному вигляді. Дані для цієї таблиці завантажуються з сервера за допомогою функції fetchApiSchedules, яка надсилає

автентифікований GET-запит до ендпоінту /api/schedules. Отримана JSON-відповідь парситься у вектор C++ структур ApiScheduleEntry (визначених у api_data_structures.h та api_data_structures.cpp) і відображається у вигляді таблиці з колонками, що містять ідентифікатор заняття, назву групи, предмет, список викладачів та час початку й кінця заняття. Для зручності користувача реалізовано спливаючі підказки з детальною інформацією при наведенні на певні елементи таблиці.

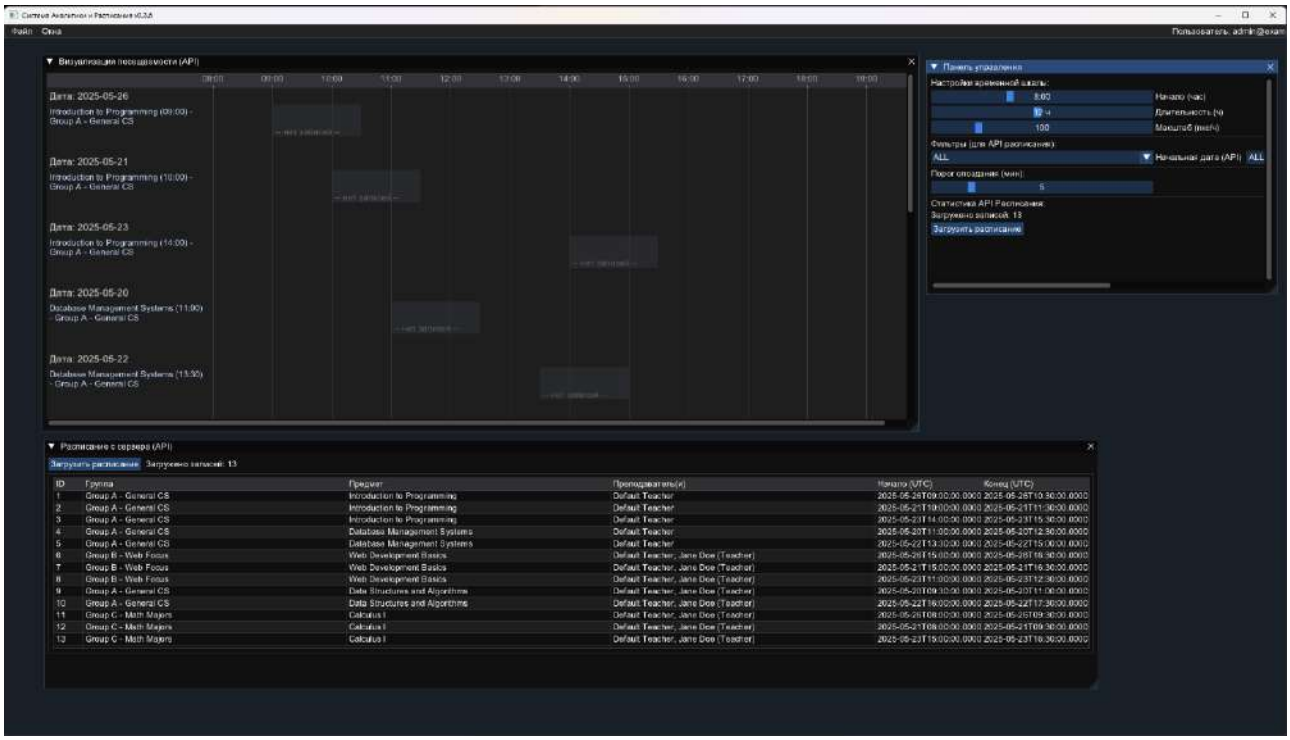

Рис. 4.4. Десктопний додаток

Особливу увагу при розробці десктопного додатку було приділено реалізації модуля візуалізації відвідуваності на часовій шкалі (ShowAttendanceVisualization, верхня частина рис. 4.4). Цей функціонал надає унікальну можливість графічного представлення присутності студентів на заняттях. Для малювання використовується об'єкт ImDrawList бібліотеки ImGui, що дозволяє створювати кастомні графічні примітиви. На екрані формується горизонтальна часова лінійка, параметри якої (початковий час, тривалість, масштаб в пікселях на годину) можуть налаштовуватися користувачем через “Панель управления” (ShowControlPanel, права частина рис. 4.4). Кожне заняття з завантаженого розкладу, що відповідає обраним у панелі управління фільтрам

дати, відображається на цій шкалі у вигляді фонового прямокутника, позиція та ширина якого відповідають часу початку та тривалості заняття. Зліва від шкали виводиться текстова інформація про заняття (Дата, Назва предмету, Група). Присутність кожного студента на конкретному занятті (дані про яку отримуються разом з розкладом з поля attendances структури ApiScheduleEntry) візуалізується у вигляді кольорової горизонтальної смужки, розташованої під відповідним заняттям. Початок смужки відповідає часу підключення студента до заняття, а її довжина – тривалості присутності (зазвичай до кінця заняття). Важливим елементом візуалізації є ідентифікація студентів, що запізнилися: якщо час підключення студента перевищує час початку заняття на встановлений у панелі управління поріг (s_ui_latenessThresholdMinutes), його смужка візуально виділяється (наприклад, червоною рамкою). При наведенні курсору на смужку студента відображається детальна інформація про нього та його відвідуваність. У випадку відсутності записів про відвідуваність для певного заняття, відображається відповідне повідомлення (наприклад, "-- нет записей --").

Взаємодія з API серверної частини, як зазначалося, реалізована у api_client.cpp з використанням бібліотеки cpp-httplib. Здійснюється коректна обробка HTTP-заголовків, включаючи передачу X-XSRF-TOKEN та сесійних кук для підтримки автентифікованої сесії. Відповіді сервера у форматі JSON парсяться за допомогою бібліотеки nlohmann/json у відповідні C++ структури, що забезпечує типізований доступ до отриманих даних. Реалізовано базову обробку можливих HTTP-помилок та помилок парсингу JSON, з виведенням відповідної інформації для користувача або в консоль розробника.

Десктопний додаток пройшов етап ретельного ручного тестування. Перевірялася працездатність вікна автентифікації, коректність відображення даних у таблиці розкладу та на часовій шкалі візуалізації відвідуваності. Тестувалися елементи управління в "Панелі управления", зокрема, застосування фільтрів дати, налаштування параметрів візуалізації та ініціація завантаження даних з сервера. Важливим аспектом була перевірка коректності обробки та відображення даних, отриманих від API.

Таким чином, розробка десктопного додатку на C++ з використанням ImGui дозволила створити спеціалізований інструмент, що доповнює функціонал веб-клієнта. Його практичне значення полягає у наданні викладачам та адміністрації потужних можливостей для візуального аналізу даних відвідуваності. Це може бути корисним для виявлення тенденцій (наприклад, систематичних запізнень або пропусків певними студентами чи на певних заняттях), моніторингу загальної дисципліни та прийняття обґрунтованих педагогічних або адміністративних рішень. Можливість налаштування часової шкали та фільтрів дозволяє адаптувати візуалізацію під конкретні аналітичні завдання.

Висновки до розділу 4

Було детально представлено процес розробки ключових програмних компонентів системи контролю відвідуваності студентів, а також висвітлено методику та узагальнено результати проведеного тестування. Розглянуто реалізацію серверної частини на PHP Laravel, клієнтської веб-частини на Next.js та десктопного додатку на C++ з використанням ImGui, що в сукупності формують цілісний програмний комплекс.

Розробка серверної частини (підрозділ 4.1) була зосереджена на створенні надійного та безпечного API, що забезпечує всю необхідну бізнес-логіку для управління даними про користувачів, розклад, відвідуваність та навчальні активності. Використання патерну MVC, можливостей Eloquent ORM та вбудованих механізмів Laravel дозволило реалізувати ефективну обробку запитів, валідацію даних та розмежування прав доступу. Функціональне тестування API через клієнтські додатки підтвердило коректність його роботи.

У рамках розробки інтерфейсу користувача сайту (підрозділ 4.2), реалізованого на Next.js, основну увагу було приділено створенню інтерактивного, інтуїтивно зрозумілого та адаптивного веб-додатку. Ключові компоненти, такі як головна інформаційна панель (рис. 4.1), список курсів

(ClassroomSection), календар розкладу (ScheduleList) та стрічка активностей предмету (рис. 4.3), забезпечують користувачам зручний доступ до необхідної інформації та функціоналу. Використання Redux Toolkit Query для взаємодії з API та UI-бібліотеки Rizzui сприяло створенню сучасного користувацького досвіду. Ретельне ручне тестування веб-інтерфейсу дозволило перевірити коректність відображення та взаємодії елементів.

Розробка інтерфейсу користувача додатку на C++ з ImGui (підрозділ 4.3) була спрямована на створення спеціалізованого інструменту з розширеними можливостями аналізу даних. Реалізовано функціонал автентифікації, табличного відображення розкладу та унікальний модуль графічної візуалізації відвідуваності на часовій шкалі (рис. 4.4), що надає цінні аналітичні можливості. Тестування десктопного клієнта підтвердило працездатність його основних функцій та коректність взаємодії з API.

Інтегрований у опис кожного компонента аналіз результатів тестування показав, що розроблений програмний комплекс загалом функціонує стабільно та відповідає сформульованим вимогам. Виявлені в ході ручного тестування недоліки були усунені, що дозволило досягти належного рівня якості програмного продукту. Практичне значення розробленої системи полягає у її потенціалі для автоматизації обліку відвідуваності, підвищення прозорості навчального процесу та надання інструментів для аналізу даних, що сприятиме оптимізації освітньої діяльності.

Таким чином, у четвертому розділі було продемонстровано успішну реалізацію всіх запланованих програмних компонентів та їхню відповідність проектним завданням

ВИСНОВКИ

У ході дослідження та розробки, що лягли в основу даної кваліфікаційної бакалаврської роботи, мною було вирішено актуальне науково-практичне завдання, яке полягало у створенні програмного комплексу для автоматизації процесу контролю та аналізу відвідуваності занять студентами. Розроблена система є комплексним рішенням, що охоплює серверний, веб-клієнтський та десктопний компоненти, реалізовані з використанням сучасного стеку технологій, та спрямована на оптимізацію навчального процесу й підвищення його ефективності.

На основі проведеного мною дослідження та виконаної розробки, я дійшов наступних ключових висновків:

1. Детальний аналіз предметної області та існуючих програмних рішень підтвердив високу актуальність автоматизації контролю відвідуваності. Мною було виявлено суттєві недоліки традиційних методів обліку, що полягають у значній трудомісткості, низькій оперативності та обмежених аналітичних можливостях. Я визначив, що сучасні освітні установи потребують гнучких, надійних та користувацько-орієнтованих цифрових інструментів для вирішення цього завдання, з особливим акцентом на полегшенні доступу до інформації для студентів та зменшенні навантаження на викладачів.
2. Мною було сформульовано комплексні функціональні та нефункціональні вимоги до розроблюваної системи, що стали основою для подальшого проектування. Я обґрунтував вибір технологічного стеку, що включає PHP з фреймворком Laravel для серверної частини, бібліотеку Next.js (React) для клієнтської веб-частини, мову C++ з бібліотекою ImGui для десктопного додатку та СУБД PostgreSQL, як такий, що оптимально відповідає поставленим завданням щодо продуктивності, масштабованості, безпеки та швидкості розробки.

3. Мною було розроблено трирівневу архітектуру програмного комплексу, що забезпечує чітке розділення відповідальностей між рівнем представлення, рівнем бізнес-логіки та рівнем доступу до даних. Я спроектував детальну реляційну модель бази даних PostgreSQL, що охоплює всі необхідні сутності та їхні взаємозв'язки для зберігання інформації про користувачів, навчальні групи, дисципліни, розклад, відвідуваність та супутні навчальні активності. Також я розробив уніфікований RESTful API для взаємодії між серверною частиною та клієнтськими додатками.
4. У ході роботи я успішно реалізував усі ключові програмні компоненти системи. Серверна частина на PHP Laravel забезпечує надійну обробку бізнес-логіки, автентифікацію, авторизацію та управління даними. Клієнтська веб-частина на Next.js надає сучасний, інтерактивний та адаптивний інтерфейс для студентів та викладачів. Десктопний додаток на C++ пропонує унікальні можливості для візуалізації та аналізу даних відвідуваності, доповнюючи функціонал веб-клієнта.
5. Проведене мною ручне функціональне тестування та тестування користувацького інтерфейсу підтвердило працездатність основного функціоналу системи та його відповідність висунутим вимогам. Виявлені в ході тестування недоліки були мною проаналізовані та усунені, що дозволило досягти задовільного рівня стабільності та надійності програмного продукту.
6. Розроблений мною програмний комплекс має значне практичне значення. Він пропонує дієвий інструмент для автоматизації процесу контролю відвідуваності, сприяє підвищенню прозорості навчального процесу, зменшує адміністративне навантаження на викладачів, стимулює самоорганізацію студентів та надає адміністрації цінні дані для аналізу та прийняття управлінських рішень. Реалізовані користувацькі інтерфейси, зокрема веб-панель з компонентами ClassroomSection, ScheduleList та

RecentActivities, а також десктопна візуалізація відвідуваності, спрямовані на забезпечення зручності та ефективності роботи з системою.

Таким чином, я вважаю, що мета кваліфікаційної бакалаврської роботи, яка полягала у розробці програмного додатку для ефективного контролю відвідування занять студентами, була успішно досягнута. Усі поставлені завдання були мною виконані. Створений програмний продукт демонструє можливість ефективного застосування обраного стеку технологій для вирішення актуальних завдань цифровізації освітнього процесу та має потенціал для подальшого розвитку, вдосконалення та впровадження у практичну діяльність закладів вищої освіти. Виконана робота підтверджує набуття мною необхідних теоретичних знань та практичних навичок у галузі інженерії програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дей М. О. Laravel. Розробка сучасних веб-додатків на PHP / М. О. Дей. – Київ : Комп’ютерний Всесвіт, 2022. – 512 с.

2. Фрімен Е. Т. JavaScript та jQuery. Інтерактивна веб-розробка / Е. Т. Фрімен, Е. М. Робсон. – Київ : Діалектика, 2020. – 656 с.

3. Бэнкс А. Р. React и Redux: функциональная веб-разработка / А. Р. Бэнкс, Е. М. Порселло. – СПб. : Питер, 2018. – 336 с.

4. Страуструп Б. Й. Программирование: принципы и практика с использованием C++ / Б. Й. Страуструп. – М. : Вильямс, 2016. – 1328 с.

5. Мартін Р. С. Чиста архітектура. Мистецтво розробки програмного забезпечення / Р. С. Мартін. – Харків : Фабула, 2019. – 368 с.

6. Харрісон Г. М. PostgreSQL. Детальний посібник / Г. М. Харрісон. – Київ : Університетська книга, 2021. – 488 с.

7. МакЛафлін Б. Д. PHP и MySQL. Разработка веб-приложений / Б. Д. МакЛафлін. – СПб. : БХВ-Петербург, 2019. – 752 с.

8. Робін В. П. React. Швидкий старт з хуками / В. П. Робін, А. С. Морель. – Львів : Видавництво Старого Лева, 2022. – 450 с.

9. Шилдт Г. А. C++: руководство для начинающих / Г. А. Шилдт. – М. : Діалектика, 2020. – 720 с.

10. Фаулер М. К. Рефакторинг. Улучшение проекта существующего кода / М. К. Фаулер, К. Бек. – СПб. : ДіаСофт, 2019. – 448 с.

11. Петренко В. П. Застосування веб-технологій для моніторингу відвідуваності в освітніх закладах / В. П. Петренко // Інноваційні технології в освіті та науці. – 2023. – № 1. – С. 112–119.

12. Коваленко А. В. Розробка інтегрованої системи обліку відвідуваності "AttendancePro" для університетів / А. В. Коваленко, О. М. Лисенко, П. С. Іванов // Інформаційні системи та технології: зб. матеріалів X Міжнар. наук.-техн. конф. – Львів : Вид-во Львівської політехніки, 2022. – С. 315–317.

13. Сидоренко О. І. Методи та засоби автоматизації контролю навчальної дисципліни студентів / О. І. Сидоренко, М. В. Кравчук // Проблеми інформатизації навчального процесу. – 2021. – Вип. 2. – С. 45–53.

14. Гамма Е. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Е. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – СПб. : Питер, 2007. – 368 с.

15. Макконнелл С. Совершенный код. Мастер-класс / С. Макконнелл. – М. : Русская Редакция, 2005. – 896 с.

16. Себаста Р. В. Основные концепции языков программирования / Р. В. Себаста. – М. : Вильямс, 2015. – 768 с.

17. Ульман Дж. Д. Введение в системы баз данных / Дж. Д. Ульман, Дж. Уидом. – М. : ЛОРИ, 2000. – 376 с.

18. Зандстра М. PHP. Объекты, шаблоны и методики программирования / М. Зандстра. – СПб. : Діалектика, 2019. – 752 с.

19. Кнут Д. Э. Искусство программирования. Том 1. Основные алгоритмы / Д. Э. Кнут. – М. : Вильямс, 2011. – 720 с.

20. Кормен Т. Х. Алгоритмы: построение и анализ / Т. Х. Кормен, Ч. И. Лейзерсон, Р. Л. Ривест, К. Штайн. – М. : Вильямс, 2013. – 1328 с.

21. Боос А. Laravel. Створення веб-додатків з нуля / А. Боос, Д. Шторм. – Київ : Наш Формат, 2023. – 384 с.

22. Іванов П. С. Архітектурні підходи до розробки сучасних веб-додатків / П. С. Іванов // Вісник комп’ютерних наук та інженерії. – 2022. – Т. 4, № 2. – С. 78–85.

23. Лисенко О. М. Сучасні підходи до управління станом у React-додатках / О. М. Лисенко, А. В. Коваленко, П. С. Іванов // Інформаційні технології: теорія та практика: зб. матеріалів V Всеукр. наук.-практ. конф. студентів та молодих вчених. – Київ : КПІ ім. Ігоря Сікорського, 2023. – С. 102–104.

24. Мейєрс С. Эффективное использование C++. 55 верных способов улучшить структуру и код ваших программ / С. Мейєрс. – М. : ДМК Пресс, 2016. – 304 с.

25. Хомоненко А. Д. Базы данных: Учебник для высших учебных заведений / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев. – СПб. : КОРОНА-Век, 2014. – 736 с.

ДОДАТКИ

Додаток А

Структура бази даних

```
create table migrations
(
    id        serial
        primary key,
    migration varchar(255) not null,
    batch     integer      not null
);

alter table migrations
    owner to root;

create table password_reset_tokens
(
    email      varchar(255) not null
        primary key,
    token      varchar(255) not null,
    created_at timestamp(0)
);

alter table password_reset_tokens
    owner to root;

create table sessions
(
    id           varchar(255) not null
        primary key,
    user_id      bigint,
    ip_address   varchar(45),
    user_agent   text,
    payload      text         not null,
    last_activity integer     not null
);

alter table sessions
    owner to root;

create index sessions_user_id_index
    on sessions (user_id);
```

```
create index sessions_last_activity_index
    on sessions (last_activity);

create table cache
(
    key        varchar(255) not null
        primary key,
    value      text         not null,
    expiration integer      not null
);

alter table cache
    owner to root;

create table cache_locks
(
    key        varchar(255) not null
        primary key,
    owner      varchar(255) not null,
    expiration integer      not null
);

alter table cache_locks
    owner to root;

create table jobs
(
    id          bigserial
        primary key,
    queue       varchar(255) not null,
    payload     text         not null,
    attempts    smallint     not null,
    reserved_at  integer,
    available_at integer      not null,
    created_at   integer      not null
);

alter table jobs
    owner to root;
```

```
create index jobs_queue_index
    on jobs (queue);

create table job_batches
(
    id            varchar(255) not null
        primary key,
    name          varchar(255) not null,
    total_jobs    integer      not null,
    pending_jobs  integer      not null,
    failed_jobs   integer      not null,
    failed_job_ids text        not null,
    options       text,
    cancelled_at  integer,
    created_at    integer      not null,
    finished_at   integer
);

alter table job_batches
    owner to root;

create table failed_jobs
(
    id         bigserial
        primary key,
    uuid       varchar(255)                        not null
        constraint failed_jobs_uuid_unique
            unique,
    connection text                                not null,
    queue      text                                not null,
    payload    text                                not null,
    exception  text                                not null,
    failed_at  timestamp(0) default CURRENT_TIMESTAMP not null
);

alter table failed_jobs
    owner to root;

create table personal_access_tokens
(
```

```
    id              bigserial
        primary key,
    tokenable_type varchar(255) not null,
    tokenable_id   bigint       not null,
    name           varchar(255) not null,
    token          varchar(64)  not null
        constraint personal_access_tokens_token_unique
            unique,
    abilities      text,
    last_used_at   timestamp(0),
    expires_at     timestamp(0),
    created_at     timestamp(0),
    updated_at     timestamp(0)
);

alter table personal_access_tokens
    owner to root;

create index personal_access_tokens_tokenable_type_tokenable_id_index
    on personal_access_tokens (tokenable_type, tokenable_id);

create table permissions
(
    id         bigserial
        primary key,
    name       varchar(255) not null,
    guard_name varchar(255) not null,
    created_at timestamp(0),
    updated_at timestamp(0),
    constraint permissions_name_guard_name_unique
        unique (name, guard_name)
);

alter table permissions
    owner to root;

create table roles
(
    id         bigserial
        primary key,
```

```
    name       varchar(255) not null,
    guard_name varchar(255) not null,
    created_at timestamp(0),
    updated_at timestamp(0),
    constraint roles_name_guard_name_unique
        unique (name, guard_name)
);

alter table roles
    owner to root;

create table model_has_permissions
(
    permission_id bigint       not null
        constraint model_has_permissions_permission_id_foreign
            references permissions
            on delete cascade,
    model_type    varchar(255) not null,
    model_id      bigint       not null,
    primary key (permission_id, model_id, model_type)
);

alter table model_has_permissions
    owner to root;

create index model_has_permissions_model_id_model_type_index
    on model_has_permissions (model_id, model_type);

create table model_has_roles
(
    role_id    bigint       not null
        constraint model_has_roles_role_id_foreign
            references roles
            on delete cascade,
    model_type varchar(255) not null,
    model_id   bigint       not null,
    primary key (role_id, model_id, model_type)
);

alter table model_has_roles
```

```
    owner to root;

create index model_has_roles_model_id_model_type_index
    on model_has_roles (model_id, model_type);

create table role_has_permissions
(
    permission_id bigint not null
        constraint role_has_permissions_permission_id_foreign
            references permissions
            on delete cascade,
    role_id       bigint not null
        constraint role_has_permissions_role_id_foreign
            references roles
            on delete cascade,
    primary key (permission_id, role_id)
);

alter table role_has_permissions
    owner to root;

create table groups
(
    id          bigserial
        primary key,
    title       varchar(50) not null
        constraint groups_title_unique
            unique,
    description text,
    created_at  timestamp(0),
    updated_at  timestamp(0)
);

alter table groups
    owner to root;

create table users
(
    id                bigserial
        primary key,
```

```
    group_id          bigint
        constraint users_group_id_foreign
            references groups
            on delete set null,
    name              varchar(255) not null,
    email             varchar(255) not null
        constraint users_email_unique
            unique,
    email_verified_at timestamp(0),
    password          varchar(255) not null,
    remember_token    varchar(100),
    created_at        timestamp(0),
    updated_at        timestamp(0)
);

alter table users
    owner to root;

create table classroom_lessons
(
    id         bigserial
        primary key,
    room       varchar(50) not null,
    building   varchar(100),
    created_at timestamp(0),
    updated_at timestamp(0)
);

alter table classroom_lessons
    owner to root;

create table online_lessons
(
    id          bigserial
        primary key,
    meeting_url varchar(500) not null,
    meeting_id  varchar(100),
    passcode    varchar(100),
    created_at  timestamp(0),
    updated_at  timestamp(0)
```

```
);

alter table online_lessons
    owner to root;

create table activity_types
(
    id          bigserial
        primary key,
    type_name   varchar(50) not null
        constraint activity_types_type_name_unique
            unique,
    description text,
    created_at  timestamp(0),
    updated_at  timestamp(0)
);

alter table activity_types
    owner to root;

create table files
(
    id            bigserial
        primary key,
    original_name  varchar(255) not null,
    file_size      bigint       not null,
    content_type   varchar(50)  not null,
    file_path      varchar(500) not null,
    file_name      varchar(255) not null,
    file_extension varchar(50)  not null,
    file_type      varchar(50)  not null,
    file_hash      varchar(255) not null
        constraint files_file_hash_unique
            unique,
    storage_type   varchar(50)  not null,
    created_at     timestamp(0),
    updated_at     timestamp(0)
);

alter table files
```

```
    owner to root;

create table classes
(
    id           bigserial
        primary key,
    title        varchar(100) not null,
    description  text,
    created_at   timestamp(0),
    updated_at   timestamp(0),
    image_file_id bigint
        constraint classes_image_file_id_foreign
            references files
            on delete set null
);

alter table classes
    owner to root;

create index classes_image_file_id_index
    on classes (image_file_id);

create table schedules
(
    id             bigserial
        primary key,
    group_id       bigint       not null
        constraint schedules_group_id_foreign
            references groups
            on delete cascade,
    class_id       bigint       not null
        constraint schedules_class_id_foreign
            references classes
            on delete restrict,
    lessonable_type varchar(255) not null,
    lessonable_id  bigint       not null,
    started_at     timestamp(0) not null,
    ended_at       timestamp(0) not null,
    created_at     timestamp(0),
    updated_at     timestamp(0)
```

```
);

alter table schedules
    owner to root;

create index schedules_lessonable_type_lessonable_id_index
    on schedules (lessonable_type, lessonable_id);

create index schedules_lessonable_id_lessonable_type_index
    on schedules (lessonable_id, lessonable_type);

create table activities
(
    id              bigserial
        primary key,
    class_id        bigint          not null
        constraint activities_class_id_foreign
            references classes
            on delete restrict,
    group_id        bigint          not null
        constraint activities_group_id_foreign
            references groups
            on delete cascade,
    activity_type_id bigint          not null
        constraint activities_activity_type_id_foreign
            references activity_types
            on delete restrict,
    created_by      bigint          not null
        constraint activities_created_by_foreign
            references users
            on delete restrict,
    title           varchar(200)    not null,
    description     text,
    due_date        timestamp(0),
    max_points      integer,
    "order"         integer default 0 not null,
    file_id         bigint
        constraint activities_file_id_foreign
            references files
            on delete set null,
```

```
    created_at      timestamp(0),
    updated_at      timestamp(0)
);

alter table activities
    owner to root;

create table activity_materials
(
    id          bigserial
        primary key,
    activity_id bigint      not null
        constraint activity_materials_activity_id_foreign
            references activities
            on delete cascade,
    title       varchar(200) not null,
    description text,
    file_id     bigint
        constraint activity_materials_file_id_foreign
            references files
            on delete set null,
    created_at  timestamp(0),
    updated_at  timestamp(0)
);

alter table activity_materials
    owner to root;

create table submissions
(
    id          bigserial
        primary key,
    activity_id  bigint                      not null
        constraint submissions_activity_id_foreign
            references activities
            on delete cascade,
    user_id      bigint                      not null
        constraint submissions_user_id_foreign
            references users
            on delete cascade,
```

```
    file_id      bigint
        constraint submissions_file_id_foreign
            references files
            on delete set null,
    submitted_at timestamp(0) default CURRENT_TIMESTAMP not null,
    content      text,
    created_at   timestamp(0),
    updated_at   timestamp(0),
    constraint submissions_activity_id_user_id_unique
        unique (activity_id, user_id)
);

alter table submissions
    owner to root;

create table grades
(
    id           bigserial
        primary key,
    submission_id bigint not null
        constraint grades_submission_id_unique
            unique
        constraint grades_submission_id_foreign
            references submissions
            on delete cascade,
    graded_by     bigint not null
        constraint grades_graded_by_foreign
            references users
            on delete restrict,
    grade        integer,
    feedback     text,
    created_at   timestamp(0),
    updated_at   timestamp(0)
);

alter table grades
    owner to root;

create table class_user
(
```

```
    id          bigserial
        primary key,
    class_id    bigint                                  not null
        constraint class_user_class_id_foreign
            references classes
            on delete cascade,
    user_id     bigint                                  not null
        constraint class_user_user_id_foreign
            references users
            on delete cascade,
    role        varchar(255) default 'teacher'::character varying not null,
    created_at timestamp(0),
    updated_at timestamp(0),
    constraint class_user_class_id_user_id_unique
        unique (class_id, user_id)
);

alter table class_user
    owner to root;

create index class_user_role_index
    on class_user (role);

create table class_group
(
    id          bigserial
        primary key,
    class_id    bigint not null
        constraint class_group_class_id_foreign
            references classes
            on delete cascade,
    group_id    bigint not null
        constraint class_group_group_id_foreign
            references groups
            on delete cascade,
    start_date date    not null,
    end_date    date    not null,
    created_at timestamp(0),
    updated_at timestamp(0),
    constraint class_group_class_id_group_id_unique
```

```
    unique (class_id, group_id)
);

alter table class_group
    owner to root;

create table attendances
(
    id          bigserial
        primary key,
    schedule_id bigint       not null
        constraint attendances_schedule_id_foreign
            references schedules
            on delete cascade,
    user_id     bigint       not null
        constraint attendances_user_id_foreign
            references users
            on delete cascade,
    status      varchar(255) not null,
    created_at  timestamp(0),
    updated_at  timestamp(0)
);

alter table attendances
    owner to root;

create index attendances_status_index
    on attendances (status);
```