ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система автоматизації обліку відвідувань на основі технології QR-кодування з використанням Node.js та PostgreSQL»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Технологій цифрового розвитку»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Андрій ОПАНАСЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ТЦР-42
Андрій ОПАНАСЕНКО

Керівник: Артур ГАВОР
ст. викл.

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технологій цифрового розвитку
_____ Вікторія ЖЕБКА
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Опанасенку Андрію Юрійовичу

1. Тема кваліфікаційної роботи: «Система автоматизації обліку відвідувань на основі технології QR-кодування з використанням Node.js та PostgreSQL»
керівник кваліфікаційної роботи ст. викл. Артур ГАВОР,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51 .
2. Строк подання кваліфікаційної роботи « 18 » травня 2026 року.
3. Вихідні дані до кваліфікаційної роботи: технічна документація платформи Node.js та фреймворку Express.js, документація СУБД PostgreSQL 16, специфікація стандарту ISO/IEC 18004 для QR-кодів, специфікації форматів CSV та XLSX для імпорту даних про студентів, стандарт RFC 7519 (JSON Web Token), вимоги до безпеки веб-застосунків (захист від XSS, CSRF, SQL-ін'єкцій відповідно до рекомендацій OWASP).
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та класифікація існуючих систем обліку відвідуваності, порівняльний аналіз наявних рішень, формулювання функціональних та нефункціональних вимог до застосунку.
2. Обґрунтування вибору технологічного стеку: Node.js з Express.js для серверної частини, PostgreSQL як СУБД, JWT з дворівневою схемою токенів для автентифікації, бібліотеки qrcode та jsQR для роботи з QR-кодами, засоби клієнтської частини.

3. Проєктування клієнт-серверної архітектури REST API, розробка ER-моделі бази даних, діаграми варіантів використання та послідовності для ключових сценаріїв, проєктування інтерфейсу користувача.
4. Реалізація серверної частини (REST API, автентифікація, генерація QR-кодів, імпорт студентів з CSV/XLSX, формування звітів) та клієнтської частини (SPA, сканування QR через камеру, аналітика Chart.js), тестування функціональних сценаріїв.

5. Перелік графічного матеріалу: *презентація*
   1. Аналіз аналогів.
   2. Вимоги до програмного забезпечення.
   3. Програмні засоби реалізації.
   4. Діаграма варіантів використання.
   5. Архітектура програмного забезпечення.
   6. Діаграма послідовності взаємодії користувача із застосунком.
   7. Екранні форми.
   8. Апробація результатів дослідження.
6. Дата видачі завдання «_20_» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання	Примітка
1	Підбір та аналіз науково-технічної літератури з питань обліку відвідуваності та веб-розробки	20.02 – 25.02.2026	Виконано
2	Аналіз існуючих програмних рішень для обліку відвідуваності студентів та визначення вимог	26.02 – 10.03.2026	Виконано
3	Вибір та обґрунтування технологічного стеку (Node.js, Express.js, PostgreSQL, JWT, QR-коди)	11.03 – 20.03.2026	Виконано
4	Проєктування архітектури застосунку, ER-моделі бази даних та діаграм варіантів використання	21.03 – 25.03.2026	Виконано
5	Програмна реалізація серверної частини: REST API, автентифікація JWT, генерація QR-кодів, імпорт CSV/Excel	26.03 – 10.04.2026	Виконано
6	Реалізація клієнтської частини: SPA-інтерфейс, сканування QR через камеру, аналітика Chart.js	11.04 – 20.04.2026	Виконано
7	Тестування застосунку, оформлення роботи: вступ, висновки, реферат	21.04 – 26.04.2026	Виконано
8	Розробка демонстраційних матеріалів	26.04 – 02.05.2026	Виконано
9	Попередній захист роботи	03.05 – 05.05.2026	Виконано

Здобувач вищої освіти _____ Андрій ОПАНАСЕНКО

Керівник
кваліфікаційної роботи _____ Артур ГАВОР

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 15 табл., 26 рис., 24 джерела.

Мета роботи – підвищення ефективності обліку відвідувань студентів у закладах вищої освіти. Це здійснюється шляхом автоматизації реєстрації присутності через QR-кодування. Це забезпечує швидке та точне фіксування відвідуваності, зручне управління даними для викладачів і адміністраторів, а також формування аналітичних звітів.

Об'єкт дослідження – процес автоматизованого обліку відвідувань студентів у закладах вищої освіти.

Предмет дослідження – методи та засоби проєктування і розробки системи для обліку відвідувань на основі технології QR-кодування з використанням платформи Node.js та PostgreSQL.

Короткий зміст роботи: проведено огляд та аналіз для порівняння існуючих систем обліку відвідуваності студентів з метою виявлення їх недоліків та обґрунтування потребності розробки власного рішення. Сформульовано функціональні та нефункціональні вимоги. Обґрунтовано вибір технологічного стеку: серверна частина – Node.js 24 із фреймворком Express.js, збереження даних – PostgreSQL 16, автентифікація – JSON Web Token з механізмом refresh-токенів, генерація QR-кодів – спеціалізована бібліотека «qrcode». Спроєктовано архітектуру за патерном REST API, розроблено ER-модель бази даних, діаграми варіантів використання та послідовності. Реалізовано систему із функціями: управління студентами, заняттями та відвідуваністю, генерація та зчитування QR-кодів через камеру пристрою, ролева модель(адміністратор/викладач/ публічний доступ для студентів), формування звітів та їх експорт у форматі Excel, імпорт студентів із CSV та Excel файлів. Проведено функціональне тестування застосунку.

Сфера використання: заклади вищої та загальної середньої освіти для автоматизованої реєстрації відвідувань студентів і учнів.

КЛЮЧОВІ СЛОВА: ОБЛІК ВІДВІДУВАНЬ, NODE.JS, POSTGRESQL, QR-КОД, REST API, JWT, EXPRESS.JS, ВІДВІДУВАНІСТЬ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

CRUD – Create, Read, Update, Delete

CSV – Comma-Separated Values

HTTP – HyperText Transfer Protocol

HTTPS– HyperText Transfer Protocol Secure

JSON – JavaScript Object Notation

JWT – JSON Web Token

QR – Quick Response

REST – Representational State Transfer

SPA – Single Page Application

SQL – Structured Query Language

СУБД – Система управління базами даних

UI – User Interface

UX – User Experience

XSS – Cross-Site Scripting

CSRF – Cross-Site Request Forgery

ВСТУП

Автоматизація адміністративних процесів у закладах освіти є однією з ключових задач цифрової трансформації сучасного суспільства. Облік відвідуваності студентів це один із найбільш трудомістких рутинних процесів, що потребує значних часових витрат з боку викладачів і адміністраторів. Традиційні методи фіксації присутності такі як паперові журнали, усні переклики або ручне введення даних до електронних таблиць мають суттєві недоліки: схильність до помилок, можливість фальсифікації, відсутність оперативного доступу до зведеної статистики та значні витрати робочого часу.

Технологія швидкого реагування набула широкого поширення завдяки своїй простоті, надійності та доступності на будь-якому сучасному пристрої, що має камеру. Поєднання QR-кодів із сучасними веб-технологіями дозволяє побудувати ефективну систему обліку відвідувань, де кожен студент ідентифікується за унікальним кодом, а процес реєстрації присутності займає лічені секунди.

Актуальність даної роботи зумовлена загальносвітовою тенденцією цифровізації освітніх процесів: відповідно до концепції цифрової трансформації освіти, заклади вищої освіти активно переходять від паперового документообігу до інтегрованих цифрових платформ управління навчальним процесом. Облік відвідуваності залишається одним із небагатьох адміністративних процесів, які у значній частині вітчизняних вищих навчальних закладів досі ведуться вручну – у паперових журналах або у розрізнених електронних таблицях без централізованої аналітики.

Водночас масове поширення смартфонів та планшетів серед викладацького складу і студентів створює технічне підґрунтя для впровадження рішень на основі QR-кодів без придбання спеціалізованого обладнання. Більшість існуючих комерційних систем орієнтовані на зарубіжні ринки та є або економічно недоступними для закладів із обмеженим бюджетом, або не підтримують організаційну специфіку вітчизняної вищої освіти (групова структура, предметно-потоковий розклад, рольова модель «адміністратор – викладач»), або потребують

встановлення додаткового програмного забезпечення на кожен пристрій, що ускладнює їх масштабування.

Мета дипломної роботи – підвищення ефективності обліку відвідувань студентів шляхом автоматизації реєстрації присутності через QR-кодування, реалізованого у вигляді веб-застосунку на платформі Node.js із СУБД PostgreSQL, що забезпечує миттєву фіксацію відвідуваності, розмежування прав доступу між адміністраторами та викладачами і формування детальної аналітики.

Для досягнення поставленої мети в роботі вирішуються такі завдання:

- проаналізувати існуючі рішення для обліку відвідуваності студентів, визначити їх переваги та недоліки;
- сформулювати функціональні та нефункціональні вимоги до розроблюваного застосунку;
- обґрунтувати вибір технологічного стеку для реалізації серверної та клієнтської частин;
- спроєктувати архітектуру системи, ER-модель бази даних та діаграми взаємодії компонентів;
- реалізувати серверну частину із REST API та систему автентифікації на основі JWT з refresh-токенами;
- реалізувати функціонал генерації та зчитування QR-кодів через камеру пристрою;
- розробити клієнтську частину у вигляді односторінкового застосунку (SPA) з розмежуванням прав доступу;
- впровадити функцію формування звітів та їх експорту у форматі Excel;
- провести тестування застосунку та оцінити відповідність реалізованого рішення висунутим вимогам.

Об'єктом дослідження є процес автоматизованого обліку відвідувань студентів у закладах вищої освіти.

Предметом дослідження є методи та засоби проєктування і розробки системи для обліку відвідувань на основі технології QR-кодування з використанням платформи Node.js та PostgreSQL.

Практична значущість роботи полягає у розробці повнофункціональної системи, що може бути розгорнутий у будь-якому закладі освіти без додаткових витрат на ліцензії, забезпечує реєстрацію відвідуваності через будь-який пристрій із браузером, підтримує кілька ролей користувачів та формує детальну аналітичну звітність.

У роботі використовуються методи порівняльного аналізу існуючих програмних рішень для обліку відвідувань, системного проєктування (ER-моделювання бази даних, UML-діаграми архітектури та взаємодії компонентів), компонентного проєктування серверної і клієнтської частин застосунку на платформі Node.js, а також функціонального тестування розробленого застосунку методом «чорної скриньки».

Структура дипломної роботи складається з вступу, чотирьох розділів, висновків та списку використаних джерел. У першому розділі проводиться аналіз предметної області та існуючих рішень. У другому розділі обґрунтовується вибір технологічного стеку. У третьому розділі описується проєктування системи. У четвертому розділі висвітлюється реалізація та тестування застосунку.

1. VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Застосування QR-Кодування для автоматизації обліку відвідуваності у закладах вищої освіти.
2. Всеукраїнська наукова-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація. Аналіз вразливостей протоколу Oauth 2.0 у вебзастосунках та методи захисту від типових атак

За результатами апробації опубліковано тези доповідей[1-2].

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд та класифікація сучасних систем обліку відвідувань

Облік відвідуваності студентів є важливою складовою адміністративного процесу у закладах вищої освіти. Від точності та своєчасності фіксації присутності залежать успішність студента, стипендіальне рейтингування та дотримання навчального плану. З розвитком цифрових технологій з'явилась велика кількість рішень, що намагаються автоматизувати цей процес.

Сучасні системи обліку відвідувань можна класифікувати за кількома критеріями.

Таблиця 1.1

Класифікація критеріїв системи обліку відвідувань

Критерій	Тип/характеристика
За способом ідентифікації студента	- картки з магнітною смугою або RFID-мітки, Потребують спеціального зчитувального обладнання, встановленого біля входу до аудиторії або на прохідних; - біометричні системи, вони забезпечують високу точність, але коштують значно дорожче та викликають занепокоєння щодо конфіденційності; - QR-коди, студент пред'являє QR-код на смартфоні або роздрукований варіант, викладач сканує через веб-камеру. Не потребують додаткового обладнання; - мобільні застосунки з геолокацією, вони перевіряють, чи знаходиться студент у межах навчального корпусу. Потребують смартфон та дозволи на геолокацію; - ручне введення, коли викладач відзначає присутніх у паперовому журналі.
За типом розгортання	- хмарні рішення, вони не потребують власної серверної інфраструктури, доступні з будь-якого пристрою, але залежать від стороннього провайдера та потребують сплати підписки; - серверні рішення, вони розгортаються на власному сервері закладу, забезпечують повний контроль над даними; - десктопні застосунки: встановлюються на конкретний комп'ютер, що обмежує доступ до системи; - веб-застосунки: доступні з будь-якого пристрою з браузером, не потребують встановлення.

Продовження таблиці 1.1

За функціональністю	- базові системи реєстрації присутності: дозволяють лише відзначити студентів як присутніх або відсутніх; - системи з аналітикою, вони формують звіти за групами, предметами, студентами, дозволяють відстежувати динаміку відвідуваності; - комплексні платформи управління навчальним процесом. Включають модулі розкладу, оцінювання, комунікацій та обліку відвідувань.
За цільовою аудиторією	- для шкіл та ліцеїв. - для університетів та коледжів. - для корпоративного навчання.

Аналіз ринку показує, що найпопулярнішими є хмарні та веб-рішення з підтримкою кількох методів ідентифікації. При цьому зростає попит на рішення з підтримкою QR-кодів як найдоступнішого і найшвидшого способу ідентифікації без додаткового обладнання.

1.2. Аналіз існуючих рішень та їх недоліків

Для розуміння поточного стану ринку та виявлення прогалин, які має заповнити розроблювана система, проведено аналіз кількох найбільш відомих систем.

Google Forms разом з Sheets – це один із найпростіших підходів. Студенти заповнюють Google-форму з паролем заняття, а результати автоматично потрапляють до таблиці. Перевага полягає в нульовій вартості та відсутності потреби у розгортанні. Недоліки: відсутня ідентифікація студента, будь-хто може ввести чуже ім'я; немає автоматичного зв'язку з розкладом; аналітика примітивна; потрібен доступ до акаунту Google.

Classter – це комплексна хмарна система для освітніх закладів, яка включає модуль відвідуваності поряд із модулями оцінювання, комунікацій та управління. Перевагою є широкий функціонал. Серед недоліків: висока ціна підписки (від 3 до 10 доларів США на студента на місяць), складний процес впровадження та налаштування, інтерфейс доступний переважно англійською мовою, залежність від стороннього хмарного провайдера.

Attendo – це рішення для Android та iOS, де викладач позначає присутніх у мобільному додатку. Це зручне для індивідуального використання, але не має централізованого сервера, що ускладнює адміністрування у масштабах закладу. Відсутня інтеграція з навчальними системами.

Окремі рішення такі як Quick scan attendance на базі QR-кодів, існують у вигляді мобільних застосунків або простих веб-форм. Як правило, вони реалізують лише базову функцію сканування без повноцінної адміністративної частини: немає управління групами, не формуються звіти, відсутня ролева модель.

Університетські портали такі системи як «Електронний деканат», «АСУ ВНЗ» та подібні вітчизняні розробки. Деякі українські університети розробили власні портали, інтегровані з системами обліку відвідувань. Такі рішення зазвичай є частиною більших ERP-систем, специфічних для конкретного закладу, не доступних для повторного використання, і не підтримують сканування QR-кодів через браузер. Узагальнюючи аналіз існуючих рішень, виділено такі спільні проблеми:

- більшість хмарних рішень є платними та орієнтованими на іноземний ринок, не підтримуючи потреби українських закладів;
- безкоштовні інструменти (Google Forms) не забезпечують достатнього рівня ідентифікації та захисту від фальсифікації;
- системи на основі RFID або біометрії потребують дорогого обладнання.
- більшість рішень або занадто прості, або занадто складні та дорогі;
- відсутні зручні вітчизняні веб-рішення, де викладач може через звичайний браузер та веб-камеру миттєво відзначити відвідуваність, а адміністратор - отримати зведені звіти.

Порівняльний аналіз наведено у таблиці 1.2.

Таблиця 1.2

Порівняльний аналіз існуючих рішень

Критерій	Google Forms	Classter	Attendo	Розроблена система
Безкоштовність	+	-	-	+
Qr-Ідентифікація	-	-	+	+
Веб-Інтерфейс	+	+	+	+
Ролева модель	-	+	-	+
Звіти та аналітика	-	+	-	+
Експорт Excel	-	+	-	+
Без зовн. обладнання	+	+	+	+
Без хмарної залежності	-	-	+	+
Відкритий вихідний код	-	-	-	+
Підтримка укр.мови	+	-	-	+

1.3. Функціональні та нефункціональні вимоги до застосунку

1.3.1. Функціональні вимоги

Після аналізу предметної області та існуючих аналогів були сформульовані основні функціональні вимоги до системи обліку відвідуваності студентів за допомогою QR-кодів.

Система повинна працювати з користувачами і підтримувати автентифікацію. У системі є дві ролі користувачів: адміністратор і викладач. Користувачі можуть входити до системи через захищений механізм авторизації. Адміністратор має повний доступ до системи. Він може створювати викладачів, керувати студентами, заняттями і переглядати всі звіти. Викладач, в свою чергу,

може створювати свої заняття, сканувати QR-коди студентів і переглядати статистику тільки своїх занять.

Система також повинна підтримувати управління студентами, адміністратор може додавати нових студентів, редагувати наявні записи або їх видаляти. Для кожного студента зберігаються основні дані: повне ім’я, унікальний студентський код, група та електронна пошта. Після створення запису система автоматично генерує унікальний QR-код, який використовується для фіксації відвідуваності. Передбачено імпорт списків студентів із файлів CSV і Excel, а також можливість пошуку, сортування і фільтрації студентів за групами. Для зручності перегляду список студентів поділено на сторінки по 10 записів.

Окремим модулем який повинен бути це управління заняттями. Викладач або адміністратор можуть створювати заняття із зазначенням предмета, дати та групи. Викладач має доступ лише до своїх занять і може їх редагувати або видаляти. Адміністратор має доступ до всіх створених занять.

Для обліку відвідуваності реалізувати механізм сканування QR-кодів через камеру пристрою. Під час сканування система автоматично визначає студента і фіксує його присутність на занятті. Відмітка може бути «присутній» або «запізнився». Щоб уникнути повторних записів, система ігнорує повторне сканування одного й того самого QR-коду протягом трьох секунд. Крім того, передбачена можливість самостійного відзначення студентом через окрему публічну сторінку.

Система повинна надавати можливість перегляду звітів і статистики відвідуваності. Для кожного студента відображається кількість відвідувань, запізнень і пропусків, а також список останніх занять із відповідними статусами.

Також включена функція експорту звітів у форматі Excel. Для зручності аналізу даних використовується кольорове форматування: присутні студенти виділяються зеленим, відсутні – червоним, а запізнілі – жовтим.

1.3.2. Нефункціональні вимоги

Однією з основних нефункціональних вимог реалізувати безпеку системи за допомогою бібліотек. Паролі користувачів повинні зберігатися в хешованому вигляді за допомогою бібліотеки bcrypt із коефіцієнтом складності 10. Передача даних між клієнтом і сервером здійснюється через захищений протокол HTTPS. Для автентифікації використовується JWT: access-токен має термін дії 1 годину, а refresh-токен – 7 днів і зберігається в httpOnly cookie для підвищення захисту від XSS-атак. Щоб запобігти SQL-ін’єкціям, використовуються параметризовані запити до бази даних.

Важливою вимогою є продуктивність системи. Сторінки застосунку повинні відкриватися не довше ніж за 2 секунди при нормальному підключенні до мережі. Процес сканування QR-коду та реєстрація відвідування мають виконуватися менш ніж за 1 секунду. Система також повинна забезпечувати стабільну роботу щонайменше для 50 одночасно активних користувачів без помітного зниження швидкості.

Інтерфейс системи повинен бути простим і зрозумілим для користувачів без додаткового навчання. Сканування QR-кодів має працювати у звичайному браузері без необхідності встановлення додаткових програм або розширень. Система повинна коректно функціонувати в сучасних браузерах, зокрема Chrome, Firefox, Edge та Safari.

Для забезпечення надійності система повинна зберігати цілісність даних навіть у разі помилок або збоїв. У випадку проблем із мережею користувач повинен отримувати зрозумілі повідомлення про помилки. Також передбачено автоматичне оновлення access-токена через refresh без необхідності повторної авторизації користувача.

Архітектура системи повинна бути гнучкою та масштабованою, щоб у майбутньому можна було збільшувати кількість користувачів і обсяг даних без суттєвих змін у структурі застосунку.

1.4. Постановка задачі

На підставі виконаного аналізу предметної галузі, огляду існуючих програмних продуктів та сформульованих вимог визначено наступну постановку задачі кваліфікаційної роботи.

Необхідно розробити систему на тему «Система автоматизації обліку відвідувань» на базі Node.js та PostgreSQL, що дозволить:

- реєструвати відвідуваність студентів шляхом сканування QR-кодів;
- управляти даними студентів через зручний адміністративний інтерфейс;
- розмежовувати права доступу між адміністраторами та викладачами;
- формувати зведені звіти відвідуваності та вивантажувати їх у форматі Excel;
- забезпечити безпеку даних шляхом застосування сучасних практик автентифікації та захисту від веб-вразливостей.

Для вирішення поставленої задачі необхідно:

- проаналізувати існуючі рішення та визначити вимоги до системи.
- вибрати та обґрунтувати технологічний стек;
- спроєктувати структуру бази даних, архітектуру REST API та діаграми взаємодії компонентів;
- реалізувати серверну частину із повним набором ендпоінтів, системою автентифікації JWT та генерацією QR-кодів;
- розробити клієнтську частину у вигляді SPA з підтримкою ролей та функцією зчитування QR-кодів через камеру;
- впровадити функцію імпорту студентів із CSV/Excel та експорту звітів у Excel;
- протестувати систему та зафіксувати результати.

Практичним результатом роботи є готова до розгортання система, який не залежить від сторонніх хмарних сервісів, не потребує ліцензійних відрахувань та може бути встановлений на будь-якому сервері з підтримкою Node.js і PostgreSQL.

2. ВИБІР ТЕХНОЛОГІЧНОГО СТЕКУ

Вибір технологій є одним із ключових архітектурних рішень, що визначає можливості системи, продуктивність та зручність розробки і підтримки. У даному розділі обґрунтовано вибір кожної технології, що використовується у застосунку.

2.1. Серверна частина на основі Node.js та Express.js

Для реалізації серверної частини обрано платформу Node.js версії 24 – середовище виконання JavaScript поза браузером, побудоване на рушії V8 від Google [3]. Основними перевагами Node.js для даного проєкту є:

На користь Node.js свідчить кілька ключових переваг. По-перше, асинхронна неблокуюча модель вводу-виводу. Node.js використовує подієво-орієнтовану архітектуру. Це дозволяє ефективно обробляти велику кількість паралельних підключень без створення окремого потоку для кожного запиту. Це критично важливо для системи обліку відвідувань, де одночасно можуть надходити сканування від кількох викладачів. По-друге, єдина мова для сервера та клієнта. JavaScript використовується як на сервері, так і у браузері. Це спрощує розробку та дозволяє переносити код між частинами застосунку. По-третє, велика екосистема пакетів. Реєстр npm налічує понад 2 мільйони пакетів, що дозволяє швидко підключити готові рішення для будь-якого завдання, від генерації QR-кодів до роботи з Excel. І останнє простота розгортання – Node.js-застосунок легко запускається на будь-якому сервері чи хмарній платформі.

Як веб-фреймворк обрано “Express.js” версії 5 [4] – мінімалістичний і гнучкий фреймворк, що є де-факто стандартом для Node.js. Express.js надає зручний механізм маршрутизації запитів, middleware pipeline для обробки HTTP-запитів, вбудовану підтримку статичних файлів та просту інтеграцію з будь-якою базою даних.

Порівняння з альтернативними серверними технологіями наведено у таблиці 2.1.

Таблиця 2.1

Порівняння з альтернативними серверними технологіями

Критерій	Node.js	Django	Spring Boot	Laravel
Мова	JS/TS	Python	Java	PHP
Асинхронність	Нативна	Часткова	Є	Обмеж.
Продуктивність	Висока	Середня	Висока	Середня
npm екосистема	2M+ пак	pip	Maven	Composer
Простота старту	Висока	Висока	Низька	Висока
Ліцензія	MIT	BSD	APACHE	MIT

Node.js обрано з урахуванням таких факторів: відмінна продуктивність при роботі з I/O, висока швидкість розробки завдяки простоті мови та великій кількості готових пакетів, а також кросплатформність – однаково добре працює на Windows, Linux та macOS.

Структура серверного застосунку побудована за принципом розділення відповідальності (Separation of Concerns):

- src/config/: конфігурація підключення до бази даних;
- src/controllers/: обробники бізнес-логіки для кожного ресурсу;
- src/routes/: визначення маршрутів API;
- src/middleware/: проміжні обробники (автентифікація, авторизація);
- public/: статичні файли клієнтської частини;
- server.js: точка входу, ініціалізація та міграції.

2.2. Система управління базами даних PostgreSQL

Для зберігання даних обрано PostgreSQL версії 16 [5] – реляційну СУБД з відкритим вихідним кодом, яка вважається одним з найнадійніших рішень корпоративного рівня. PostgreSQL надає повну підтримку ACID-транзакцій, що є критично важливим для фінансово та академічно значущих даних про відвідуваність.

Таблиця 2.2

Параметри пулу з'єднань PostgreSQL

Параметр	Значення
Максимум з’єднань	10
Тайм-аут простою	30 000 ms
Тайм-аут підключення	2 000 ms
СУБД	PostgreSQL 16
База Даних	attendance_db

Ключові переваги PostgreSQL при реалізації проєкту:

- строга типізація даних, оскільки PostgreSQL суворо перевіряє типи під час вставки та оновлення. Це запобігає збереженню некоректних даних;
- підтримка складних запитів так як СУБД підтримує агрегатні функції, підзапити та FILTER-вирази, які використовуються для формування статистики відвідуваності;
- надійність та цілісність, адже зовнішні ключі, каскадні дії та обмеження забезпечують консистентність даних;
- масштабованість, оскільки PostgreSQL добре масштабується і підтримує роботу з мільйонами записів без суттєвого падіння продуктивності;
- відкритий вихідний код та безкоштовна ліцензія.

Для взаємодії з PostgreSQL використовується пакет «pg » (node-postgres) [6] – офіційний драйвер PostgreSQL для Node.js. Запити до бази виконуються через пул з'єднань (connection pool) із параметрами: максимум 10 з'єднань, час очікування 30 секунд, час очікування підключення 2 секунди. Всі запити до бази даних виконуються параметризовано, що унеможливлює SQL-ін'єкції.

2.3. Автентифікація та безпека (JWT)

Для автентифікації користувачів реалізовано систему на основі JSON Web Token (JWT) [7]. JWT – відкритий стандарт (RFC 7519), що визначає компактний і самодостатній спосіб безпечної передачі даних між сторонами у вигляді JSON-об'єкта.

У системі реалізовано дворівневу схему токенів:

Access-токен – це короткоживучий токен, який передається в заголовку Authorization кожного запиту. Він містить корисне навантаження: ідентифікатор користувача, email, роль та ім'я. Токен підписується секретним ключем алгоритмом HS256;

Refresh-токен – це довгоживучий токен, який зберігається у httpOnly cookie на клієнті. Коли строк дії access-токена закінчується, клієнт надсилає запит на оновлення, використовуючи refresh-токен. Refresh-токени зберігаються у базі даних, що дозволяє інвалідувати їх на сервері при виході з системи. Після кожного використання refresh-токен ротується то старий видаляється і генерується новий.

Зберігання refresh-токену у httpOnly cookie забезпечує захист від XSS-атак: JavaScript-код у браузері не має доступу до такого cookie. Дана схема відповідає рекомендаціям OWASP щодо безпечної автентифікації у веб-застосунках [8].

Хешування паролів виконується за допомогою бібліотеки «bcrypt» [9] із коефіцієнтом складності (cost factor) 10. Бібліотека «bcrypt» автоматично генерує унікальну сіль для кожного пароля, що унеможливлює атаки на основі rainbow tables та забезпечує унікальність хешів навіть для однакових паролів.

Таблиця 2.3

Параметри системи автентифікації

Параметр	Значення
Алгоритм підпису JWT	HS256
Строк дії access-токена	1 година
Строк дії refresh-токена	7 днів
Зберігання refresh-токена	httpOnly Cookie + БД
Алгоритм хешування паролів	bcrypt (cost factor 10)
Ротація refresh-токенів	При кожному використанні

2.4. Технологія QR-кодування

QR-код – двовимірний штрих-код, розроблений у 1994 році компанією Denso Wave [10]. Порівняно зі звичайними штрих-кодами QR-коди мають значно більшу інформаційну ємність (до 7089 цифрових символів або 4296 алфавітно-цифрових символів), підтримують корекцію помилок, що дозволяє зчитувати пошкоджені коди, та можуть бути зчитані будь-яким пристроєм із камерою.

Для генерації QR-кодів використовується npm-пакет «qrcode» [11] - бібліотека, що реалізує кодування даних у QR-символіку відповідно до стандарту ISO/IEC 18004. Бібліотека підтримує генерацію у різних форматах: PNG, SVG, Canvas та DataURL (base64). У застосунку обрано формат DataURL, що дозволяє зберігати зображення QR-коду безпосередньо в базі даних у вигляді рядка та передавати його клієнту без необхідності зберігати файли на диску.

Дані, закодовані у QR-коді студента, мають формат JSON:

{"type": "student", "code": "ST001"}, де "code" – унікальний студентський ідентифікатор. QR-код генерується одноразово при створенні або імпорті студента та зберігається у базі даних.

Для зчитування QR-кодів на клієнтській стороні використовується бібліотека «jsQR» [12] версії 1.4.0, це спеціалізований модуль, що реалізує аналіз растрових зображень з відеопотоку камери та декодування QR-символів у реальному часі.

Перевагою «jsQR» є відсутність залежностей, виконання виключно на стороні браузера та підтримка сучасних браузерів.

2.5. Клієнтська частина застосунку

Клієнтська частина реалізована у вигляді Single Page Application на чистому JavaScript (Vanilla JS) без використання фреймворків на зразок React або Vue.js. Такий підхід обрано з метою:

- відсутності додаткових залежностей та складного процесу збірки;
- невеликого розміру клієнтського коду;
- повного контролю над поведінкою застосунку;
- простоти підтримки та модифікації.

Для стилізації інтерфейсу використовуються кастомні CSS-змінні та флексбокс, що забезпечують адаптивний дизайн без підключення зовнішніх CSS-фреймворків.

Для побудови графіків та діаграм аналітики відвідуваності використовується бібліотека «Chart.js» [13] версії 4.4.0 – бібліотека для візуалізації статистичних даних у браузері, що реалізує побудову кругових, стовпчастих та лінійних діаграм із підтримкою анімації та інтерактивних підказок.

Для іконок інтерфейсу підключено бібліотеку «Lucide Icons» [14] – набір відкритих SVG-іконок із мінімалістичним дизайном.

Навігація між розділами застосунку реалізована без перезавантаження сторінки шляхом перемикання видимості секцій. Кожен розділ автоматично оновлює дані при переключенні на нього, що забезпечує актуальність відображуваної інформації.

Таблиця 2.4

Клієнтські бібліотеки

Бібліотека	Версія	Призначення
«jsQR»	1.4.0	Декодування QR-кодів з камери
«Chart.js»	4.4.0	Побудова діаграм та графіків
«Lucide icons»	Latest	SVG-іконки інтерфейсу

Для зчитування QR-кодів через камеру використовується Web API: navigator.mediaDevices.getUserMedia() для отримання доступу до камери та requestAnimationFrame() для обробки відеокадрів у реальному часі. Кожен кадр передається у «jsQR », яка повертає результат декодування або null. При успішному розпізнаванні автоматично надсилається запит до сервера для реєстрації відвідування. Для уникнення повторної реєстрації одного студента реалізовано захист від дублікатів: повторне сканування того самого коду ігнорується протягом 3 секунд.

Сервер-клієнтська взаємодія реалізована через REST API з форматом обміну даними JSON. Всі запити, крім публічних ендпоінтів (сканування для студентів та вхід), вимагають наявності Bearer-токена у заголовку Authorization. При отриманні відповіді з кодом 401 клієнт автоматично намагається оновити access-токен через refresh-ендпоінт та повторює запит – процес «silent refresh» є прозорим для користувача.

3. ПРОЄКТУВАННЯ СИСТЕМИ

3.1. Загальна архітектура системи

Архітектура системи побудована за принципом клієнт-серверної взаємодії через REST API. Система складається з трьох основних шарів: клієнтський, серверний, який реалізовано за допомогою платформи Node.js та фреймворку Express.js, а для зберігання даних використовується СУБД PostgreSQL.

Серверна частина організована за багаторівневим принципом. Рівень маршрутизації. Він визначає ендпоінти API та пов'язує їх з відповідними обробниками. Для кожного ресурсу створено окремий файл маршрутів: auth.js, students.js, lessons.js, attendance.js, users.js. Маршрути передають запити через middleware автентифікації та авторизації до контролерів.

Рівень middleware. Включає:

- authMiddleware, перевіряє наявність та валідність JWT access-токена у заголовку Authorization;
- adminOnly, перевіряє, чи має аутентифікований користувач роль «адміністратор».

Рівень контролерів. Містить бізнес-логіку обробки запитів. Кожен контролер відповідає за окремий ресурс системи:

- authController.js: реєстрація, вхід, оновлення та відкликання токенів;
- studentController.js: CRUD-операції над студентами, генерація QR-кодів, імпорт із файлів, статистика;
- lessonController.js: управління заняттями;
- attendanceController.js: сканування QR-кодів, звіти, експорт у Excel;
- userController.js: управління обліковими записами.

Рівень доступу до даних. Всі запити до PostgreSQL виконуються через спільний пул з'єднань, що забезпечує ефективне управління ресурсами бази даних.

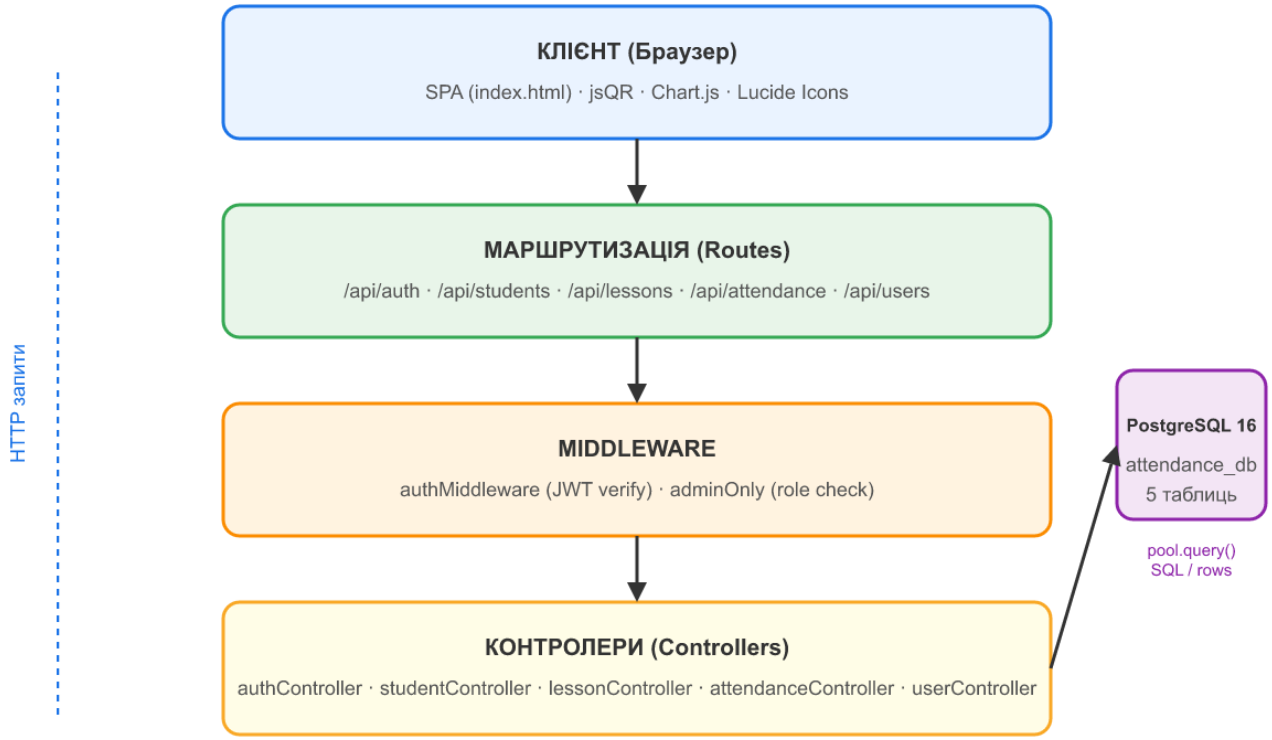


Рис. 3.1 Архітектура системи

REST API реалізовано за принципами RESTful архітектури, де ресурси ідентифікуються іменниками у множині, наприклад, /api/students, /api/lessons, /api/attendance. Дії визначаються HTTP-методами, такими як GET для читання, POST для створення, PUT для оновлення, DELETE для видалення. Відповіді повертаються у форматі JSON. Статусні коди відповідають семантиці операції: 200, 201, 400, 401, 403, 404, 500.

3.2. Проєктування бази даних

3.2.1. ER-модель даних

База даних attendance_db містить п'ять основних сутностей. В таблиці 3.1 представлено всі таблиці.

Таблиця 3.1

Таблиці бази даних системи

Таблиця	Призначення
users	Облікові записи адміністраторів та викладачів, що мають доступ до адміністративного інтерфейсу
students	Дані студентів; кожен студент має унікальний код та прив'язаний QR-код для ідентифікації
lessons	Заняття, прив'язані до конкретного викладача та групи
attendance	Відмітки відвідування: студент, заняття, статус та час сканування
refresh_tokens	Активні refresh-токени для кожного облікового запису

Зв'язки між сутностями:

- users (1) до (N) lessons;
- students (1) до (N) attendance;
- lessons (1) до (N) attendance;
- users (1) до (N) refresh_tokens.

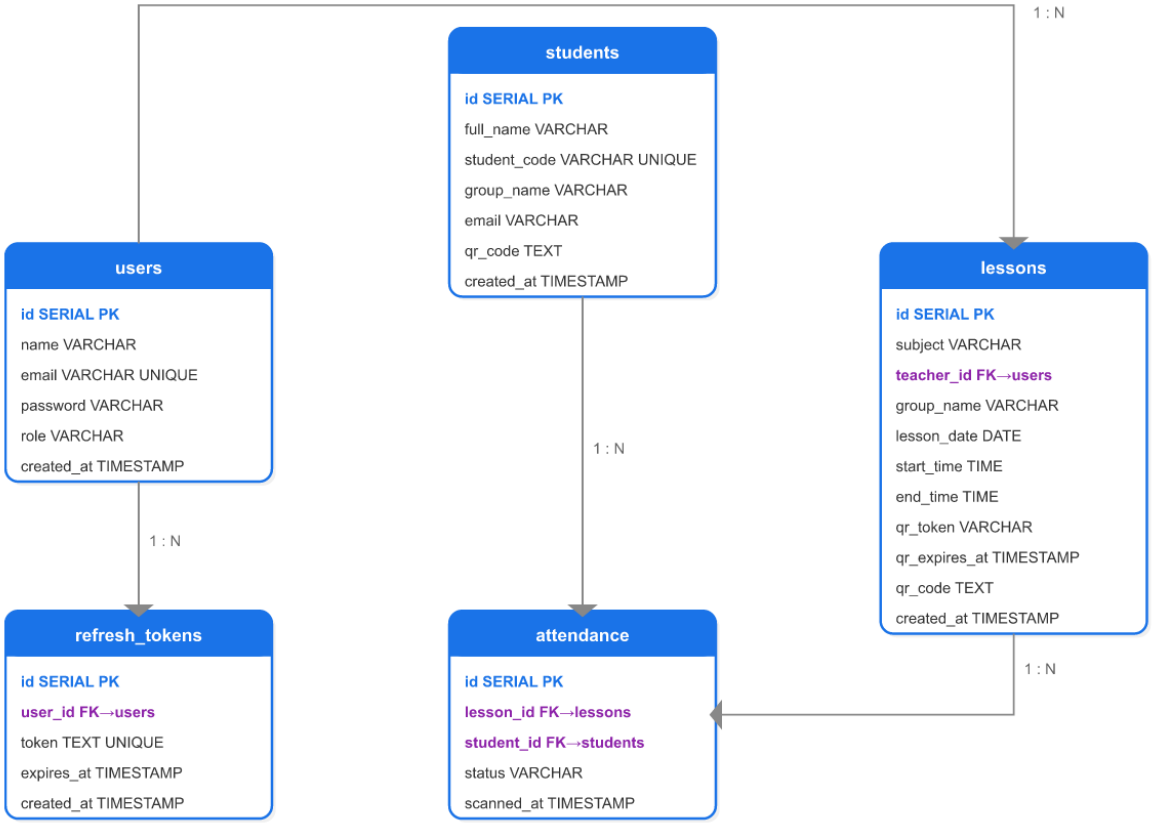


Рис. 3.2 Er-діаграма

3.2.2. Логічна структура таблиць

Таблиця 3.2

Структура таблиці users

Поле	Тип	Опис
id	SERIAL PRIMARY KEY	Унікальний ідентифікатор
email	VARCHAR(255) UNIQUE	Електронна пошта
password	TEXT NOT NULL	Хеш пароля
role	VARCHAR(20) DEFAULT 'teacher'	Роль: admin або teacher
name	VARCHAR(255)	Повне ім'я
created_at	TIMESTAMP DEFAULT NOW	Дата реєстрації

Таблиця users зберігає облікові записи системи. Поле role визначає рівень доступу користувача – викладач (teacher) або адміністратор (admin). Пароль зберігається у вигляді хешу бібліотеки bcrypt. Поле email є унікальним і використовується як ідентифікатор при автентифікації.

Таблиця 3.3

Структура таблиці students

Поле	Тип	Опис
id	SERIAL PRIMARY KEY	Унікальний ідентифікатор
full_name	VARCHAR(255) NOT NULL	Повне ім'я студента
student_code	VARCHAR(50) UNIQUE	Унікальний код студента
group_name	VARCHAR(100)	Назва групи
email	VARCHAR(255)	Електронна пошта
qr_code	TEXT	QR-код у форматі base64
created_at	TIMESTAMP DEFAULT NOW	Дата додавання

Таблиця students містить дані про студентів, що перебувають в системі відвідуваності. Кожен студент має унікальний student_code, який також є ідентифікатором у QR-коді. Поле qr_code зберігає згенероване зображення у форматі base64 для подальшого відображення. Поле group_name дозволяє групувати студентів за навчальними групами. Це використовується коли перевіряється чи належить студент до заняття. Система порівнює групу студента з групою, вказаною у занятті. Якщо є розбіжність, система повертає відповідь із кодом 403. Поле email є необов'язковим і може використовуватися для подальшого розширення функціоналу системи. Часова мітка created_at фіксує момент додавання студента до системи та заповнюється автоматично.

Таблиця 3.4

Структура таблиці lessons

Поле	Тип	Опис
id	SERIAL PRIMARY KEY	Унікальний ідентифікатор
subject	VARCHAR(255) NOT NULL	Назва предмету
lesson_date	DATE NOT NULL	Дата заняття
start_time	TIME NOT NULL	Час початку заняття
end_time	TIME NOT NULL	Час закінчення заняття
teacher_id	INT FK - users.id	Викладач-власник заняття
group_name	VARCHAR(100)	Група (для фільтрації)
qr_token	TEXT	Токен для QR-коду заняття
qr_expires_at	TIMESTAMP	Термін дії QR-коду
qr_code	TEXT	QR-код заняття у форматі base64
created_at	TIMESTAMP DEFAULT NOW	Дата створення запису

Таблиця lessons описує навчальні заняття. Зв’язок із таблицею users через зовнішній ключ teacher_id визначає викладача, який належить до заняття. Поля start_time та end_time фіксують часомежі заняття. Ці дані використовують, щоб автоматично визначити статус відвідування. Якщо студент сканує QR-код після початку заняття, система надає статус «запізнення». Поля qr_token та qr_expires_at забезпечують тимчасову дійсність QR-коду заняття. Після закінчення терміну сканування QR-код стає недоступним. Поле group_name дозволяє обмежити заняття для конкретної навчальної групи. Поле qr_code зберігає зображення QR-коду у форматі base64 для відображення в інтерфейсі викладача.

Таблиця 3.5

Структура таблиці attendance

Поле	Тип	Опис
id	SERIAL PRIMARY KEY	Унікальний ідентифікатор
student_id	INT FK – students.id	Студент
lesson_id	INT FK – lessons.id	Заняття
status	VARCHAR(20)	present / late
scanned_at	TIMESTAMP DEFAULT NOW	Час реєстрації відвідування

Таблиця attendance фіксує відвідування конкретним студентом конкретного заняття. Комбінація полів lesson_id та student_id є унікальною, що не дозволяє дублювати записи. Повторне сканування одного студента на тому самому занятті не створює нового запису. Поле status набуває значення present або late залежно від часу реєстрації відносно початку заняття. Якщо момент сканування перевищує start_time заняття на визначений порог хвилин, система автоматично присвоює статус запізнення. Поле scanned_at заповнюється автоматично та слугує основою для формування звітів про відвідуваність і аналітики.

Таблиця 3.6

Структура таблиці refresh_tokens

Поле	Тип	Опис
id	SERIAL PRIMARY KEY	Унікальний ідентифікатор
user_id	INT FK – users.id	Власник токена
token	TEXT UNIQUE NOT NULL	Значення refresh-токена
expires_at	TIMESTAMP NOT NULL	Термін дії
created_at	TIMESTAMP DEFAULT NOW	Дата створення

Таблиця refresh_tokens забезпечує ротацію токенів автентифікації. Кожен запис пов'язаний з конкретним користувачем через user_id і має термін дії (expires_at). При кожному успішному оновленні access-токена старий refresh-токен видаляється і замінюється новим, що запобігає його повторному використанню. Поле token має обмеження UNIQUE, що гарантує відсутність колізій між активними сесіями різних користувачів. Поле created_at фіксує момент видачі токена і може використовуватися для аудиту активних сесій.

Міграції виконуються автоматично при запуску сервера – server.js перевіряє наявність таблиць і створює їх при необхідності через CREATE TABLE IF NOT EXISTS, що забезпечує ідемпотентне розгортання.

3.3. Моделювання функціональних вимог

3.3.1. Діаграма варіантів використання

У системі визначено два актори: Адміністратор та Викладач. Адміністратор розширює можливості Викладача, тобто має доступ до всіх функцій викладача плюс адміністративні функції.

Варіанти використання для Викладача:

- вхід до системи;
- перегляд списку своїх занять;
- створення нового заняття;
- редагування/видалення свого заняття;
- сканування QR-коду студента;
- перегляд звіту відвідуваності заняття;
- перегляд статистики конкретного студента.

Варіанти використання лише для Адміністратора:

- реєстрація нових користувачів (викладачів);
- перегляд та управління всіма студентами;
- додавання/редагування/видалення студента;
- імпорт студентів із CSV або Excel;

- перегляд та управління заняттями всіх викладачів;
- управління обліковими записами;
- експорт звітів у Excel.

Публічні варіанти використання без автентифікації – Студент сканує QR-код заняття для самостійного відмічання

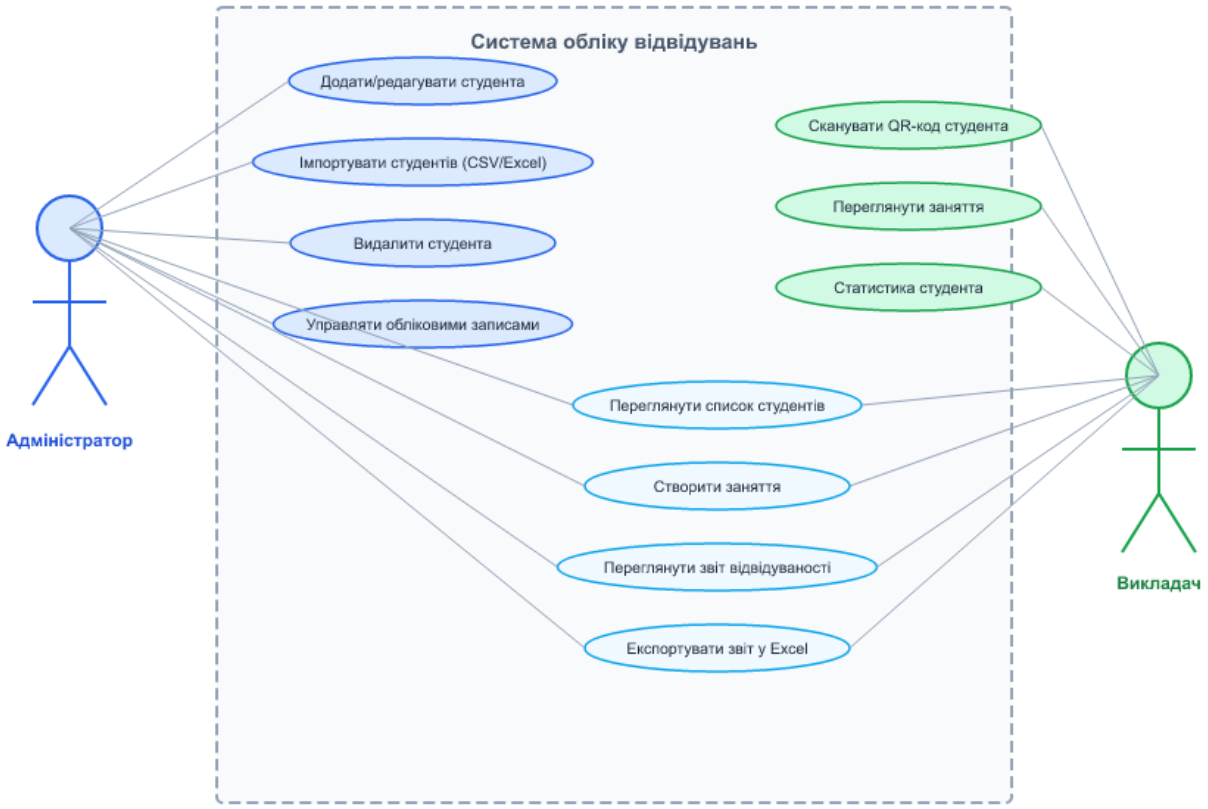


Рис. 3.3 Діаграма варіантів використання

3.3.2. Діаграми послідовності

В даному підрозділ буде розглянуто певні сценарії використання: реєстрація відвідуваності через QR-код, Оновлення access-токена, Імпорт студентів із Excel

Першим процесом який буде розглянуто є реєстрація відвідуваності через QR-код. Реєстрацію ініціює викладач, він відкриває сторінку заняття у веб-браузері і активує режим сканування за допомогою відповідного елемента управління. Браузер запитує дозвіл на доступ до камери. Після його отримання починається безперервна обробка відеопотоку за допомогою бібліотеки «jsQR». Коли студент підносить свій QR-код до камери, бібліотека декодує графічний ідентифікатор.

Вміст коду перетворюється на текстовий рядок, що містить унікальний код студента. Перед відправкою запиту на сервер клієнтська частина перевіряє час, що пройшов з моменту останнього сканування цього ж коду, щоб уникнути дублювання відміток. Якщо забезпечено відповідність обмеженню (не менше 3 секунд), клієнт формує і надсилає HTTP-запит POST /api/attendance/scan з корисним навантаженням {lesson_token, student_code}. На сервері проміжний обробник authMiddleware перевіряє JWT-токен. Після цього контролер перевіряє, чи існує студент з зазначеним кодом у базі даних і чи немає дублікатів відміток. Це забезпечується унікальним обмеженням (UNIQUE constraint). Якщо всі перевірки проходять успішно, запис вставляється в таблицю attendance, а клієнту повертається структурована відповідь {success: true, student: {name, code}}. Результат операції відображається користувачеві у формі спливаючого сповіщення з іменем зареєстрованого студента.

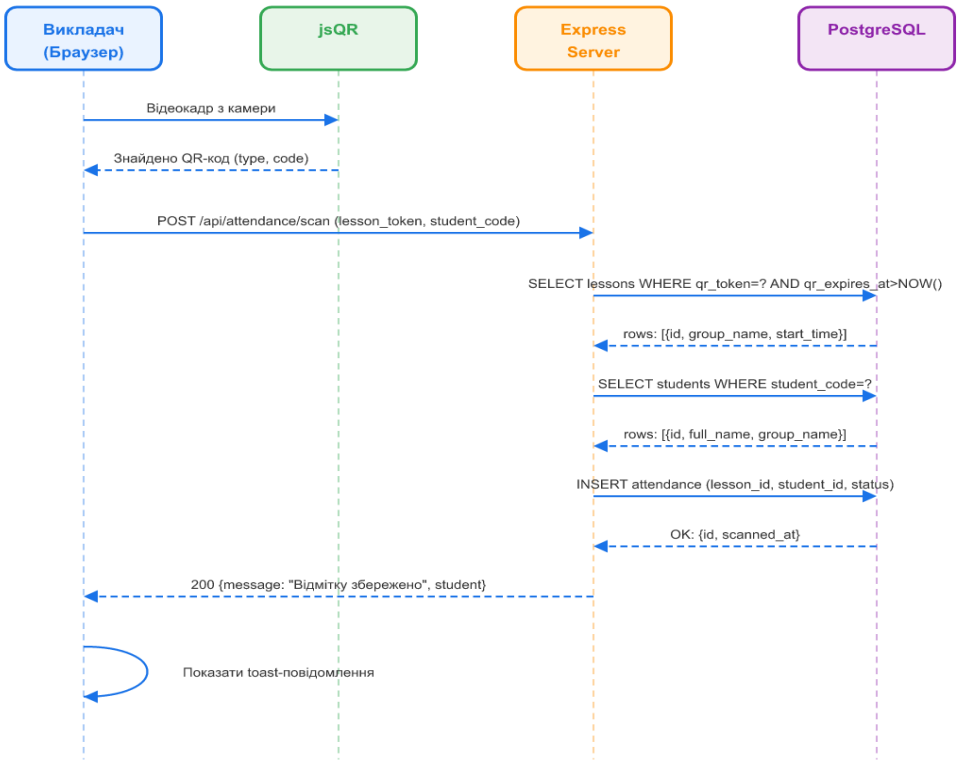


Рис. 3.4 Діаграма послідовності: реєстрація відвідуваності через QR-код

Другим процесом буде розглянуто є Оновлення access-токена. Механізм автоматичного оновлення токена активується у разі отримання клієнтом відповіді сервера зі статусом 401 Unauthorized на запит із простроченим access-токеном. Клієнтська функція apiReq(), що є єдиною точкою виконання HTTP-запитів у системі, перехоплює зазначену відповідь та без будь-якого втручання з боку користувача ініціює допоміжний запит POST /api/auth/refresh, передаючи refresh-токен через механізм httpOnly cookie. Сервер виконує верифікацію отриманого токена шляхом пошуку відповідного запису у базі даних із перевіркою терміну його дії. У разі успішної валідації здійснюється анулювання наявного refresh-токена та генерація нової пари токенів доступу. Оновлений refresh-токен встановлюється у захищений httpOnly cookie, тоді як новий access-токен передається клієнту у тілі відповіді. Клієнтська частина зберігає отриманий токен у сховищі localStorage та автоматично повторює початковий запит із оновленими обліковими даними. Описаний процес є повністю прозорим для кінцевого користувача та не спричиняє переривання роботи з інтерфейсом системи.

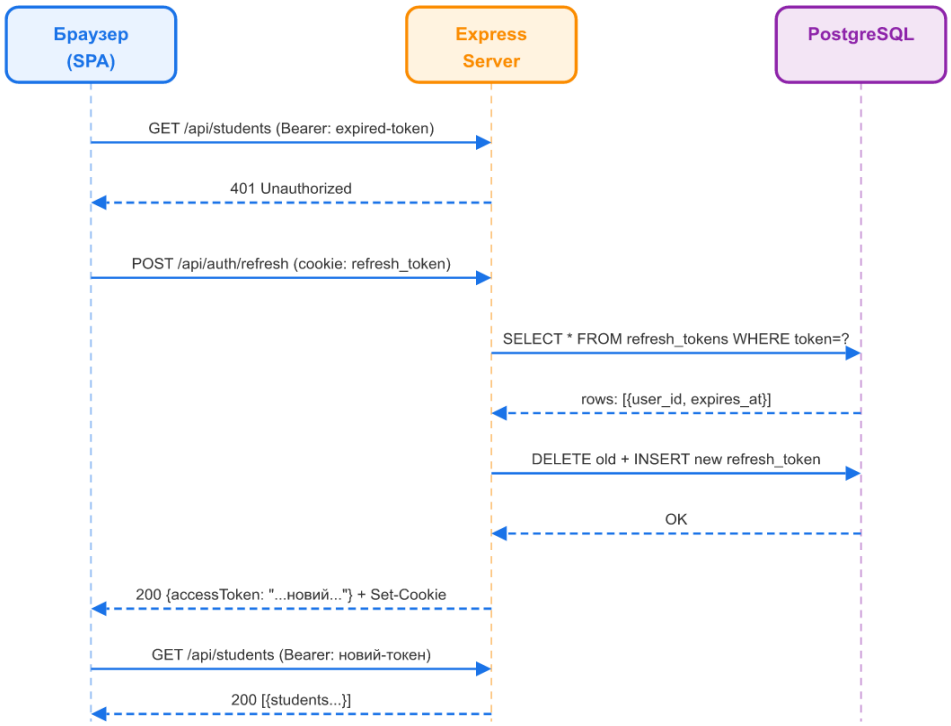


Рис. 3.5 Діаграма послідовності: Silent Refresh access-токена

І третім процесом, який буде розглянуто, є Імпорт студентів із Excel. Процедура масового імпорту даних студентів ініціюється адміністратором системи шляхом відкриття відповідного модального вікна та вибору файлу у форматі .xlsx. Після підтвердження операції клієнтська частина формує об'єкт FormData та надсилає його у складі HTTP-запиту POST /api/students/import-excel із типом вмісту multipart/form-data. На стороні сервера обробка завантаженого файлу здійснюється бібліотекою «multer» із використанням стратегії збереження у пам'яті процесу (memoryStorage), що виключає необхідність запису тимчасових файлів на диск. Отриманий буфер передається бібліотеці «xlsx», яка виконує синтаксичний аналіз першого аркуша книги та формує масив об'єктів-рядків; ключі атрибутів при цьому нормалізуються до нижнього регістру для забезпечення стійкості до варіативності форматування вхідних даних. Для кожного рядка виконується перевірка унікальності значення student_code у базі даних: за відсутності збігу генерується персональний QR-код студента та виконується вставка нового запису. За результатами опрацювання всього масиву даних сервер повертає об'єкт зі статистикою операції {added, skipped, errors[]}, що відображається користувачеві у вигляді структурованого підсумкового повідомлення. Список студентів в інтерфейсі оновлюється автоматично без необхідності перезавантаження сторінки.

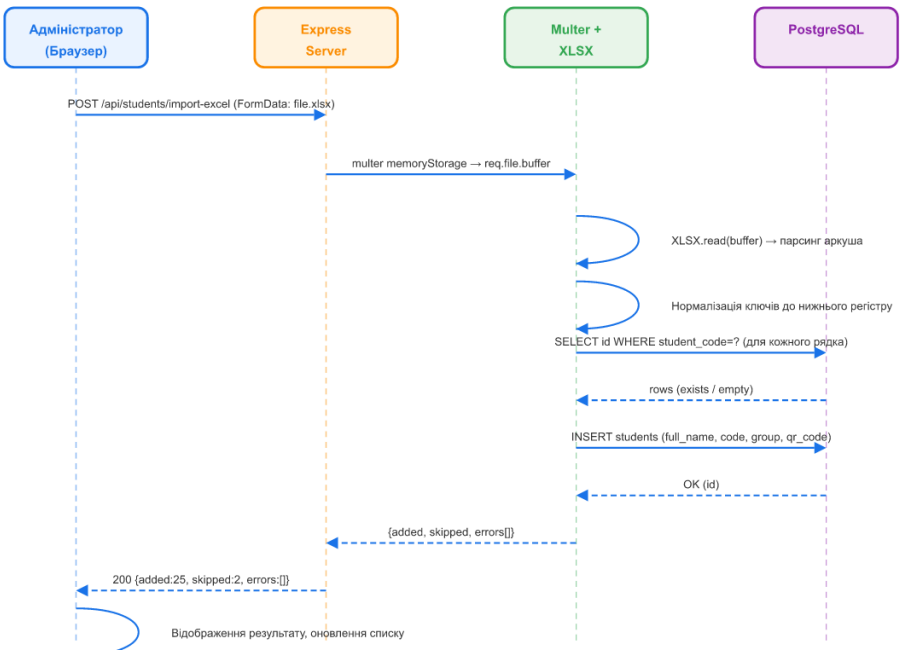


Рис. 3.6 Діаграма послідовності: Імпорт студентів із Excel

3.4. Проєктування інтерфейсу користувача

Інтерфейс застосунку реалізований у вигляді Single Page Application. Після входу в систему користувач бачить головний екран із боковою навігаційною панеллю (sidebar) та основним робочим простором.

Бокова навігація містить розділи залежно від ролі користувача:

- панель (всі ролі): загальна статистика: кількість студентів, занять та відміток;
- студенти (всі ролі): список студентів із пошуком, фільтром та пагінацією;
- заняття (всі ролі): список занять із можливістю сканування;
- відвідуваність (всі ролі): звіти та фільтрація;
- звіт (всі ролі): детальний звіт із функцією експорту Excel;
- користувачі (лише адмін): управління обліковими записами.

3.4.1. Сторінка автентифікації

Перша сторінка застосунку – форма входу, що містить поля «Email» та «Пароль» і кнопку «Увійти». Неавторизованим користувачам доступна лише ця сторінка – всі інші маршрути захищені перевіркою JWT-токена.

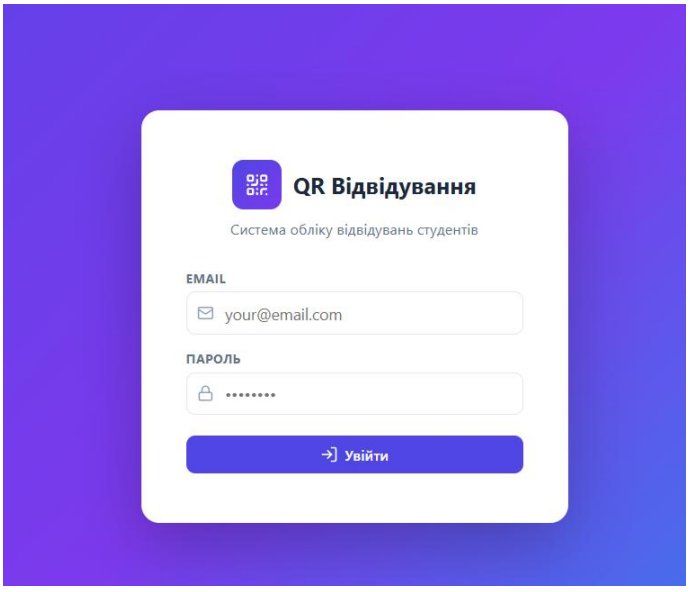


Рис. 3.7 Сторінка входу до систем

3.4.2. Головна панель (Dashboard)

Головна панель відображає картки з ключовими показниками: загальна кількість студентів, кількість занять та загальна кількість відміток. Також показується кругова діаграма розподілу відвідуваності, побудована за допомогою бібліотеки «Chart.js» – засобу візуалізації статистичних даних.

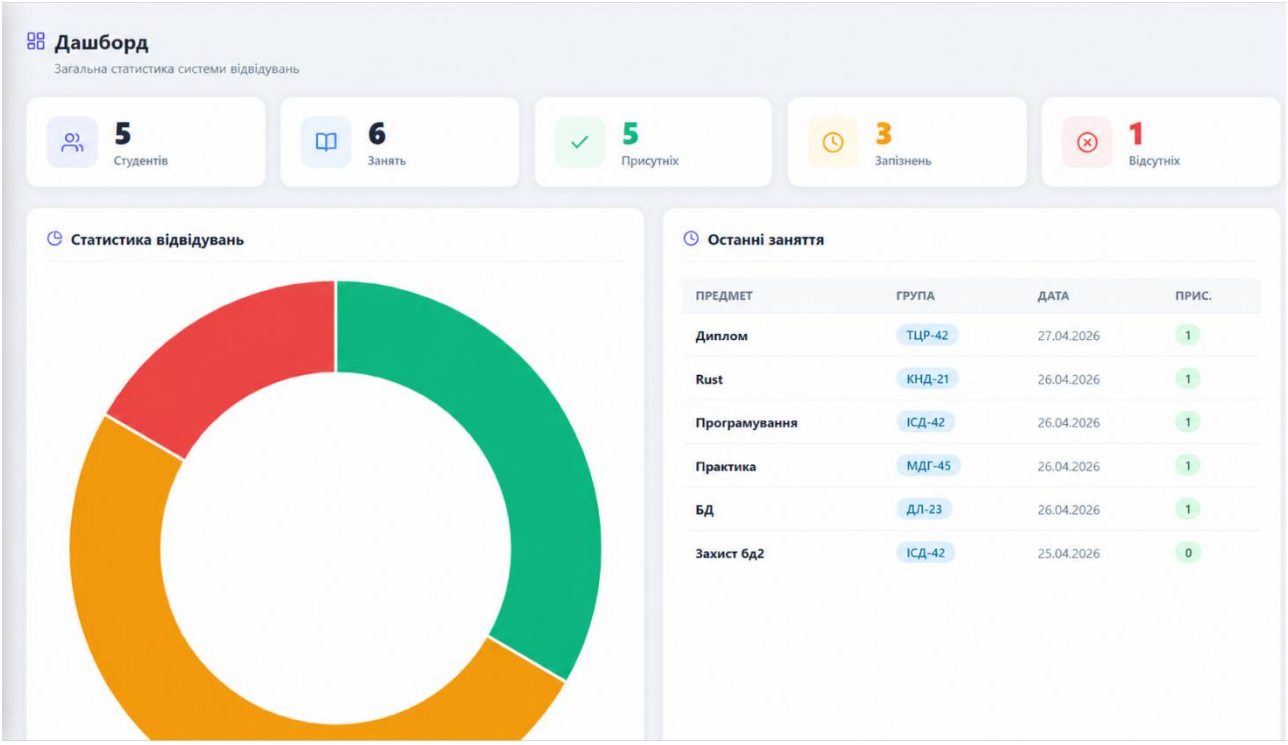


Рис. 3.8 Головна панель із статистикою

3.4.3. Розділ «Студенти»

Таблиця з переліком студентів, рядки якої містять ПІБ, код, групу, email та кнопки дій (перегляд QR, редагування, видалення). Над таблицею – поле пошуку, фільтр за групою, кнопки «Додати», «Імпорт CSV», «Імпорт Excel», «Скачати QR». Пагінація: 10 студентів на сторінку.

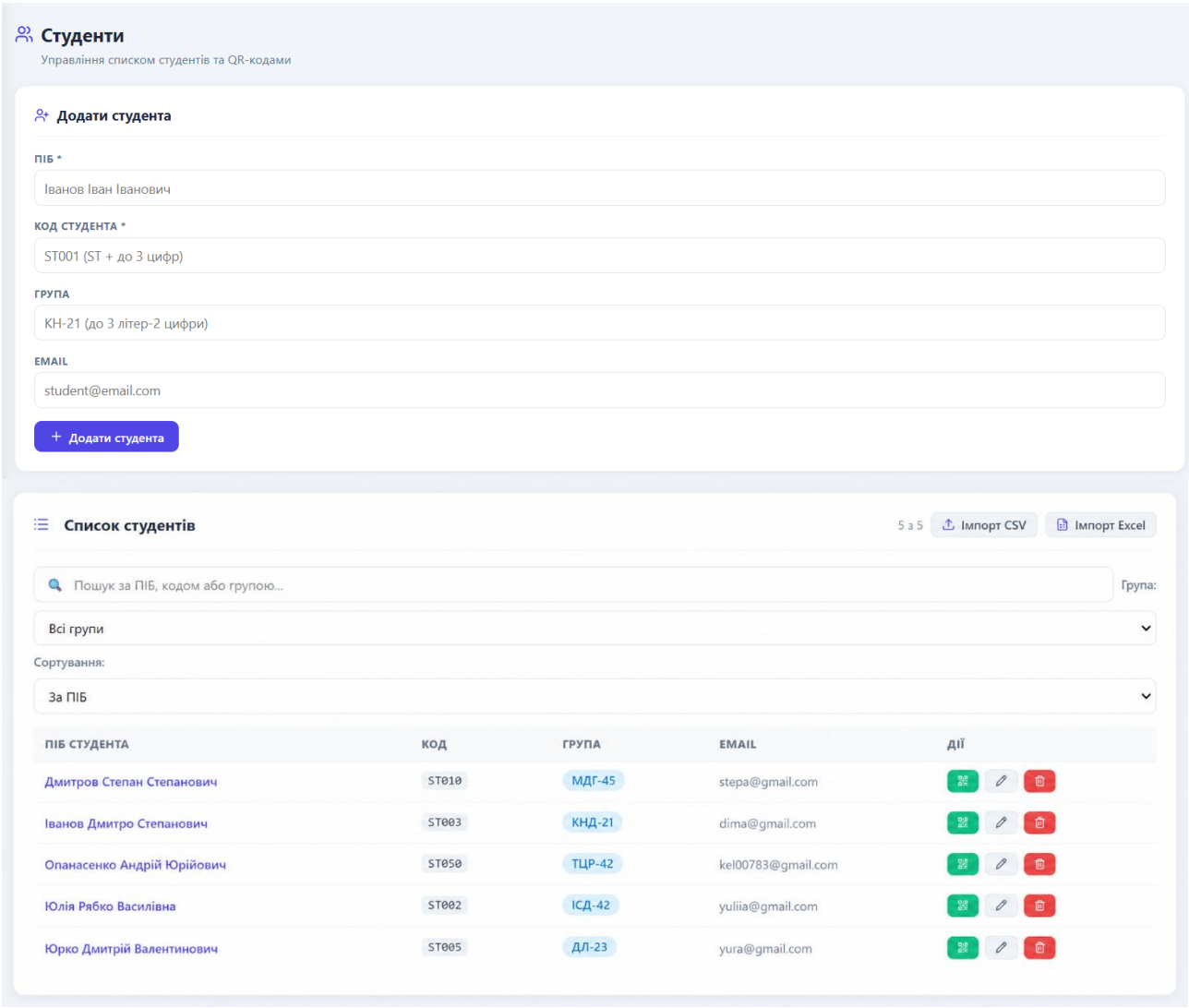


Рис. 3.9 Розділ студенти

При натисканні кнопки «QR» відкривається модальне вікно з QR-кодом студента, готовим для друку або збереження.

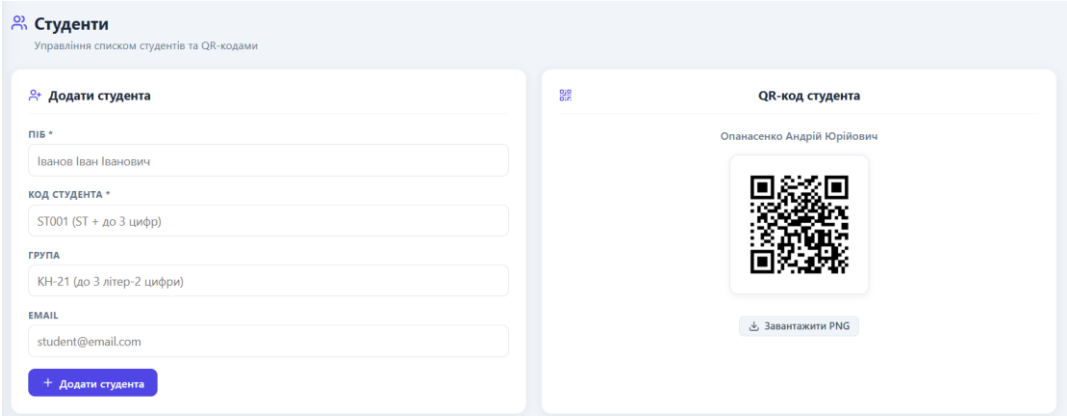


Рис. 3.10 Модальне вікно з QR-кодом студента

3.4.4. Розділ «Заняття» та сканування

Картки занять із назвою предмету, датою та групою. Кожна картка має кнопки: «Сканувати» та «Редагувати», «Видалити». Для адміністратора відображаються заняття всіх викладачів, для викладача – лише власні.

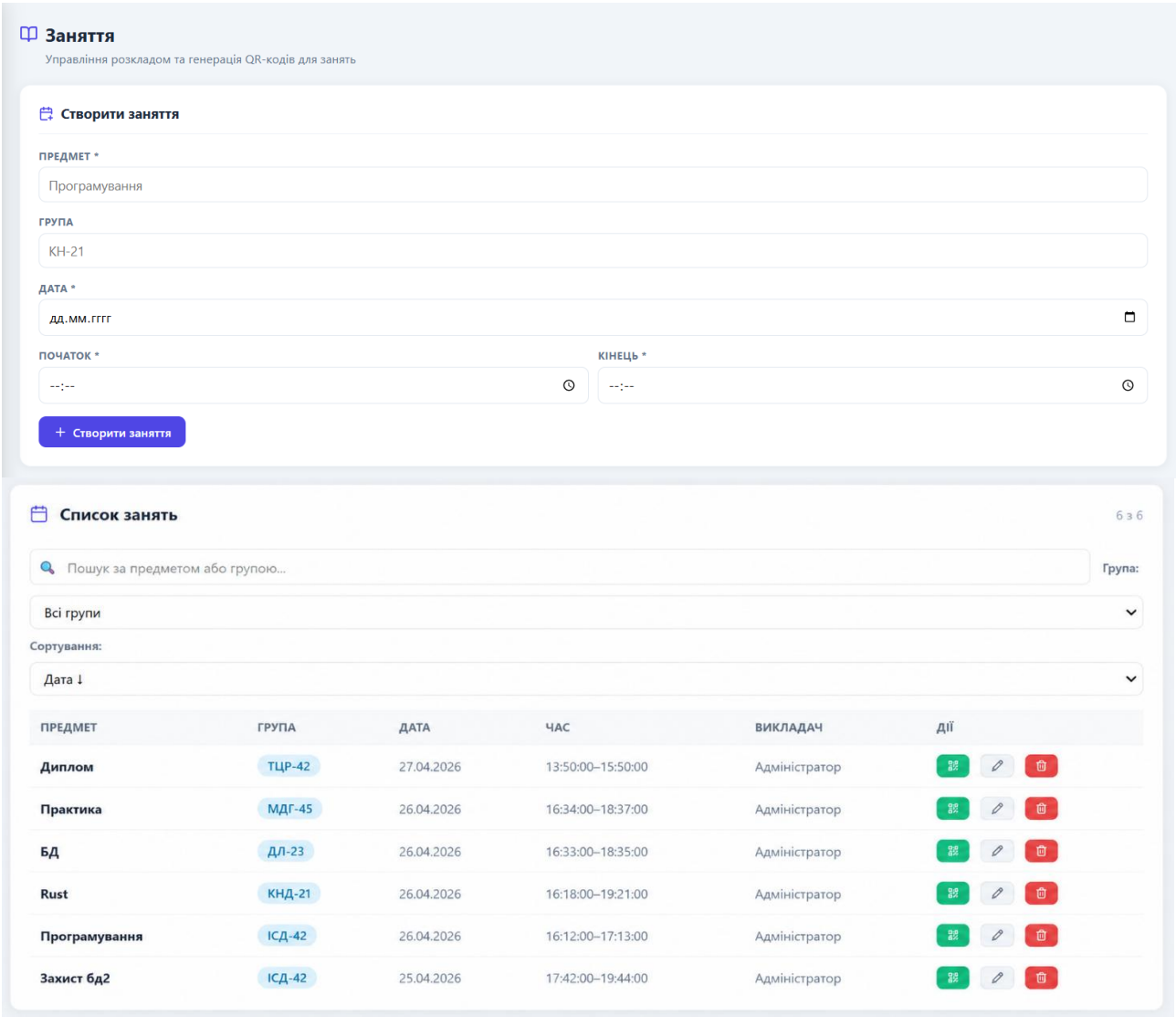


Рис. 3.11 Розділ Заняття

Модальне вікно сканування містить відеопотік із камери та лічильник відсканованих студентів. Під час сканування відображаються toast-повідомлення з ім'ям студента при кожному успішному скануванні.

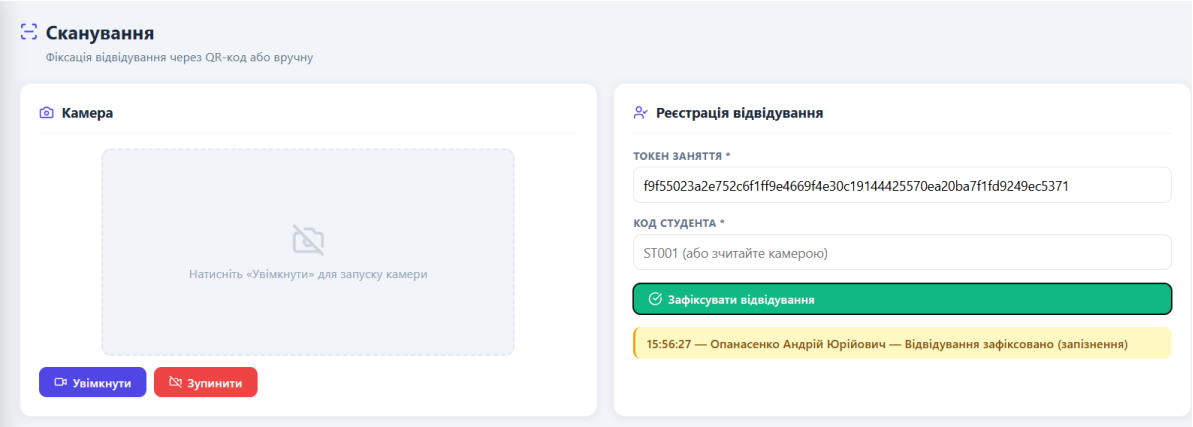


Рис. 3.12 Модальне вікно сканування QR-кодів

3.4.5. Розділ «Звіт» та експорт

Форма фільтрації за заняттям та групою. Таблиця результатів із кольоровим маркуванням рядків: зелений – присутній, жовтий – запізнився, червоний – відсутній. Кнопка «Експорт Excel» вивантажує відформатований звіт у файл .xlsx.

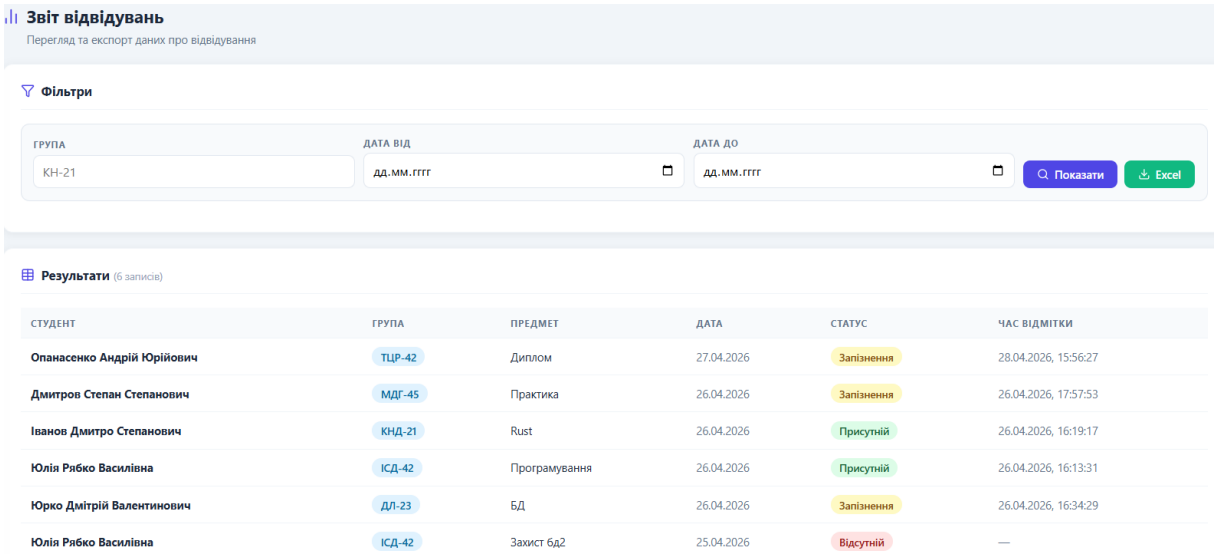


Рис. 3.13 Розділ Звіт відвідувань

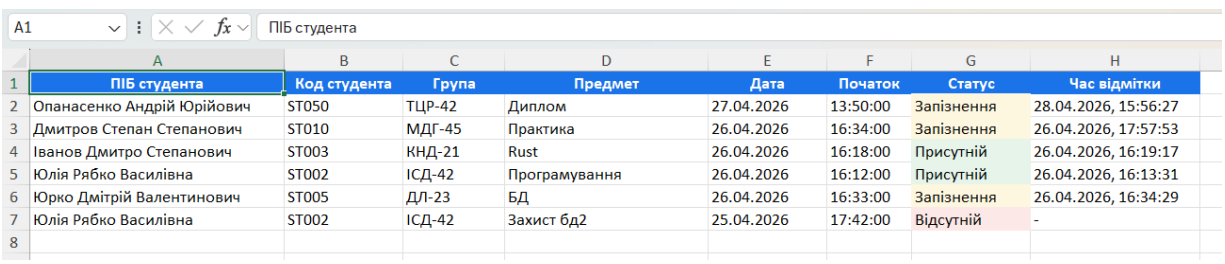


Рис. 3.14 Згенерований Excel-файл звіту з кольоровим форматуванням

4. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ

4.1. Реалізація серверної частини

4.1.1. Структура серверного застосунку

Точкою входу серверного застосунку є файл server.js, що виконує такі функції при запуску:

- завантаження змінних середовища за допомогою бібліотеки «dotenv», що забезпечує ізоляцію конфігураційних параметрів від вихідного коду шляхом зчитування .env-файлу;
- ініціалізація Express-застосунку та підключення проміжних обробників та роздача статичних файлів;
- перевірка підключення до бази даних (SELECT 1);
- автоматичне виконання міграцій – CREATE TABLE IF NOT EXISTS для всіх таблиць;
- реєстрація маршрутів API;
- запуск HTTP-сервера на порту 3000.

Табл 4.1

Серверні залежності

Пакет	Версія	Призначення
«express	5.2.1	HTTP-фреймворк
«pg»	8.20.0	Драйвер PostgreSQL
«bcrypt»	6.0.0	Хешування паролів
«jsonwebtoken»	9.0.3	Генерація та перевірка JWT
«cookie-parser»	1.4.7	Читання httpOnly cookie
«cors»	2.8.6	CORS-заголовки
«qrcode»	1.5.4	Генерація QR-кодів
«exceljs»	4.4.0	Генерація Excel-файлів
«multer»	2.1.1	Завантаження файлів
«xlsx»	0.18.5	Читання Excel-файлів
«dotenv»	17.4.2	Управління змінними середовища

Залежності проєкту визначені у package.json

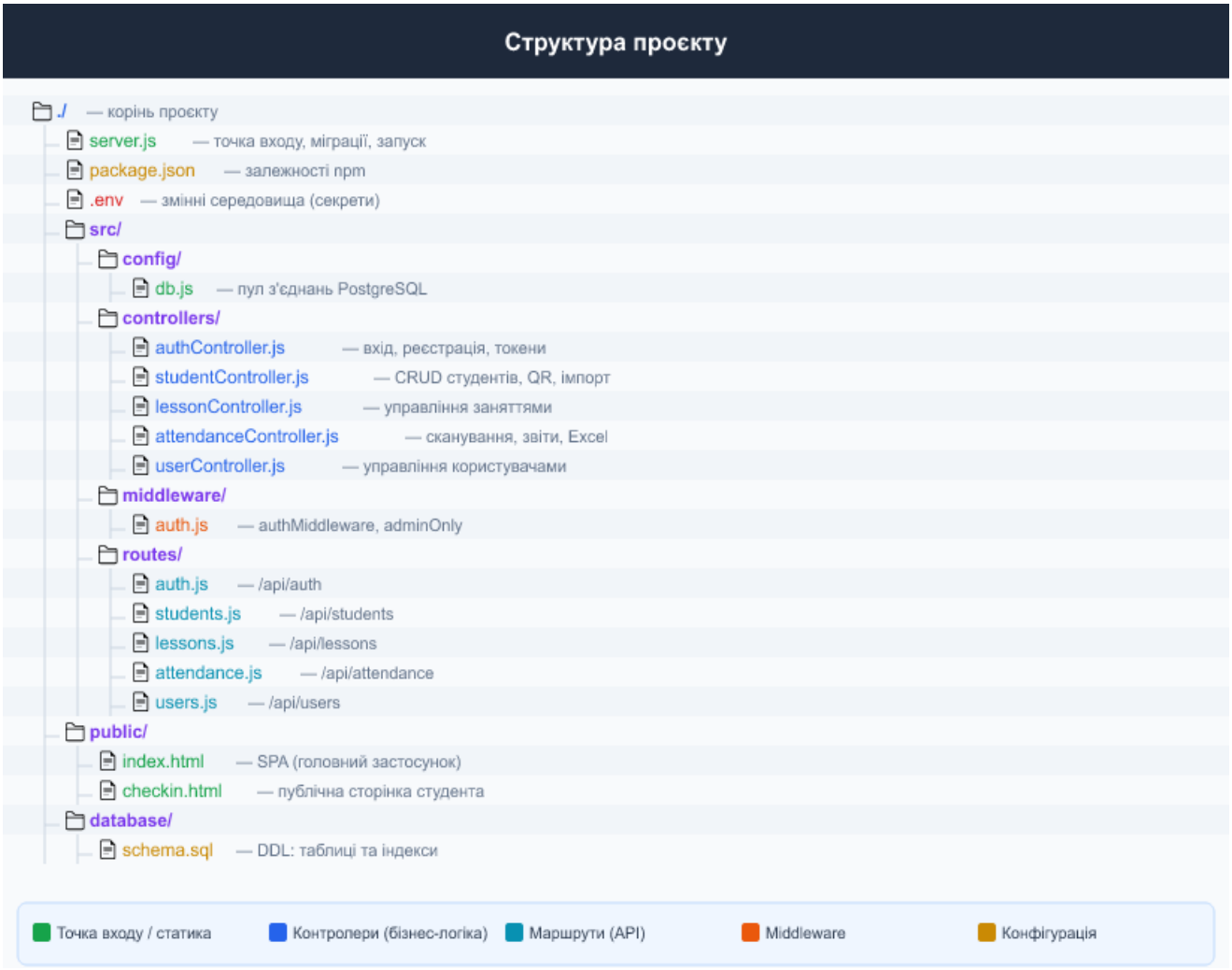


Рис. 4.1 Структура файлів проєкту

Налаштування підключення до PostgreSQL винесено у окремий модуль src/config/db.js. Використання пулу з'єднань замість одиничного підключення дозволяє ефективно обслуговувати кілька одночасних HTTP-запитів без очікування звільнення єдиного з'єднання. Параметри пулу задаються через змінні середовища (.env), що унеможливлює потрапляння облікових даних бази у репозиторій вихідного коду.

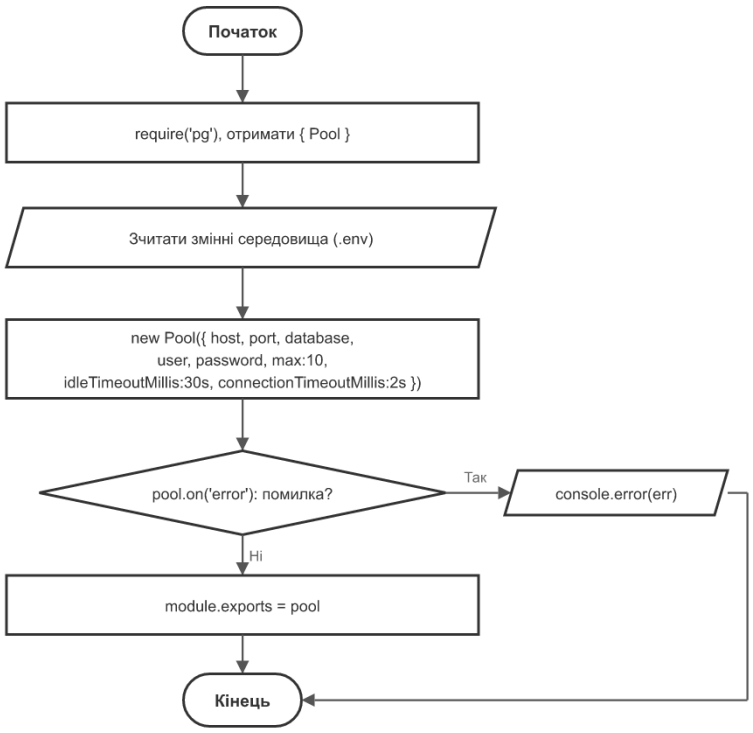


Рис. 4.2 Блок схема підключення до бази даних

4.1.2. Реалізація REST API

REST API організований за ресурсним принципом. Всі маршрути мають префікс /api/.

Детальний перелік REST API маршрутів системи наведено в таблиці В.1 додатка В.

Маршрутизатор /api/students (router.use(authMiddleware))

Метод	Шлях	Middleware	Контролер
GET	/	—	getAllStudents
GET	/:id	—	getStudentById
GET	/:id/qr	—	getStudentQR
GET	/:id/stats	—	getStudentStats
POST	/	adminOnly	createStudent
POST	/import	adminOnly	importStudents
POST	/import-excel	adminOnly + multer	importStudentsExcel
PUT	/:id	adminOnly	updateStudent
DELETE	/:id	adminOnly	deleteStudent

Рис. 4.3 Маршрутизатор students.js

4.1.3. Система автентифікації та refresh-токени

Реалізація системи автентифікації зосереджена в authController.js. Функція login() отримує email і password із тіла запиту. Вона знаходить користувача у таблиці users і перевіряє пароль за допомогою бібліотеки bcrypt. Якщо все вірно, генерується пара токенів: access-токен і refresh-токен. Refresh-токен зберігається у таблиці refresh_tokens разом із user_id та expires_at. Потім він встановлюється в httpOnly cookie через виклик setRefreshCookie(). У відповіді клієнту повертається access-токен і дані користувача.

Функція refreshToken() реалізує ротацію токенів. Вона зчитує refresh-токен із httpOnly cookie, яка недоступна для JavaScript. Якщо токена немає, функція повертає статус 401. Знайшовши відповідний запис у таблиці refresh_tokens, вона перевіряє термін дії поля expires_at і видаляє старий токен із бази даних. Після цього генерується нова пара токенів: новий refresh-токен зберігається у базі та встановлюється в cookie, а новий access-токен повертається клієнту.

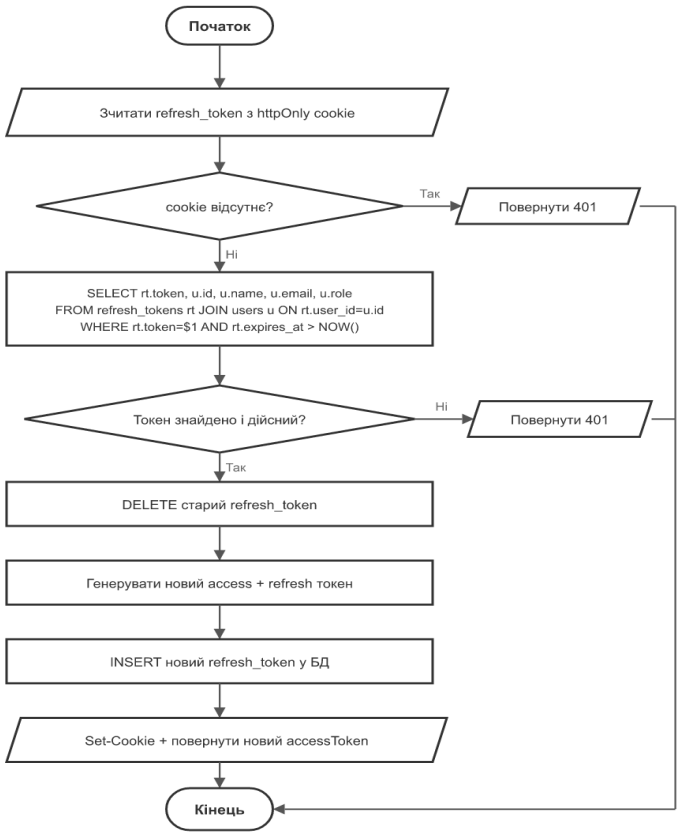


Рис. 4.4 Блок схема функції refreshToken()

Функція setRefreshCookie() встановлює cookie з параметрами безпеки: httpOnly: true (недоступний для JavaScript), secure: true у production (тільки HTTPS), sameSite: 'strict' (захист від CSRF), path: '/api/auth' (доступний лише для ендпоінтів автентифікації).

Перевірка JWT-токена у кожному захищеному запиті реалізована у middleware функції authMiddleware (src/middleware/auth.js). Вона витягує токен із заголовка Authorization, верифікує його підпис та термін дії. Функція adminOnly додатково перевіряє роль користувача, що дозволяє обмежити доступ до окремих ендпоінтів лише адміністраторам.

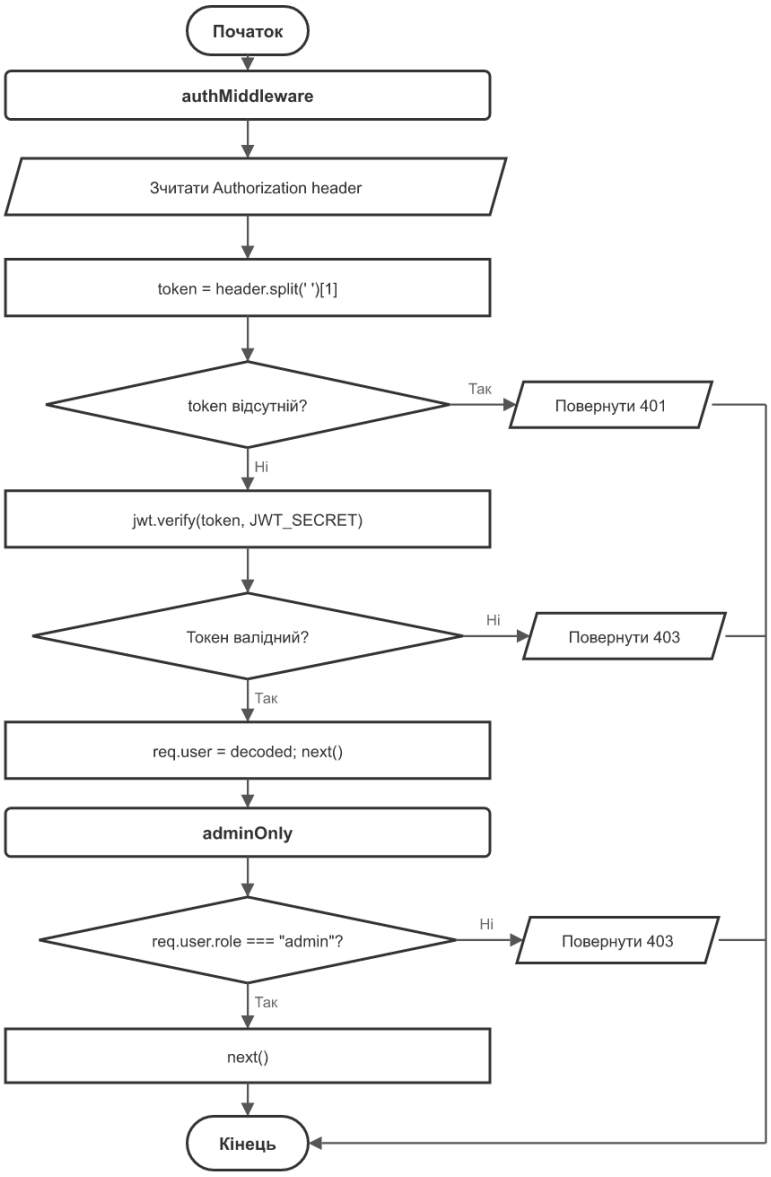


Рис. 4.5 Блок схема middleware автентифікації (auth.js)

4.1.4. Генерація та зчитування QR-кодів

Генерація QR-коду відбувається при створенні студента у функції createStudent() контролера studentController.js.

Дані, що кодуються у QR-коді:

{"type": "student", "code": "STUDENT_CODE"}

де STUDENT_CODE це унікальний ідентифікатор студента. Обраний формат JSON дозволяє у майбутньому розширити протокол без зміни інфраструктури сканування як наприклад, підтримати QR-коди занять для самостійного відмічання студентами.

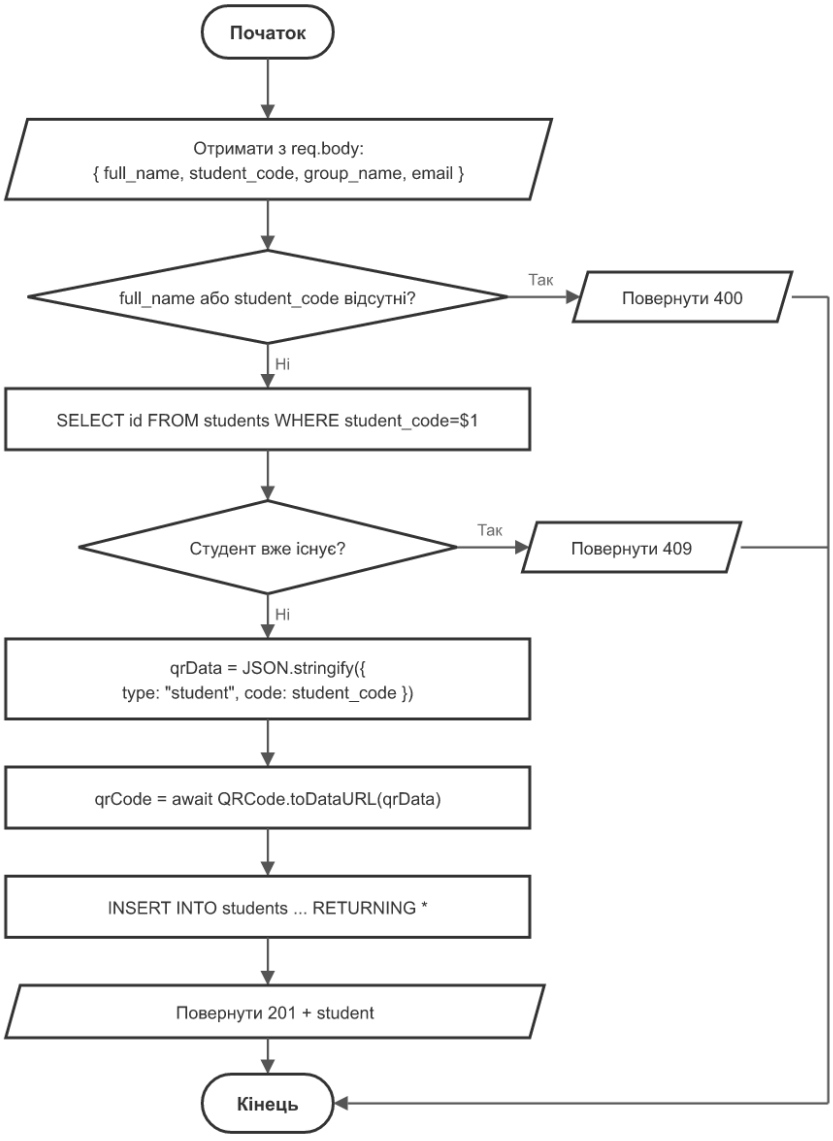


Рис. 4.6 Блок схема генерації QR-коду

QR-код зберігається у полі qr_code таблиці students у вигляді рядка ... Це дозволяє безпосередньо використовувати значення як src HTML-елементу <img> без додаткових перетворень. При отриманні запиту GET /api/students/:id/qr сервер повертає цей рядок, що клієнт відображає як зображення.

На стороні клієнта зчитування QR-коду реалізовано за допомогою бібліотеки «jsQR» – спеціалізованого модуля, що виконує аналіз растрових зображень з відеопотоку камери. Код функції scanFrame() виконується кожен кадр анімаційного циклу:

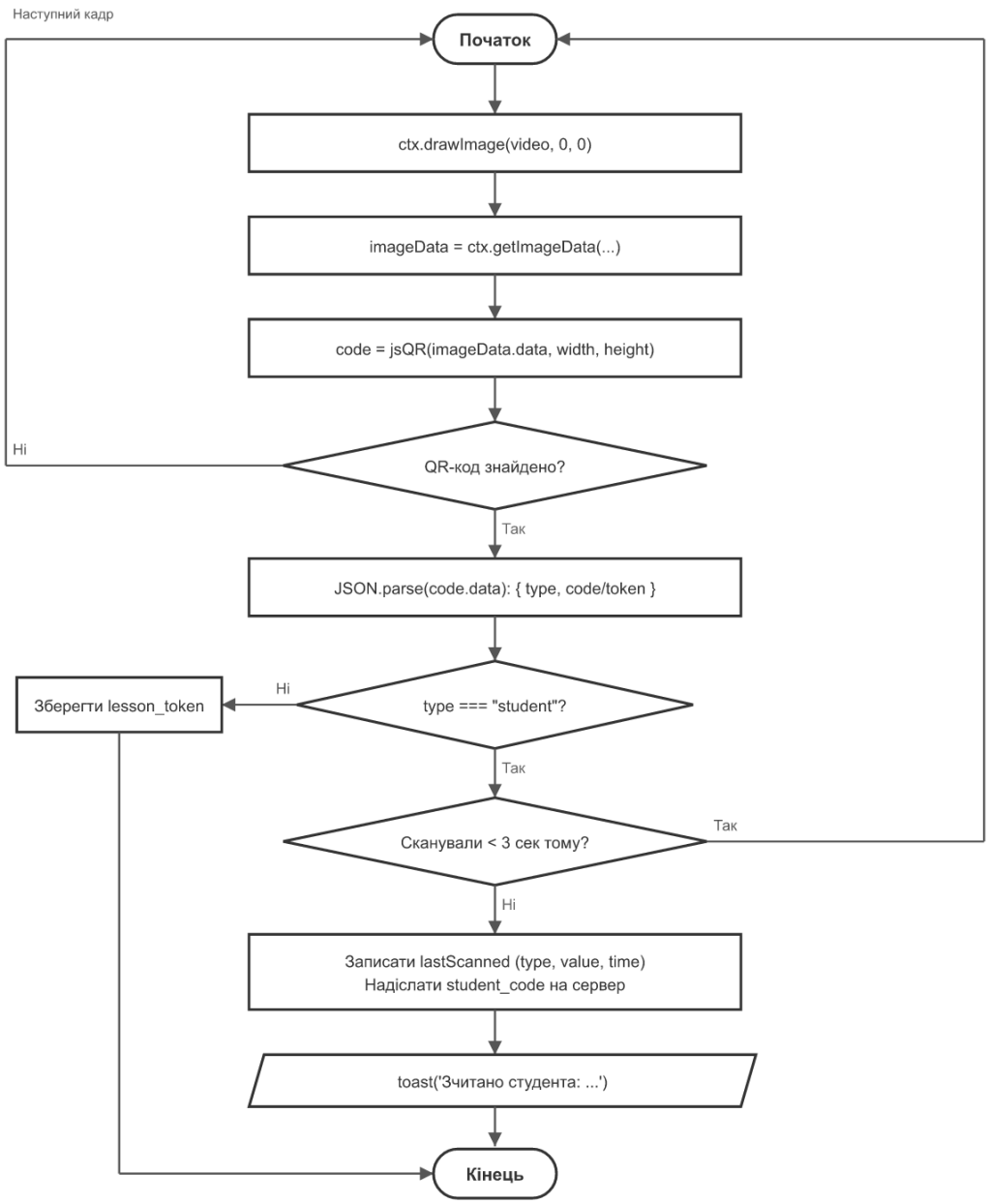


Рис. 4.7 Блок схема функції сканування («jsQR»)

4.1.5. Серверна реалізація реєстрації відвідуваності

Функція scanQR() контролера attendanceController.js обробляє сканований QR-код на сервері. Вона отримує lesson_token та student_code з тіла запиту. Функція знаходить відповідне заняття у таблиці lessons за умовою qr_token=$1 AND qr_expires_at > NOW(). Якщо заняття не знайдено або термін дії QR-коду вичерпано, повертається статус 400. Далі система виконує пошук студента за student_code; якщо студента немає, повертається 404. Після цього перевіряється відповідність групи студента та групи заняття. Якщо групи не співпадають, повертається код 403. Якщо всі перевірки пройдені успішно, розраховується статус відвідування: якщо з початку заняття минуло більше 15 хвилин, присвоюється статус late, інакше – present. Запис відвідування виконується через INSERT INTO attendance ON CONFLICT DO UPDATE, це запобігає дублікатам даних.

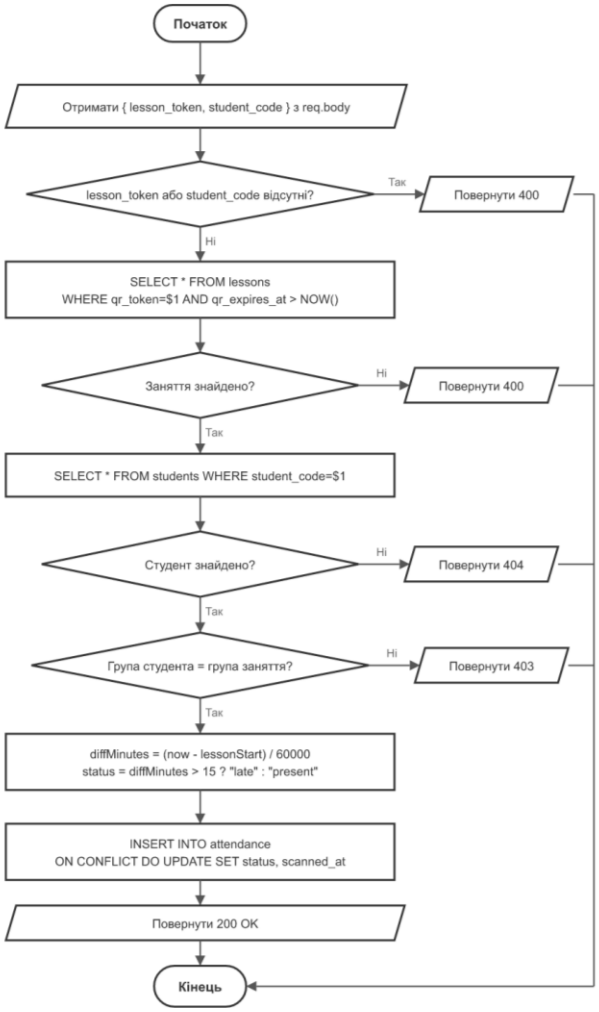


Рис. 4.8 Блок схема коду функції scanQR()

4.2. Реалізація клієнтської частини

4.2.1. SPA-архітектура фронтенду

Клієнтський застосунок реалізований як Single Page Application. Вся структура сторінок визначена у єдиному HTML-файлі, де кожен розділ обгорнутий у елемент <section> із унікальним id. Навігація між розділами відбувається через функцію showSection(name), що приховує всі секції та відображає обрану.

Access-токен зберігається у localStorage. При завантаженні сторінки перевіряється наявність токена: якщо він відсутній – відображається форма входу; якщо присутній – виконується запит GET /api/auth/me для підтвердження валідності та отримання даних поточного користувача.

Функція apiReq() – Централізована функція для всіх API-запитів, що автоматично:

- підставляє Bearer-токен у заголовок Authorization;
- включає credentials: 'include' для передачі cookie;
- при отриманні 401 виконує silent refresh та повторює запит;
- повертає пари {res, data} для зручної перевірки у коді.

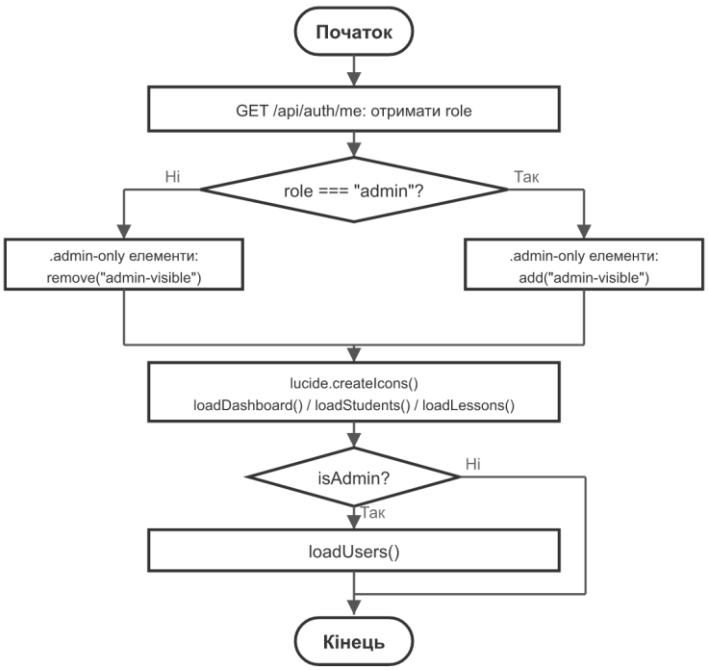


Рис. 4.9 Блок схема контролю доступу на клієнті

Контроль доступу на клієнті. Елементи з класом admin-only показуються лише адміністратору.

Авторизація на рівні клієнта є лише елементом UX – реальна перевірка прав виконується на сервері через middleware.

4.2.2. Сканування QR-кодів через камеру

Активація камери відбувається через Web API.

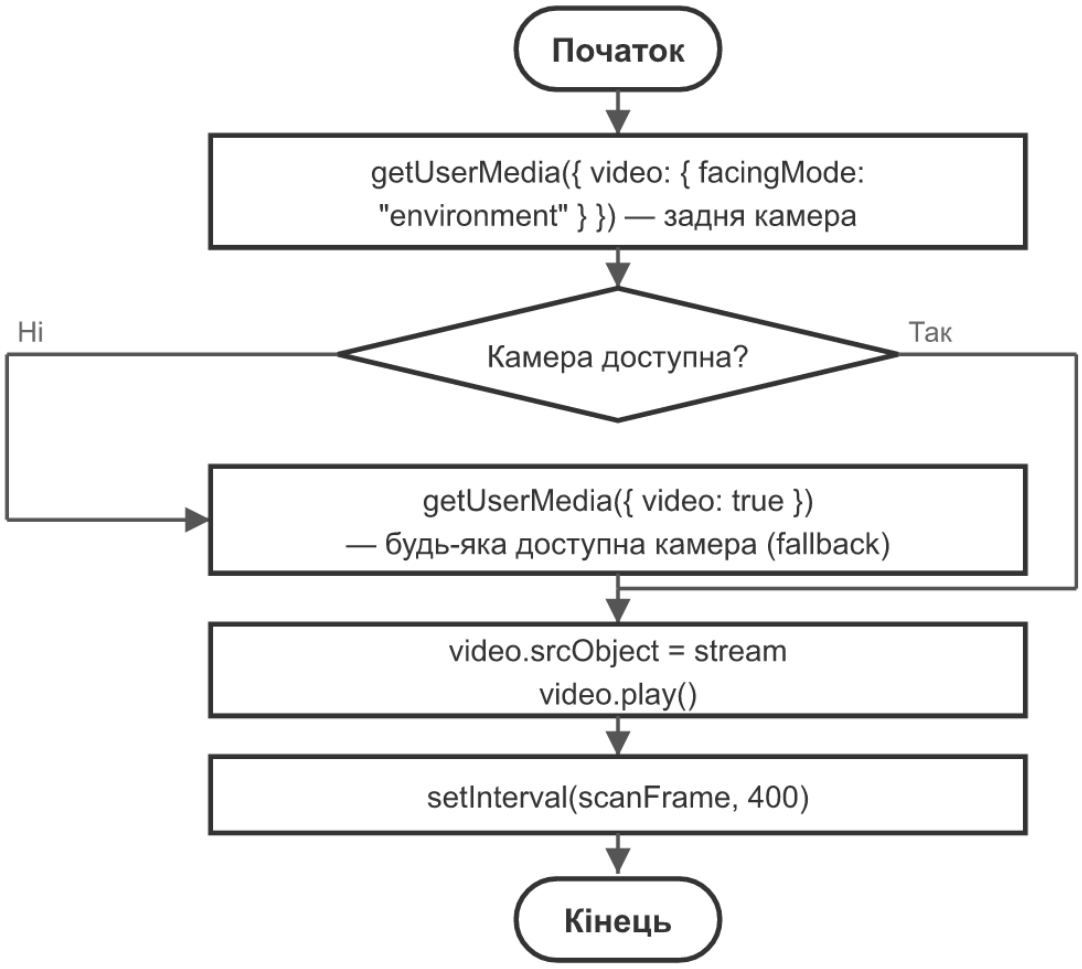


Рис. 4.10 Блок схема активації камери

Параметр facingMode: 'environment' дозволяє на мобільних пристроях автоматично вибрати задню камеру, яка зазвичай має кращу якість та підходить для сканування QR-кодів. Якщо задня камера недоступна (наприклад, на ноутбуці), браузер переключається на доступну.

Після початку відтворення відео запускається цикл покадрової обробки через стандартний Web API-механізм планування анімацій. Кожен кадр відеопотоку

копіюється на canvas-елемент, після чого «jsQR» аналізує піксельні дані та намагається знайти QR-код. Продуктивність обробки залежить від роздільної здатності відео та продуктивності пристрою, проте на сучасних пристроях сканування відбувається миттєво.

При успішному скануванні функція registerAttendance() надсилає запит до сервера та відображає toast-повідомлення. При виході зі сторінки сканування відеопотік зупиняється для звільнення ресурсів камери.

4.2.3. Звіти та аналітика

Розділ звітності реалізовано у секції «Звіт» та на сторінці статистики студента.

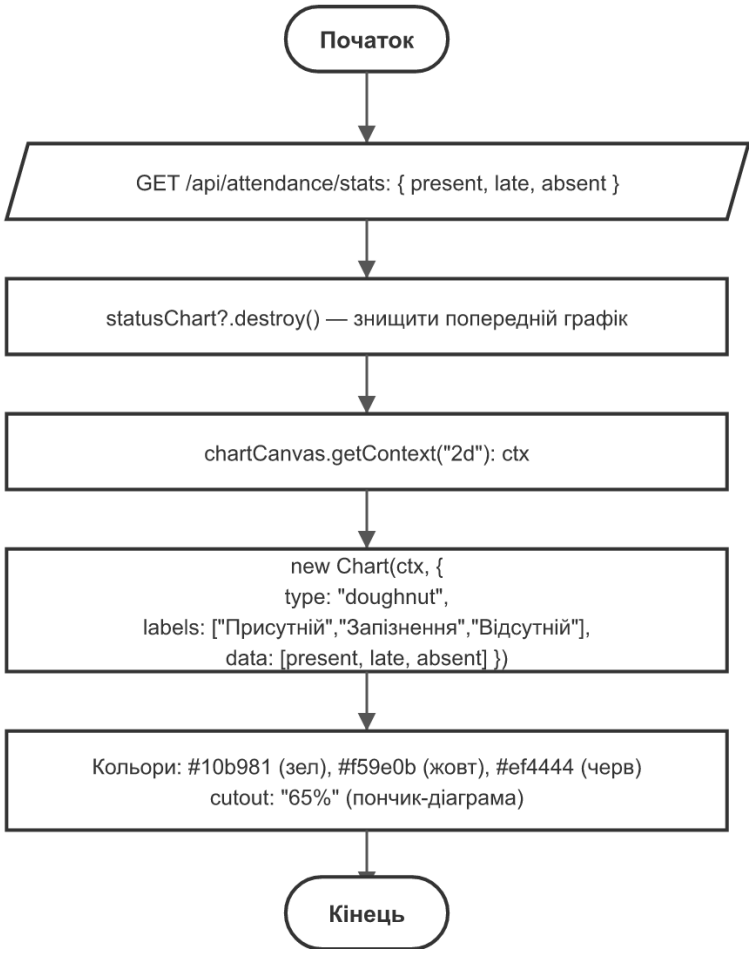


Рис. 4.11 Блок схема побудови діаграми («Chart.js »)

Звіт відвідуваності заняття – запит до GET /api/attendance/report із параметрами фільтрації повертає масив рядків, що відображається у таблиці.

Кожен рядок містить ім'я студента, код, статус та час відмічання. Статус візуально виділяється кольоровими значками-мітками : зелений – присутній, жовтий – запізнився, червоний – відсутній.

Статистика студента – запит до GET /api/students/:id/stats повертає агреговані дані: кількість відвідань, запізнень та пропусків. На клієнті розраховується відсоток відвідуваності та відображається у вигляді прогрес-бару.

Аналітика на головній панелі реалізована за допомогою бібліотеки «Chart.js» – засобу візуалізації статистичних даних у браузері. Кругова діаграма показує розподіл відміток за статусами. Дані формуються окремими запитами до таблиці attendance та передаються клієнту у форматі JSON.

4.2.4. SQL-запит для звіту відвідуваності

Для коректного відображення відсутніх студентів (тих, для кого немає запису в таблиці attendance) використовується техніка CROSS JOIN з подальшим LEFT JOIN. CROSS JOIN формує декартовий добуток занять та студентів (усі можливі пари), а LEFT JOIN додає відповідний запис про відвідування, якщо він існує. При відсутності запису COALESCE повертає значення 'absent'.

Експорт у Excel – це виконує функція exportExcel у attendanceController.js, яка використовує бібліотеку «ExcelJS» – модуль для програмного формування .xlsx файлів із підтримкою стилізації комірок, безпосередньо у пам'яті сервера:

1. Створюється новий Workbook та Worksheet.
2. Визначаються стовпці.
3. Для кожного рядка звіту додається запис із форматуванням: Зелений фон – статус present, Жовтий фон – статус late. Червоний фон – відсутній.
4. Сформований файл записується безпосередньо у HTTP-відповідь засобами вбудованого методу стрімінгового запису бібліотеки «ExcelJS».
5. Заголовки відповіді встановлюють ім'я файлу та тип контенту.

На клієнті при натисканні «Експорт Excel» виконується fetch-запит, відповідь перетворюється на Blob та завантажується через динамічно створене посилання <a>.

Блок схема формування Excel-файлу з кольоровим маркуванням рядків наведено нижче. Для кожного рядка звіту визначається колір фону залежно від статусу відвідування, після чого файл без збереження на диску записується безпосередньо у HTTP-відповідь.

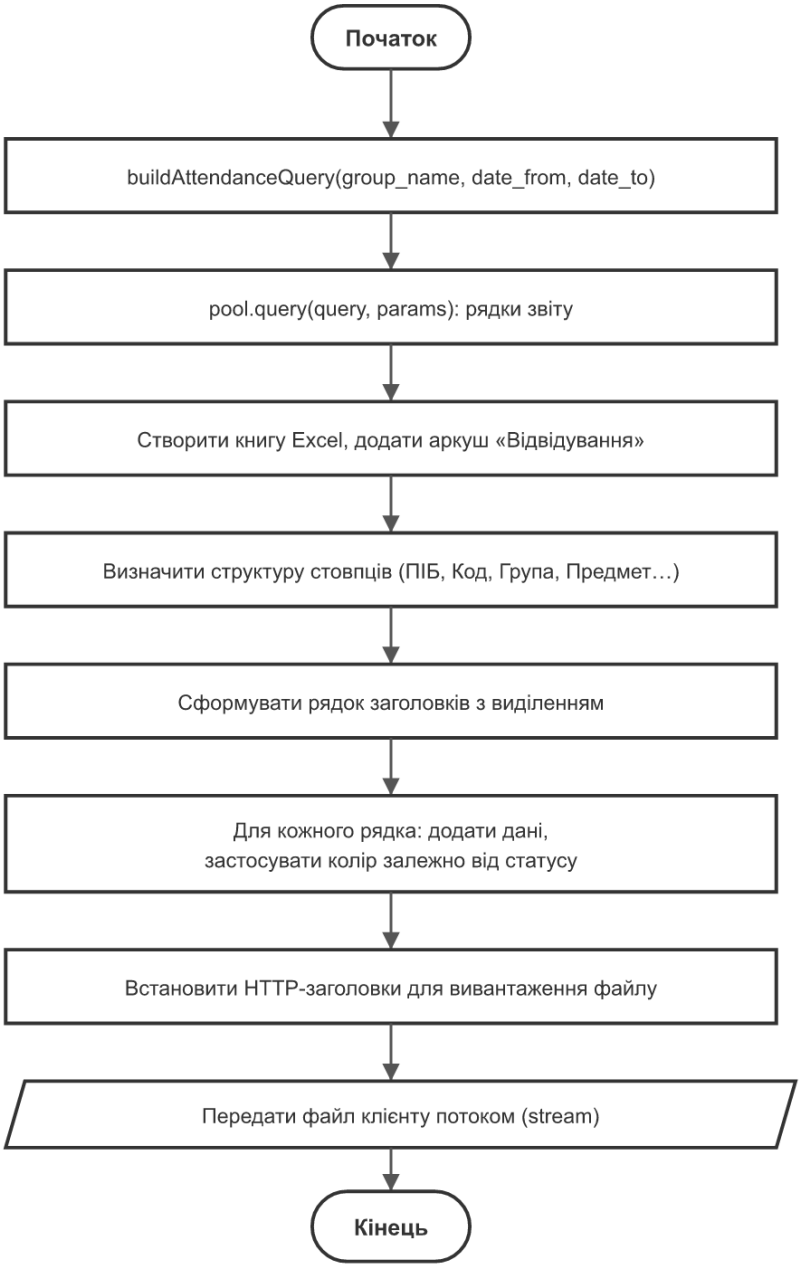


Рис. 4.12 Блок схема функції exportExcel() з кольоровим форматуванням

4.3. Тестування застосунку

4.3.1. Тестові сценарії

Для перевірки коректності роботи застосунку розроблено набір тестових сценаріїв, що охоплюють усі ключові функціональні вимоги. Тестування проводилось методом чорної скриньки через браузерний інтерфейс та через прямі запити до REST API.

Таблиця 4.3

Тестові сценарії мануального тестування

Назва	Очікуваний результат	Статус
Вхід з коректними даними	JWT токен, перехід на SPA	Пройдено
Вхід з невірним паролем	Повідомлення про помилку	Пройдено
Реєстрація як викладач	Обліковий запис створено	Пройдено
Реєстрація без авторизації	401 Unauthorized	Пройдено
Додавання студента (admin)	Студент у списку, QR є	Пройдено
Додавання студента (teacher)	403 Forbidden	Пройдено
Сканування QR: 1-й раз	Відмітка збережена, toast	Пройдено
Сканування QR: повторно <3сек	Дублікат проігноровано	Пройдено
Сканування QR: >3сек	Нова відмітка не додається	Пройдено
Перегляд звіту	Таблиця з правильними даними	Пройдено
Експорт Excel	Файл .xlsx із кольорами	Пройдено
Імпорт із CSV	Студенти додані, QR згенер	Пройдено
Імпорт із Excel (.xlsx)	Студенти додані коректно	Пройдено
Видалення заняття (власник)	Заняття видалено	Пройдено

Продовження таблиці 4.3

Видалення заняття (інший)	403 Forbidden	Пройдено
Silent refresh токена	Прозоне оновлення токена	Пройдено
Вихід із системи	Токен відкликано, редирект	Пройдено
Фільтр студентів за групою	Відфільтрований список	Пройдено
Пошук студента	Результати в реальному часі	Пройдено
Статистика студента	Коректна агрегація	Пройдено

Система не дозволяє подвійну реєстрацію – UNIQUE constraint у БД та перевірка на рівні API.

4.3.2. Результати тестування

Усі 20 тестових сценаріїв пройдено успішно. Нижче наведено зведені результати за модулями.

Таблиця 4.4

Зведені результати тестування

Модуль	Функція	Статус
Автентифікація	JWT, refresh, logout	Пройдено
Авторизація	admin/teacher ролі	Пройдено
Управління студентами	CRUD + QR	Пройдено
Сканування QR-кодів	Реєстрація + дедуп	Пройдено
Управління заняттями	CRUD, ізоляція	Пройдено
Звіти та аналітика	Агрегація, коректність	Пройдено
Експорт Excel	Формат, кольори	Пройдено
Імпорт CSV/Excel	Пакетне додавання	Пройдено
Безпека	SQL-ін'єкції, XSS	Пройдено
Ізоляція даних	Захист чужих даних	Пройдено

Під час тестування безпеки перевірено:

- параметризовані SQL-запити: спроби ввести SQL-ін'єкцію у поля форм не призводять до виконання довільного SQL;
- доступ без токена до захищених ендпоінтів повертає 401;

- доступ з токеном викладача до адмін-ендпоінтів повертає 403;
- Refresh-токен після logout стає недійсним – повторне використання повертає 401.

Перевірка продуктивності:
- сканування QR та реєстрація відвідування: ~80-120 мс;
- завантаження списку 100 студентів: ~90 мс;
- генерація Excel-файлу для 50 записів: ~200 мс;
- ініціалізація сторінки при першому завантаженні: ~600 мс.

В цілому, процес тестування показав, що розроблена система є надійним інструментом для обліку відвідуваності в закладі вищої освіти. Усі перевірені сценарії відповідали очікувальній поведінці системи, і не було виявлено критичних або блокуючих недоліків. Інтерфейс залишається зрозумілим і зручним для користувачів з різним рівнем технічної підготовки, як для адміністраторів, що керують даними, так і для викладачів, які проводять реєстрацію відвідувань через сканування. Час відгуку системи на всіх ключових операціях є прийнятним для щоденного використання і не створює затримок у навчальному процесі. Перевірка сценаріїв доступу та захисту даних підтвердила, що система правильно обмежує дії користувачів відповідно до їхніх ролей і не допускає несанкціонований доступ до чужих даних. Отримані результати свідчать про готовність системи до експлуатації в умовах навчального закладу.

Система відповідає всім функціональним та нефункціональним вимогам, сформульованим у Розділі 1.

ВИСНОВКИ

У дипломній роботі вирішено наукове та практичне завдання розроблення веб орієнтованої системи автоматизованого обліку відвідувань студентів закладів вищої освіти на основі технології QR-кодування. За результатами виконаного дослідження зроблено такі висновки.

Порівняльний аналіз систем обліку відвідуваності на основі RFID, біометрії, мобільних застосунків та веб-платформ довів відсутність на ринку рішення, що одночасно задовольняє критеріям вартісної доступності, відповідності організаційній структурі вітчизняних закладів вищої освіти та незалежності від стороннього програмного забезпечення. Виявлений дефіцит обґрунтовує наукову та практичну значущість розробленого рішення.

Встановлено, що застосування платформи Node.js 24 з асинхронною подієво-орієнтованою моделлю виконання забезпечує стабільну обробку паралельних запитів без деградації продуктивності; використання СУБД PostgreSQL 16 з підтримкою ACID-транзакцій та декларативних обмежень гарантує цілісність даних про відвідуваність; дворівнева схема автентифікації на основі стандарту JSON Web Token з механізмом ротації токенів підтверджена як така, що відповідає рекомендаціям OWASP щодо захисту веб-застосунків.

Доведено коректність прийнятих архітектурних рішень: реляційна модель бази даних із п'яти взаємопов'язаних відношень забезпечила цілісність посилань та відсутність аномалій при паралельному записі; ресурсна організація REST API із 28 ендпоінтами підтвердила достатність для покриття всіх визначених варіантів використання; механізм «silent refresh» забезпечив прозоре поновлення сесії без видимих переривань для користувача.

Підтверджено функціональну повноту реалізованого програмного комплексу: рольова модель розмежування доступу коректно обмежує привілеї між ролями адміністратора та викладача; підсистема розпізнавання графічних QR-ідентифікаторів на основі модуля «jsQR» забезпечує перетворення растрового

зображення з відеопотоку камери у текстовий рядок у реальному часі; підсистема формування звітності коректно генерує документи формату xlsx із диференційованим кольоровим маркуванням; алгоритм пакетного імпорту забезпечив коректну нормалізацію заголовків стовпців файлів CSV та Excel.

Результати комплексного функціонального тестування за 20 розробленими сценаріями підтвердили повну відповідність реалізованого рішення висунутим функціональним вимогам. Перевірка коректності алгоритму розпізнавання QR-ідентифікаторів засвідчила стабільність роботи в різних умовах освітлення та відстані від камери. Тестування механізмів безпеки підтвердило стійкість системи до атак типу SQL-ін'єкція та несанкціонованого доступу до захищених ресурсів. Вимірювання часу відгуку при різному навантаженні показали значення в межах 80–200 мс для типових операцій, що задовольняє нефункціональним вимогам щодо продуктивності.

Практичне значення отриманих результатів полягає у тому, що розроблена система є готовою до розгортання у навчальних закладах, не залежить від хмарних сервісів третіх сторін, не потребує ліцензійних відрахувань та може функціонувати на будь-якому пристрої з браузером без встановлення додаткового програмного забезпечення.

ПЕРЕЛІК ПОСИЛАНЬ

1. Опанасенко А.Ю., Гавор А.С. Застосування QR-Кодування для автоматизації обліку відвідуваності у закладах ввищої освіти. // VII Всеукраїнську науково-технічну конференцію «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 23.04.2026, ДУІКТ, Київ С. 55-57.

2. Опанасенко А.Ю., Гавор А.С. Аналіз вразливостей протоколу Oauth 2.0 у вебзастосунках та методи захисту від типових атак. // Всеукраїнська наукова-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. 1-3.05.2026, ДУІКТ, Київ – подано до друку.

3. Node.js Documentation. URL: https://nodejs.org/en/docs (дата звернення: 20.03.2026).

4. Express.js. Fast, unopinionated, minimalist web framework for Node.js. URL: https://expressjs.com/ (дата звернення: 20.03.2026).

5. PostgreSQL 16 Documentation. URL: https://www.postgresql.org/docs/16/ (дата звернення: 20.03.2026).

6. node-postgres. PostgreSQL client for Node.js. URL: https://node-postgres.com/ (дата звернення: 20.03.2026).

7. RFC 7519: JSON Web Token (JWT). IETF. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 20.03.2026).

8. OWASP Authentication Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html (дата звернення: 20.03.2026).

9. Provos N., Mazières D. A Future-Adaptable Password Scheme // USENIX Annual Technical Conference. 1999.

10. Denso Wave. QR Code Standardization. URL: https://www.qrcode.com/en/about/standards.html (дата звернення: 22.03.2026).

11. qrcode npm package. URL: https://www.npmjs.com/package/qrcode (дата звернення: 22.03.2026).

12. jsQR. A pure javascript QR code reading library. URL: https://github.com/cozmo/jsQR (дата звернення: 22.03.2026).

13. Chart.js. Simple yet flexible JavaScript charting library. URL: https://www.chartjs.org/docs/4.4.0/ (дата звернення: 22.03.2026).

14. Lucide Icons. URL: https://lucide.dev/ (дата звернення: 22.03.2026).

15. ExcelJS. Read, manipulate and write spreadsheet data and styles. URL: https://www.npmjs.com/package/exceljs (дата звернення: 23.03.2026).

16. multer. Node.js middleware for handling multipart/form-data. URL: https://www.npmjs.com/package/multer (дата звернення: 23.03.2026).

17. xlsx. SheetJS Community Edition. URL: https://www.npmjs.com/package/xlsx (дата звернення: 23.03.2026).

18. MDN Web Docs. MediaDevices.getUserMedia(). URL: https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia (дата звернення: 24.03.2026).

19. OWASP Top 10. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 28.02.2026).

20. bcrypt npm package. URL: https://www.npmjs.com/package/bcrypt (дата звернення: 24.03.2026).

21. cookie-parser. Parse Cookie header and populate req.cookies. URL: https://www.npmjs.com/package/cookie-parser (дата звернення: 24.03.2026).

22. ISO/IEC 18004:2024. Information technology – Automatic identification and data capture techniques - QR Code bar code symbology specification. ISO, 2024.

23. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.

24. dotenv. Loads environment variables from .env file into process.env.URL: https://www.npmjs.com/package/dotenv (дата звернення: 25.03.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Технологій цифрового розвитку

Система автоматизації обліку відвідувань на основі технології QR-кодування з використанням Node.js та PostgreSQL

Виконав студент 4 курсу
Групи ТЦР-42
Опанасенко Андрій Юрійович
Керівник роботи
Викл. кафедри ТЦР Гавор Артур Станіславович

Київ-2026

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – підвищення ефективності обліку відвідувань студентів закладів вищої освіти за допомогою автоматизації реєстрації присутності через QR-кодування, що забезпечує швидку та точну фіксацію відвідуваності, зручне управління даними для викладачів та адміністраторів, а також формування аналітичних звітів.

Об'єкт дослідження – процес автоматизованого обліку відвідувань студентів у закладах вищої освіти.

Предмет дослідження – методи та засоби проєктування і розробки веб-застосунку для обліку відвідувань на основі технології QR-кодування з використанням платформи Node.js та PostgreSQL.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати існуючі рішення для обліку відвідуваності студентів, визначити їх переваги та недоліки
2. Сформулювати функціональні та нефункціональні вимоги до розроблюваного застосунку
3. Обґрунтувати вибір технологічного стеку для реалізації серверної та клієнтської частин .
4. Спроєктувати архітектуру системи, діаграму для бази даних та діаграми взаємодії компонентів
5. Реалізувати серверну частину із REST API та систему автентифікації на основі JWT з refresh-токенами
6. Реалізувати функціонал генерації та зчитування QR-кодів через камеру пристрою
7. Розробити клієнтську частину у вигляді односторінкового застосунку
8. Провести тестування застосунку та оцінити відповідність реалізованого рішення висунутим вимогам.

АНАЛІЗ АНАЛОГІВ

Критерій	Google Forms	Classter	Attendo	Розроблений застосунок
Безкоштовність	+	-	-	+
Qr-Ідентифікація	-	-	-	+
Веб-Інтерфейс	+	+	-	+
Ролева модель	-	+	-	+
Звіти та аналітика	-	+	-	+
Експорт Excel	-	+	-	+
Без зовн. обладнання	+	+	+	+
Без хмарної залежності	-	-	+	+
Відкритий вихідний код	-	-	-	+
Підтримка укр.мови	+	-	-	+

ВИМОГИ ДО СИСТЕМИ

Функціональні вимоги:

1. Дві ролі: адміністратор, який має повний доступ та викладач, який має доступ лише до своїх занять.
2. Автентифікація через захищений механізм авторизації з підтримкою ролей.
3. Управління студентами: додавання, редагування, видалення, пошук, фільтрація за групою, пагінація по 10 записів.
4. Автоматична генерація QR-коду для кожного студента після створення запису.
5. Імпорт списків студентів із файлів CSV та Excel.
6. Управління заняттями: створення з зазначенням предмета, дати та групи.
7. Реєстрація відвідуваності через сканування QR-кодів із визначенням статусу «присутній» або «запізнився».
8. Захист від дублювання, ігнорування повторного сканування протягом 3 секунд.
9. Публічна сторінка для самостійного відзначення студентом.
10. Перегляд статистики відвідуваності по кожному студенту та занятті.
11. Експорт звітів у форматі Excel із кольоровим форматуванням.

ВИМОГИ ДО СИСТЕМИ

Нефункціональні вимоги:

1. Паролі зберігаються у хешованому вигляді.
2. Автентифікація через JWT: access-токен – 1 год, refresh-токен – 7 днів у httpOnly cookie.
3. Захист від SQL-ін'єкцій через параметризовані запити.
4. Час завантаження сторінок – не більше 2 секунд.
5. Реєстрація відвідування після сканування – менше 1 секунди.
6. Підтримка щонайменше 50 одночасних користувачів.
7. Сумісність із Chrome, Firefox, Edge, Safari без додаткових розширень.
8. Масштабована архітектура для подальшого розширення системи.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Технології та інструменти, використані для розробки, тестування та забезпечення безпеки додатку

Visual Studio Code

Середовище розробки з підтримкою розширень, інтегрованим терміналом та зручними інструментами для роботи з кодом.

Node.js

JavaScript-платформа для виконання коду на стороні сервера та побудови масштабованих мережевих застосунків.

Express.js

Веб-фреймворк для побудови HTTP-запитів, маршрутів, middleware та REST API.

PostgreSQL

Об'єктно-реляційна система керування базами даних для зберігання та обробки інформації.

bcrypt

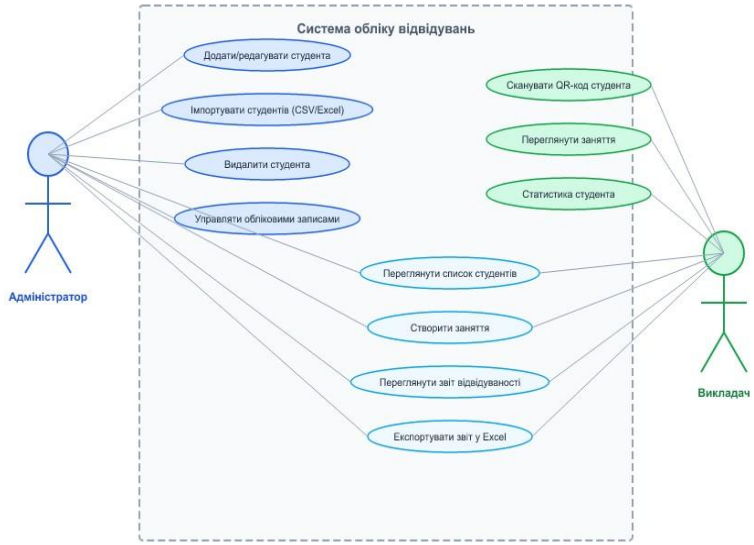
Бібліотека для хешування паролів, що забезпечує надійний захист даних користувачів за допомогою алгоритму bcrypt.

JWT (JSON Web Token)

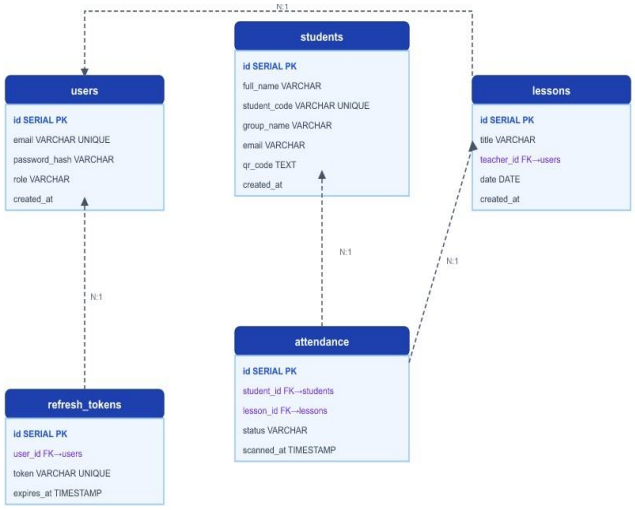
Стандарт для безпечної передачі інформації між сторонами у вигляді токенів, що використовується для автентифікації.

Для підтримки контролю версій, тестування та управління залежностями використовувались вбудовані інструменти Node.js та розширення Visual Studio Code.

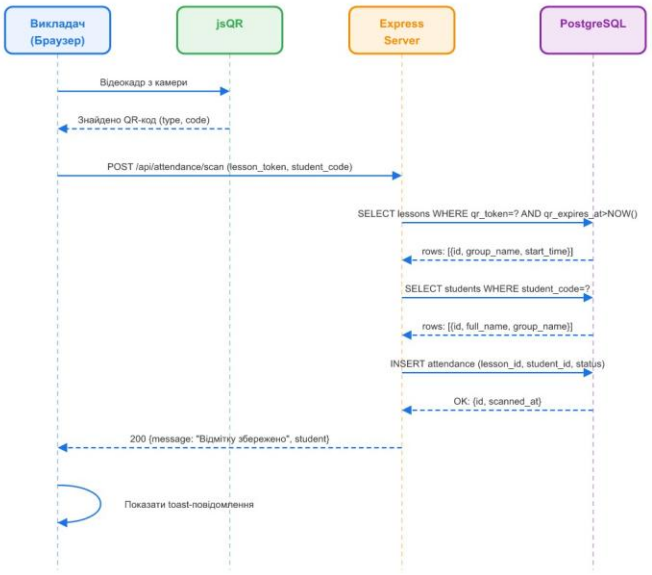
ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



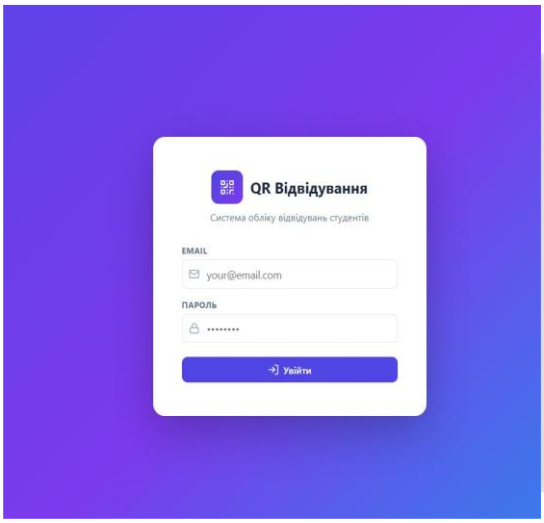
ER-модель даних



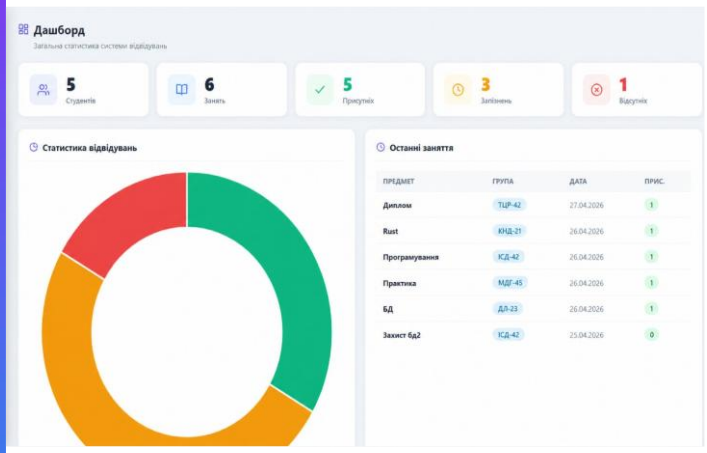
Діаграми послідовності реєстрація відвідуваності через QR-код



ЕКРАННІ ФОРМИ

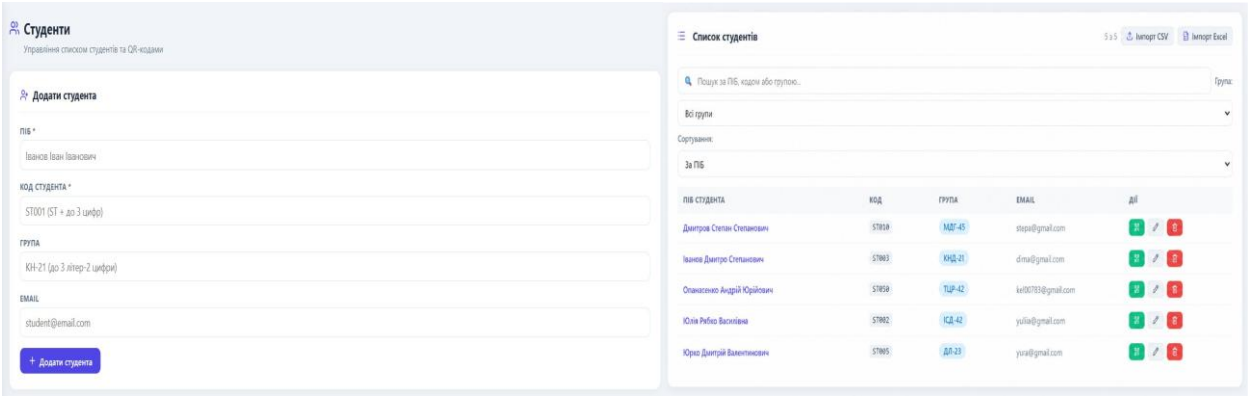


Сторінка входу до системи



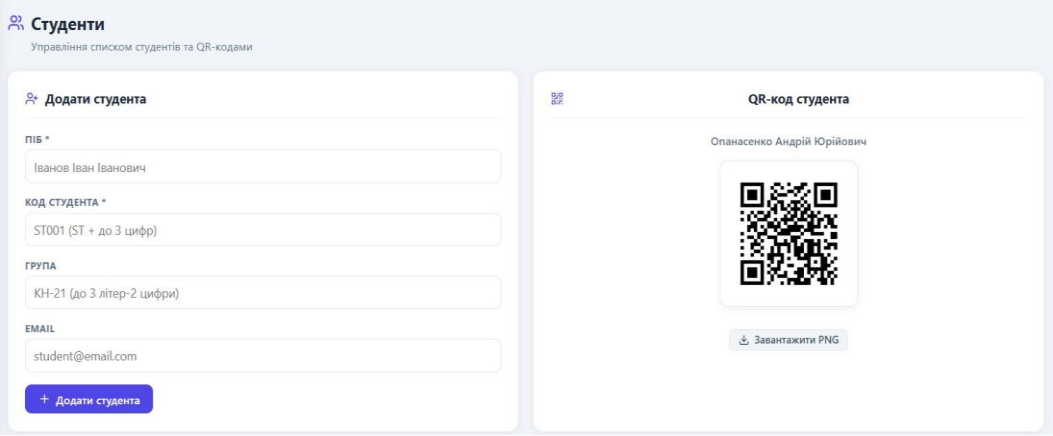
Головна панель із статистикою

ЕКРАННІ ФОРМИ



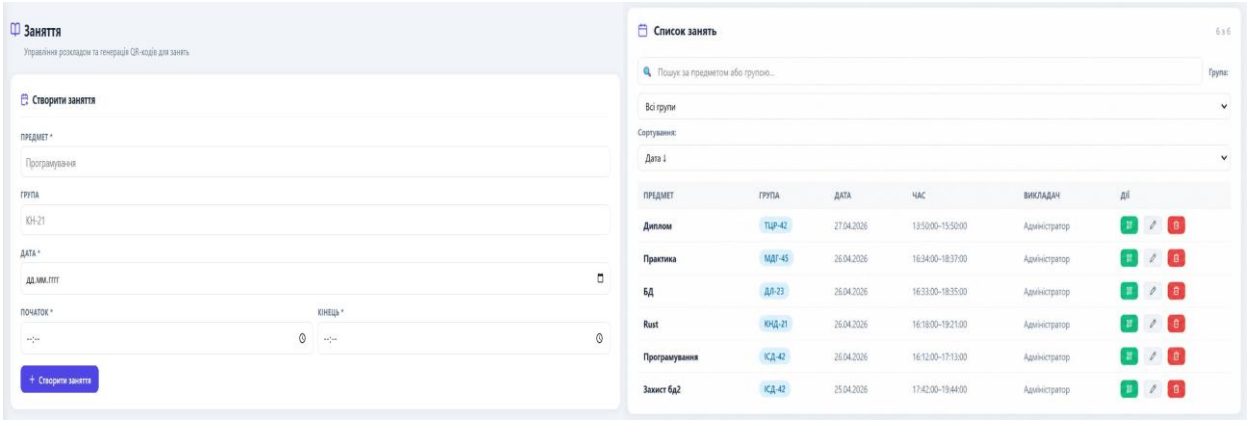
Розділ студенти

ЕКРАННІ ФОРМИ



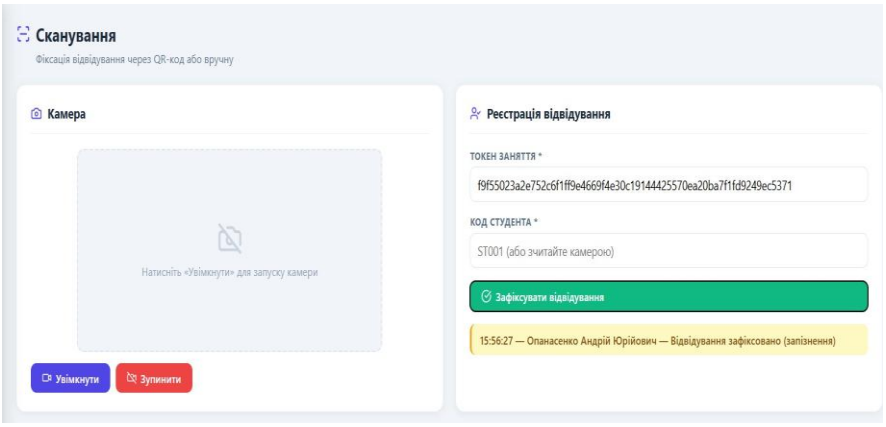
Модальне вікно з QR-кодом студента

ЕКРАННІ ФОРМИ

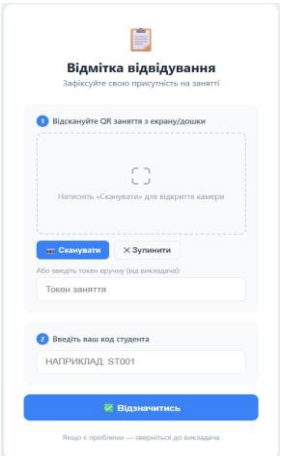


Розділ Заняття

ЕКРАННІ ФОРМИ

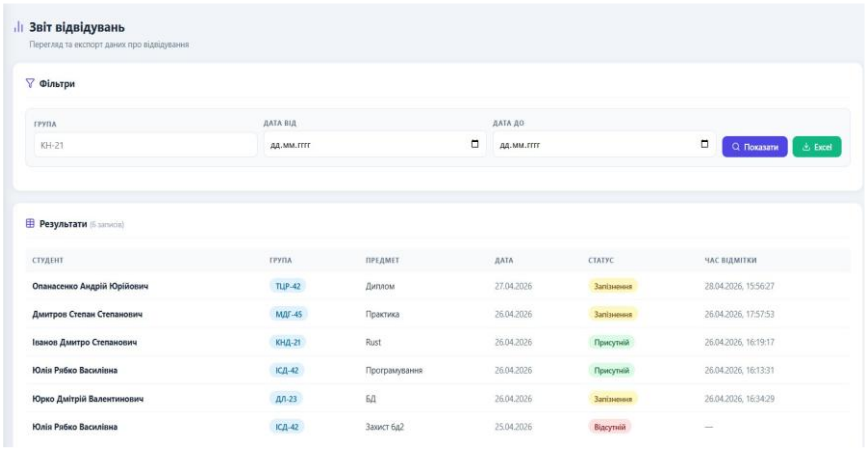


Модальне вікно сканування QR-кодів



Відмітка відвідування(для студента)

ЕКРАННІ ФОРМИ



Розділ звіт відвідувань

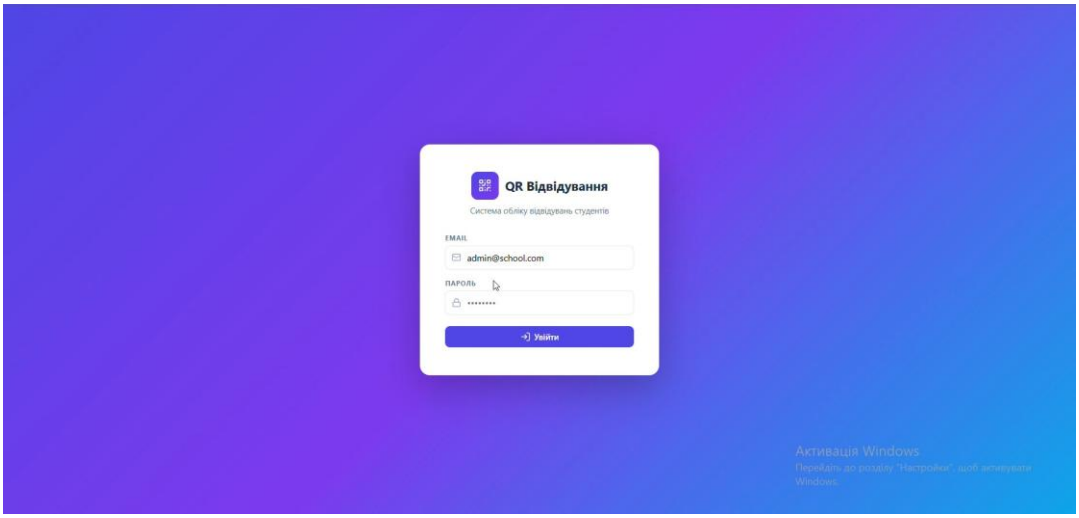
ЕКРАННІ ФОРМИ

A1 | ПІБ студента

	A	B	C	D	E	F	G	H
1	ПІБ студента	Код студента	Група	Предмет	Дата	Початок	Статус	Час відмітки
2	Опанасенко Андрій Юрійович	ST050	ТЦР-42	Диплом	27.04.2026	13:50:00	Запізнення	28.04.2026, 15:56:27
3	Дмитров Степан Степанович	ST010	МДГ-45	Практика	26.04.2026	16:34:00	Запізнення	26.04.2026, 17:57:53
4	Іванов Дмитро Степанович	ST003	КНД-21	Rust	26.04.2026	16:18:00	Присутній	26.04.2026, 16:19:17
5	Юлія Рябко Василівна	ST002	ІСД-42	Програмування	26.04.2026	16:12:00	Присутній	26.04.2026, 16:13:31
6	Юрко Дмітрій Валентинович	ST005	ДЛ-23	БД	26.04.2026	16:33:00	Запізнення	26.04.2026, 16:34:29
7	Юлія Рябко Василівна	ST002	ІСД-42	Захист бд2	25.04.2026	17:42:00	Відсутній	-
8								
9								
10								

Згенерований Excel-файл звіту з кольоровим форматуванням

ВІДЕОДЕМОНСТРАЦІЯ РОБОТИ



АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Опанасенко А.Ю., Гавор А.С. Застосування QR-Кодування для автоматизації обліку відвідуваності у закладах ввищої освіти. // VII Всеукраїнську науково-технічну конференцію «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 23.04.2026, ДУІКТ, Київ, Україна

2. Опанасенко А.Ю., Гавор А.С. Аналіз вразливостей протоколу Oauth 2.0 у вебзастосунках та методи захисту від типових атак. // Всеукраїнська наукова-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. 1-3.05.2026, ДУІКТ, Київ, Україна.

ВИСНОВКИ

1. Виконано огляд та порівняльний аналіз наявних систем обліку відвідуваності студентів. Встановлено, що більшість комерційних рішень є платними або не адаптованими до потреб вітчизняних закладів освіти, що підтвердило доцільність розробки власного застосунку.
2. Обґрунтовано вибір технологічного стеку та сформульовано вимоги до системи. Для серверної частини обрано Node.js із Express.js, для зберігання даних – PostgreSQL, для автентифікації – JWT із механізмом ротації refresh-токенів.
3. Спроєктовано архітектуру системи за клієнт-серверним принципом із REST API, розроблено ER-модель бази даних іта побудовано діаграми варіантів використання і послідовності для ключових сценаріїв.
4. Реалізовано повнофункціональну систему із модулем сканування QR-кодів через камеру браузера, системою ролевого доступу, аналітичною панеллю на базі Chart.js та підсистемою формування звітів з експортом у Excel.
5. Впроваджено функцію пакетного імпорту студентів із файлів CSV та Excel із нормалізацією заголовків, що суттєво спрощує первинне наповнення бази даних у реальних умовах експлуатації.
6. Проведено функціональне тестування застосунку за різними сценаріями, усі тести пройдено успішно, застосунок готовий до розгортання.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
require('dotenv').config();
const express = require('express');
const cors = require('cors');
const cookieParser = require('cookie-parser');
const path = require('path');
const pool = require('./src/config/db');

const authRoutes = require('./src/routes/auth');
const studentRoutes = require('./src/routes/students');
const lessonRoutes = require('./src/routes/lessons');
const attendanceRoutes = require('./src/routes/attendance');
const userRoutes = require('./src/routes/users');

const app = express();

const allowedOrigins = (process.env.ALLOWED_ORIGINS ||
'http://localhost:3000').split(',');
app.use(cors({
  origin: (origin, callback) => {
    if (!origin || allowedOrigins.includes(origin)) return callback(null,
true);
    callback(new Error('CORS: доступ заборонено'));
  },
  credentials: true
}));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/api/auth', authRoutes);
app.use('/api/students', studentRoutes);
app.use('/api/lessons', lessonRoutes);
app.use('/api/attendance', attendanceRoutes);
app.use('/api/users', userRoutes);

app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'index.html'));
});

app.get('/checkin', (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'checkin.html'));
});

app.use('/api', (req, res) => {
  res.status(404).json({ error: 'Маршрут не знайдено' });
});

app.use((req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'index.html'));
});

const PORT = process.env.PORT || 3000;

async function startServer() {
  try {
    await pool.query('SELECT 1');
    console.log('Підключено до бази даних PostgreSQL');
  } catch (err) {
    console.error('Не вдалося підключитися до БД:', err.message);
    process.exit(1);
  }

  try {
    await pool.query(`ALTER TABLE lessons ADD COLUMN IF
NOT EXISTS qr_code TEXT`);
  } catch (err) {
    console.error('Міграція qr_code:', err.message);
  }

  try {
    await pool.query(`
```

```
        CREATE TABLE IF NOT EXISTS refresh_tokens (
          id SERIAL PRIMARY KEY,
          user_id INTEGER REFERENCES users(id) ON DELETE
CASCADE,
          token TEXT NOT NULL UNIQUE,
          expires_at TIMESTAMP NOT NULL,
          created_at TIMESTAMP DEFAULT NOW()
        )
      `);
  } catch (err) {
    console.error('Міграція refresh_tokens:', err.message);
  }

  app.listen(PORT, () => {
    console.log(`Сервер запущено: http://localhost:${PORT}`);
  });
}

startServer();
```

```
const { Pool } = require('pg');
require('dotenv').config();

const pool = new Pool({
  host: process.env.DB_HOST,
  port: process.env.DB_PORT,
  database: process.env.DB_NAME,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  max: 10,
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 2000,
});

pool.on('error', (err) => {
  console.error('Помилка підключення до БД:', err);
});

module.exports = pool;
```

```
const jwt = require('jsonwebtoken');

function authMiddleware(req, res, next) {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {
    return res.status(401).json({ error: 'Токен відсутній' });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(403).json({ error: 'Невалідний токен' });
  }
}

function adminOnly(req, res, next) {
  if (req.user.role !== 'admin') {
    return res.status(403).json({ error: 'Доступ тільки для
адміністратора' });
  }
  next();
}

module.exports = { authMiddleware, adminOnly };
```

```
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const crypto = require('crypto');
const pool = require('../config/db');
```

```
const ACCESS_EXPIRES = '1h';
const REFRESH_EXPIRES_DAYS = 7;

function generateAccessToken(user) {
  return jwt.sign(
    { id: user.id, email: user.email, role: user.role, name: user.name },
    process.env.JWT_SECRET,
    { expiresIn: ACCESS_EXPIRES }
  );
}

function setRefreshCookie(res, token) {
  res.cookie('refresh_token', token, {
    httpOnly: true,
    secure: process.env.NODE_ENV === 'production',
    sameSite: 'strict',
    maxAge: REFRESH_EXPIRES_DAYS * 24 * 60 * 60 * 1000,
    path: '/api/auth'
  });
}

async function login(req, res) {
  const { email, password } = req.body;
  if (!email || !password) return res.status(400).json({ error: 'Email та пароль обов\'язкові' });

  try {
    const result = await pool.query('SELECT * FROM users WHERE email = $1', [email]);
    const user = result.rows[0];
    if (!user) return res.status(401).json({ error: 'Невірний email або пароль' });

    const isValid = await bcrypt.compare(password, user.password);
    if (!isValid) return res.status(401).json({ error: 'Невірний email або пароль' });

    const accessToken = generateAccessToken(user);
    const refreshToken = crypto.randomBytes(64).toString('hex');
    const expiresAt = new Date(Date.now() + REFRESH_EXPIRES_DAYS * 24 * 60 * 60 * 1000);
    await pool.query(
      'INSERT INTO refresh_tokens (user_id, token, expires_at) VALUES ($1, $2, $3)',
      [user.id, refreshToken, expiresAt]
    );

    setRefreshCookie(res, refreshToken);
    res.json({ token: accessToken, user: { id: user.id, name: user.name, email: user.email, role: user.role } });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

async function refreshToken(req, res) {
  const rt = req.cookies?.refresh_token;
  if (!rt) return res.status(401).json({ error: 'Refresh token відсутній' });

  try {
    const result = await pool.query(
      `SELECT rt.token, u.id, u.name, u.email, u.role
       FROM refresh_tokens rt
       JOIN users u ON rt.user_id = u.id
       WHERE rt.token = $1 AND rt.expires_at > NOW()`,
      [rt]
    );
    if (!result.rows[0]) return res.status(401).json({ error: 'Refresh token недійсний або прострочений' });

    const user = result.rows[0];

    const newRefreshToken = crypto.randomBytes(64).toString('hex');
    const expiresAt = new Date(Date.now() + REFRESH_EXPIRES_DAYS * 24 * 60 * 60 * 1000);
    await pool.query('DELETE FROM refresh_tokens WHERE token = $1', [rt]);
    await pool.query(
```

```
      'INSERT INTO refresh_tokens (user_id, token, expires_at) VALUES ($1, $2, $3)',
      [user.id, newRefreshToken, expiresAt]
    );

    const newAccessToken = generateAccessToken(user);
    setRefreshCookie(res, newRefreshToken);
    res.json({ token: newAccessToken, user: { id: user.id, name: user.name, email: user.email, role: user.role } });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

async function logout(req, res) {
  const rt = req.cookies?.refresh_token;
  if (rt) {
    await pool.query('DELETE FROM refresh_tokens WHERE token = $1', [rt]).catch(() => {});
  }
  res.clearCookie('refresh_token', { path: '/api/auth' });
  res.json({ message: 'Вихід виконано' });
}

async function register(req, res) {
  const { name, email, password, role } = req.body;
  if (!name || !email || !password) return res.status(400).json({ error: 'Всі поля обов\'язкові' });

  try {
    const existing = await pool.query('SELECT id FROM users WHERE email = $1', [email]);
    if (existing.rows.length > 0) return res.status(409).json({ error: 'Користувач з таким email вже існує' });

    const hashedPassword = await bcrypt.hash(password, 10);
    const result = await pool.query(
      'INSERT INTO users (name, email, password, role) VALUES ($1, $2, $3, $4) RETURNING id, name, email, role',
      [name, email, hashedPassword, role || 'teacher']
    );
    res.status(201).json({ user: result.rows[0] });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

module.exports = { login, register, refreshToken, logout };

const pool = require('../config/db');
const ExcelJS = require('exceljs');

async function scanQR(req, res) {
  const { lesson_token, student_code } = req.body;

  if (!lesson_token || !student_code) {
    return res.status(400).json({ error: 'Токен заняття та код студента обов\'язкові' });
  }

  try {
    const lessonResult = await pool.query(
      'SELECT * FROM lessons WHERE qr_token=$1 AND qr_expires_at > NOW()',
      [lesson_token]
    );
    if (!lessonResult.rows[0]) {
      return res.status(400).json({ error: 'QR-код недійсний або застарів' });
    }
    const lesson = lessonResult.rows[0];

    const studentResult = await pool.query(
      'SELECT * FROM students WHERE student_code=$1',
      [student_code]
    );
    if (!studentResult.rows[0]) {
      return res.status(404).json({ error: 'Студента не знайдено' });
```

```
    }

    const student = studentResult.rows[0];


    if (lesson.group_name && student.group_name &&
lesson.group_name !== student.group_name) {
      return res.status(403).json({
        error: `Студент групи ${student.group_name} не може
відвідувати заняття групи ${lesson.group_name}`
      });
    }


    const now = new Date();
    const dateStr = lesson.lesson_date instanceof Date
      ? lesson.lesson_date.toISOString().slice(0, 10)
      : String(lesson.lesson_date).slice(0, 10);
    const lessonStart = new Date(`${dateStr}T${lesson.start_time}`);
    const diffMinutes = (now - lessonStart) / 60000;
    const status = diffMinutes > 15 ? 'late' : 'present';


    const result = await pool.query(
      `INSERT INTO attendance (lesson_id, student_id, status)
       VALUES ($1, $2, $3)
       ON CONFLICT (lesson_id, student_id) DO UPDATE SET
scanned_at=NOW(), status=$3
       RETURNING *`,
      [lesson.id, student.id, status]
    );


    res.json({
      message: status === 'present' ? 'Відвідування зафіксовано' :
'Відвідування зафіксовано (запізнення)',
      student_name: student.full_name,
      status,
      scanned_at: result.rows[0].scanned_at
    });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

function buildAttendanceQuery(group_name, date_from, date_to) {
  const params = [];

  let query = `
    SELECT
      s.full_name, s.student_code, s.group_name,
      l.subject, l.lesson_date, l.start_time,
      COALESCE(a.status, 'absent') as status,
      a.scanned_at
    FROM lessons l
    CROSS JOIN students s
    LEFT JOIN attendance a ON a.lesson_id = l.id AND a.student_id
= s.id
    WHERE (
      l.group_name IS NULL OR l.group_name = '' OR
      s.group_name IS NULL OR s.group_name = '' OR
      l.group_name = s.group_name
    )
  `;
  if (group_name) { params.push(group_name); query += ` AND
s.group_name = $${params.length}`; }
  if (date_from)  { params.push(date_from);  query += ` AND
l.lesson_date >= $${params.length}`; }
  if (date_to)    { params.push(date_to);    query += ` AND
l.lesson_date <= $${params.length}`; }
  query += ' ORDER BY l.lesson_date DESC, s.full_name';
  return { query, params };
}

async function getReport(req, res) {
  const { group_name, date_from, date_to } = req.query;
  try {
    const { query, params } = buildAttendanceQuery(group_name,
date_from, date_to);
    const result = await pool.query(query, params);
    res.json(result.rows);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
  }
```

```
}

async function exportExcel(req, res) {
  const { group_name, date_from, date_to } = req.query;

  try {
    const { query, params } = buildAttendanceQuery(group_name, date_from, date_to);

    const result = await pool.query(query, params);

    const workbook = new ExcelJS.Workbook();
    const sheet = workbook.addWorksheet('Відвідування');

    sheet.columns = [
      { header: 'ПІБ студента',  key: 'full_name',    width: 30 },
      { header: 'Код студента',  key: 'student_code', width: 14 },
      { header: 'Група',         key: 'group_name',   width: 12 },
      { header: 'Предмет',       key: 'subject',      width: 24 },
      { header: 'Дата',          key: 'lesson_date',  width: 14 },
      { header: 'Початок',       key: 'start_time',   width: 10 },
      { header: 'Статус',        key: 'status',       width: 14 },
      { header: 'Час відмітки',  key: 'scanned_at',   width: 20 },
    ];

    sheet.getRow(1).eachCell(cell => {
      cell.font = { bold: true, color: { argb: 'FFFFFFFF' } };
      cell.fill = { type: 'pattern', pattern: 'solid', fgColor: { argb: 'FF1A73E8' } };
      cell.alignment = { horizontal: 'center' };
    });

    const statusMap = { present: 'Присутній', late: 'Запізнення', absent: 'Відсутній' };
    const colorMap  = { present: 'FFE6F4EA', late: 'FFFEF7E0',    absent: 'FFFCE8E6' };

    result.rows.forEach(r => {
      const row = sheet.addRow({
        full_name:    r.full_name,
        student_code: r.student_code,
        group_name:   r.group_name || '-',
        subject:      r.subject,
        lesson_date:  new Date(r.lesson_date).toLocaleDateString('uk-UA'),
        start_time:   r.start_time,
        status:       statusMap[r.status],
        scanned_at:   r.scanned_at ? new Date(r.scanned_at).toLocaleString('uk-UA') : '-',
      });
      const statusCell = row.getCell('status');
      statusCell.fill = { type: 'pattern', pattern: 'solid', fgColor: { argb: colorMap[r.status] } };
    });

    res.setHeader('Content-Type', 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet');
    res.setHeader('Content-Disposition', 'attachment; filename="attendance.xlsx"');
    await workbook.xlsx.write(res);
    res.end();
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка експорту' });
  }
}

async function getStats(req, res) {
  try {
    const totalStudents = await pool.query('SELECT COUNT(*) FROM students');
    const totalLessons  = await pool.query('SELECT COUNT(*) FROM lessons');
    const present = await pool.query("SELECT COUNT(*) FROM attendance WHERE status='present'");
    const late    = await pool.query("SELECT COUNT(*) FROM attendance WHERE status='late'");
    const absent  = await pool.query(`
      SELECT COUNT(*) FROM lessons l
      CROSS JOIN students s
      LEFT JOIN attendance a ON a.lesson_id=l.id AND a.student_id=s.id
      WHERE a.id IS NULL
```

```
        AND (
          l.group_name IS NULL OR l.group_name = '' OR
          s.group_name IS NULL OR s.group_name = '' OR
          l.group_name = s.group_name
        )
    `);
    const recentLessons = await pool.query(`
      SELECT l.subject, l.group_name, l.lesson_date,
        COUNT(a.id) FILTER (WHERE a.status IN ('present','late')) as present,
        COUNT(a.id) FILTER (WHERE a.status='late') as late
      FROM lessons l
      LEFT JOIN attendance a ON a.lesson_id = l.id
      GROUP BY l.id ORDER BY l.lesson_date DESC LIMIT 10
    `);
    res.json({
      totalStudents: parseInt(totalStudents.rows[0].count),
      totalLessons:  parseInt(totalLessons.rows[0].count),
      present:       parseInt(present.rows[0].count),
      late:          parseInt(late.rows[0].count),
      absent:        parseInt(absent.rows[0].count),
      recentLessons: recentLessons.rows,
    });
  } catch (err) {
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

module.exports = { scanQR, getReport, exportExcel, getStats };
```

```
const pool = require('../config/db');
const QRCode = require('qrcode');
const XLSX = require('xlsx');

async function getAllStudents(req, res) {
  try {
    const result = await pool.query('SELECT * FROM students ORDER BY full_name');
    res.json(result.rows);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

async function getStudentById(req, res) {
  try {
    const result = await pool.query('SELECT * FROM students WHERE id = $1', [req.params.id]);
    if (!result.rows[0]) return res.status(404).json({ error: 'Студента не знайдено' });
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

async function createStudent(req, res) {
  const { full_name, student_code, group_name, email } = req.body;

  if (!full_name || !student_code) {
    return res.status(400).json({ error: 'ПІБ та код студента обов\'язкові' });
  }

  try {
    const existing = await pool.query('SELECT id FROM students WHERE student_code = $1', [student_code]);
    if (existing.rows.length > 0) {
      return res.status(409).json({ error: 'Студент з таким кодом вже існує' });
    }

    const qrData = JSON.stringify({ type: 'student', code: student_code });
    const qrCode = await QRCode.toDataURL(qrData);

    const result = await pool.query(
```

```
      'INSERT INTO students (full_name, student_code, group_name, email, qr_code) VALUES ($1, $2, $3, $4, $5) RETURNING *',
      [full_name, student_code, group_name, email, qrCode]
    );

    res.status(201).json(result.rows[0]);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

async function updateStudent(req, res) {
  const { full_name, group_name, email } = req.body;
  try {
    const result = await pool.query(
      'UPDATE students SET full_name=$1, group_name=$2, email=$3 WHERE id=$4 RETURNING *',
      [full_name, group_name, email, req.params.id]
    );
    if (!result.rows[0]) return res.status(404).json({ error: 'Студента не знайдено' });
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

async function deleteStudent(req, res) {
  try {
    const result = await pool.query('DELETE FROM students WHERE id=$1 RETURNING id', [req.params.id]);
    if (!result.rows[0]) return res.status(404).json({ error: 'Студента не знайдено' });
    res.json({ message: 'Студента видалено' });
  } catch (err) {
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

async function getStudentQR(req, res) {
  try {
    const result = await pool.query('SELECT qr_code, full_name FROM students WHERE id=$1', [req.params.id]);
    if (!result.rows[0]) return res.status(404).json({ error: 'Студента не знайдено' });
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

async function getStudentStats(req, res) {
  try {
    const id = req.params.id;
    const student = await pool.query('SELECT * FROM students WHERE id=$1', [id]);
    if (!student.rows[0]) return res.status(404).json({ error: 'Студента не знайдено' });
    const group = student.rows[0].group_name;
    const stats = await pool.query(`
      SELECT
        COUNT(DISTINCT l.id) as total_lessons,
        COUNT(*) FILTER (WHERE a.status='present') as present,
        COUNT(*) FILTER (WHERE a.status='late') as late,
        COUNT(DISTINCT l.id) - COUNT(a.id) as absent
      FROM lessons l
      LEFT JOIN attendance a ON a.lesson_id=l.id AND a.student_id=$1
      WHERE ($2='' OR $2 IS NULL OR l.group_name='' OR l.group_name IS NULL OR l.group_name=$2)
    `, [id, group]);
    const recent = await pool.query(`
      SELECT l.subject, l.lesson_date, l.group_name, a.status, a.scanned_at
      FROM lessons l
      LEFT JOIN attendance a ON a.lesson_id=l.id AND a.student_id=$1
      WHERE ($2='' OR $2 IS NULL OR l.group_name='' OR l.group_name IS NULL OR l.group_name=$2)
      ORDER BY l.lesson_date DESC LIMIT 10
    `, [id, group]);
```

```
    res.json({ student: student.rows[0], stats: stats.rows[0], recent: recent.rows });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

async function importStudents(req, res) {
  const { students } = req.body;
  if (!Array.isArray(students) || students.length === 0)
    return res.status(400).json({ error: 'Масив студентів порожній' });
  const results = { added: 0, skipped: 0, errors: [] };
  for (const s of students) {
    if (!s.full_name || !s.student_code) { results.skipped++; results.errors.push('Пропущено: немає ПІБ або коду'); continue; }
    try {
      const ex = await pool.query('SELECT id FROM students WHERE student_code=$1', [s.student_code]);
      if (ex.rows.length > 0) { results.skipped++; results.errors.push(`${s.student_code} вже існує`); continue; }
      const qrData = JSON.stringify({ type: 'student', code: s.student_code });
      const qrCode = await QRCode.toDataURL(qrData);
      await pool.query('INSERT INTO students (full_name,student_code,group_name,email,qr_code) VALUES ($1,$2,$3,$4,$5)',
        [s.full_name, s.student_code, s.group_name||'', s.email||'', qrCode]);
      results.added++;
    } catch(err) { results.skipped++; results.errors.push(`${s.student_code}: помилка`); }
  }
  res.json(results);
}

async function importStudentsExcel(req, res) {
  if (!req.file) return res.status(400).json({ error: 'Файл не завантажено' });
  try {
    const workbook = XLSX.read(req.file.buffer, { type: 'buffer' });
    const sheetName = workbook.SheetNames[0];
    const sheet = workbook.Sheets[sheetName];
    const rows = XLSX.utils.sheet_to_json(sheet, { defval: '' });

    if (!rows.length) return res.status(400).json({ error: 'Файл порожній або не має даних' });

    const normalize = (row) => {
      const n = {};
      for (const key of Object.keys(row)) n[key.trim().toLowerCase()] = String(row[key] ?? '').trim();
      return n;
    };

    const colMap = (row) => {
      const r = normalize(row);
      return {
        full_name:    r['піб'] || r['full_name'] || r['піб студента'] || '',
        student_code: r['код'] || r['student_code'] || r['код студента'] || '',
        group_name:   r['група'] || r['group_name'] || r['назва групи'] || '',
        email:        r['email'] || r['ел. пошта'] || '',
      };
    };

    const students = rows.map(colMap).filter(s => s.full_name && s.student_code);
    if (!students.length) return res.status(400).json({ error: 'Не знайдено колонок ПІБ та Код. Перевірте заголовки файлу.' });

    const results = { added: 0, skipped: 0, errors: [] };
    for (const s of students) {
      try {
        const ex = await pool.query('SELECT id FROM students WHERE student_code=$1', [s.student_code]);
        if (ex.rows.length > 0) { results.skipped++; results.errors.push(`${s.student_code} вже існує`); continue; }
        const qrData = JSON.stringify({ type: 'student', code: s.student_code });
        const qrCode = await QRCode.toDataURL(qrData);
        await pool.query('INSERT INTO students (full_name,student_code,group_name,email,qr_code) VALUES ($1,$2,$3,$4,$5)',
```

```
            [s.full_name, s.student_code, s.group_name, s.email, qrCode]);
            results.added++;
        } catch (err) { results.skipped++; results.errors.push(`${s.student_code}: помилка`); }
      }
      res.json(results);
    } catch (err) {
      console.error(err);
      res.status(500).json({ error: 'Помилка читання файлу' });
    }
}

module.exports = { getAllStudents, getStudentById, createStudent, updateStudent, deleteStudent, getStudentQR, getStudentStats, importStudents, importStudentsExcel };

const pool = require('../config/db');
const QRCode = require('qrcode');
const crypto = require('crypto');

async function getAllLessons(req, res) {
  try {
    const result = await pool.query(`
      SELECT l.*, u.name as teacher_name
      FROM lessons l
      LEFT JOIN users u ON l.teacher_id = u.id
      ORDER BY l.lesson_date DESC, l.start_time DESC
    `);
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

async function getLessonById(req, res) {
  try {
    const result = await pool.query(`
      SELECT l.*, u.name as teacher_name
      FROM lessons l
      LEFT JOIN users u ON l.teacher_id = u.id
      WHERE l.id = $1
    `, [req.params.id]);
    if (!result.rows[0]) return res.status(404).json({ error: 'Заняття не знайдено' });
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

async function createLesson(req, res) {
  const { subject, group_name, lesson_date, start_time, end_time } = req.body;

  if (!subject || !lesson_date || !start_time || !end_time) {
    return res.status(400).json({ error: 'Заповніть всі обов\'язкові поля' });
  }

  try {
    const result = await pool.query(
      'INSERT INTO lessons (subject, teacher_id, group_name, lesson_date, start_time, end_time) VALUES ($1, $2, $3, $4, $5, $6) RETURNING *',
      [subject, req.user.id, group_name, lesson_date, start_time, end_time]
    );
    res.status(201).json(result.rows[0]);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

async function generateLessonQR(req, res) {
  try {
    const lessonId = req.params.id;
    const lesson = await pool.query('SELECT * FROM lessons WHERE id=$1', [lessonId]);
    if (!lesson.rows[0]) return res.status(404).json({ error: 'Заняття не знайдено' });
```

```
    const l = lesson.rows[0];

    if (req.user.role !== 'admin' && l.teacher_id !== req.user.id) {
      return res.status(403).json({ error: 'Можна генерувати QR тільки для власних занять' });
    }

    if (l.qr_token && l.qr_expires_at && new Date(l.qr_expires_at) > new Date() && l.qr_code) {
      return res.json({ qr_code: l.qr_code, expires_at: l.qr_expires_at, token: l.qr_token });
    }

    const token = crypto.randomBytes(32).toString('hex');
    const expiresAt = new Date(Date.now() + 60 * 60 * 1000);

    const qrData = JSON.stringify({ type: 'lesson', token, lessonId });
    const qrCode = await QRCode.toDataURL(qrData);

    await pool.query(
      'UPDATE lessons SET qr_token=$1, qr_expires_at=$2, qr_code=$3 WHERE id=$4',
      [token, expiresAt, qrCode, lessonId]
    );

    res.json({ qr_code: qrCode, expires_at: expiresAt, token });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

async function getLessonAttendance(req, res) {
  try {
    const result = await pool.query(`
      SELECT a.*, s.full_name, s.student_code, s.group_name
      FROM attendance a
      JOIN students s ON a.student_id = s.id
      WHERE a.lesson_id = $1
      ORDER BY a.scanned_at
    `, [req.params.id]);
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

async function updateLesson(req, res) {
  const { subject, group_name, lesson_date, start_time, end_time } = req.body;
  if (!subject || !lesson_date || !start_time || !end_time) {
    return res.status(400).json({ error: 'Заповніть всі обов\'язкові поля' });
  }
  try {
    let query, params;
    if (req.user.role !== 'admin') {
      query = 'UPDATE lessons SET subject=$1,group_name=$2,lesson_date=$3,start_time=$4,end_time=$5 WHERE id=$6 AND teacher_id=$7 RETURNING *';
      params = [subject, group_name, lesson_date, start_time, end_time, req.params.id, req.user.id];
    } else {
      query = 'UPDATE lessons SET subject=$1,group_name=$2,lesson_date=$3,start_time=$4,end_time=$5 WHERE id=$6 RETURNING *';
      params = [subject, group_name, lesson_date, start_time, end_time, req.params.id];
    }
    const result = await pool.query(query, params);
    if (!result.rows[0]) return res.status(403).json({ error: 'Заняття не знайдено або немає прав' });
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

async function deleteLesson(req, res) {
  try {
    let query = 'DELETE FROM lessons WHERE id=$1';
```

```
    const params = [req.params.id];
    if (req.user.role !== 'admin') {
      query += ' AND teacher_id=$2';
      params.push(req.user.id);
    }
    query += ' RETURNING id';
    const result = await pool.query(query, params);
    if (!result.rows[0]) return res.status(403).json({ error: 'Заняття не знайдено або немає прав на видалення' });
    res.json({ message: 'Заняття видалено' });
  } catch (err) {
    res.status(500).json({ error: 'Помилка сервера' });
  }
}

module.exports = { getAllLessons, getLessonById, createLesson, updateLesson, generateLessonQR, getLessonAttendance, deleteLesson };
```

ДОДАТОК В

Таблиця В.1

Перелік ендпоінтів REST API системи

Ресурс	Метод	Шлях	Доступ	Опис
Автентифікація	POST	/api/auth/login	Публічно	Вхід; повертає access-токен, встановлює refresh-токен у httpOnly cookie
	POST	/api/auth/register	Лише admin	Реєстрація нового користувача
	POST	/api/auth/refresh	Публічно	Оновлення access-токена за refresh-токеном з cookie
	POST	/api/auth/logout	Публічно	Відкликання refresh-токена, очищення cookie
	GET	/api/auth/me	JWT	Інформація про поточного користувача
Студенти	GET	/api/students/	JWT	Список студентів (пагінація, фільтри)
	GET	/api/students/:id	JWT	Дані конкретного студента
	GET	/api/students/:id/qr	JWT	QR-код студента (base64)
	GET	/api/students/:id/stats	JWT	Статистика відвідуваності студента
	POST	/api/students/	JWT + admin	Створення студента
	POST	/api/students/import	JWT + admin	Пакетний імпорт із CSV
	POST	/api/students/import-excel	JWT + admin	Пакетний імпорт із Excel
	PUT	/api/students/:id	JWT + admin	Оновлення даних студента

Продовження таблиці В.1

	DELETE	/api/students/:id	JWT + admin	Видалення студента
Заняття	GET	/api/lessons/	JWT	Список занять (викладач бачить лише свої)
	GET	/api/lessons/:id	JWT	Дані заняття
	GET	/api/lessons/:id/attendance	JWT	Список відвідувань заняття
	POST	/api/lessons/	JWT	Створення заняття
	POST	/api/lessons/:id/qr	JWT + власник/admin	Генерація QR-коду заняття
	PUT	/api/lessons/:id/qr	JWT + власник/admin	Оновлення заняття
	DELETE	/api/lessons/:id	JWT + власник/admin	Видалення заняття
Відвідуваність	POST	/api/attendance/scan	Публічно / JWT	Реєстрація скану QR-коду
	GET	/api/attendance/report	JWT	Звіт відвідуваності
	GET	/api/attendance/export	JWT	Вивантаження звіту у Excel
	GET	/api/attendance/stats	JWT	Вивантаження звіту у Excel
Користувачі	GET	/api/users/	JWT + admin	Список облікових записів
	POST	/api/users/	JWT + admin	Створення нового облікового запису
	DELETE	/api/users/:id	JWT + admin	Видалення облікового запису