МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

АВТОМАТИЗОВАНА СИСТЕМА ПЛАНУВАННЯ РОЗКЛАДУ НАВЧАЛЬНИХ ЗАНЯТЬ

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ **Іван ТРУНОВ**

«__» _____ 2026 р.

Керівник роботи

д-рка техн. наук,
професорка _____ **Альона ШВЕД**

«__» _____ 2026 р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп'ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«26» грудня 2025 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувача

Трунова Івана

1. Тема кваліфікаційної роботи «Автоматизована система планування розкладу навчальних занять» затверджена наказом ректора ЧНУ ім. Петра Могили № 349 від «26» грудня 2025 р.
2. Строк представлення кваліфікаційної роботи «» червня 2026 р.
3. Очікуваний результат роботи та початкові дані якщо такі потрібні.

Очікуваним результатом є автоматизована система планування розкладу навчальних занять.

4. Перелік питань, що підлягають розробці:

– аналіз предметної області та існуючих аналогів;
– моделювання та конструювання ПЗ;

– проєктування структури бази даних;
– розробка інтерфейсу користувача;
– реалізація серверної та клієнтської частин вебзастосунку;
– тестування вебзастосунку.

5. Перелік графічних матеріалів: презентація

6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «13» січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: **Автоматизована система планування розкладу навчальних занять**

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КБР	26.12.2025	13.01.2026	Виконано
2.	Огляд літератури за темою роботи	14.01.2026	22.01.2026	Виконано
3.	Складання календарного плану КБР	25.01.2026	26.01.2026	Виконано
4.	Аналіз предметної області	27.01.2026	31.01.2026	Виконано
5.	Розробка проєктних рішень	09.02.2026	13.02.2026	Виконано
6.	Моделювання та конструювання ПЗ	13.02.2026	22.02.2026	Виконано
7.	Кодування розробленого ПЗ, розробка керівництва користувача	22.02.2026	16.03.2026	Виконано
8.	Тестування та апробація ПЗ, аналіз результатів тестування	22.03.2026	06.04.2026	Виконано
9.	Відгук керівника КБР	25.04.2026	28.04.2026	Виконано
10.	Оформлення КБР та презентації	01.05.2026	26.05.2026	Виконано
11.	Попередній захист	27.05.2026	27.05.2026	Виконано
12.	Завершення оформлення КБР та презентації	28.05.2026	04.06.2026	Виконано
13.	Рецензування КБР	05.06.2026	08.06.2026	
14.	Захист кваліфікаційної роботи			

Здобувач _____ **Іван ТРУНОВ**

«26» січня 2026 р.

Керівник роботи

д-рка техн. наук,

професорка _____ **Альона ШВЕД**

«26» січня 2026 р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

«Автоматизована система планування розкладу навчальних занять»

Здобувач 409 гр. : Трунов Іван

Керівник: д-рка техн. наук, професорка Альона Швед

У сучасну епоху стрімкий розвиток сучасних цифрових технологій значно впливає на автоматизацію задач і може суттєво впливати на різні аспекти людської діяльності. У сфері освіти планування навчальних занять є одним із найскладніших організаційних завдань і належить до класу NP-повних оптимізаційних задач. Існує багато жорстких і м'яких обмежень: навантаження викладачів, місткість аудиторій, специфіка дисциплін тощо. Створення спеціалізованої автоматизованої системи залишається важливим і актуальним завданням, оскільки присутня проблема відсутності вітчизняних програмних рішень, що поєднують гнучкість налаштувань та українську локалізацію.

Метою роботи є розробка автоматизованої системи планування навчальних занять для підвищення ефективності управління освітнім процесом.

Об'єкт – процес автоматизованого формування розкладу занять.

Предметом роботи виступають алгоритми, методи та програмні засоби автоматизації створення розкладу.

Кваліфікаційна робота складається із вступу, чотирьох розділів, висновків та переліку джерел посилання.

У вступі висвітлено актуальність розробки автоматизованої системи планування розкладу навчальних занять, визначено мету, завдання дослідження, об'єкт та предмет.

У першому розділі проведено аналіз предметної області та існуючих рішень у сфері автоматизованого планування розкладу. Представлено переваги та недоліки існуючих програмних рішень. Визначено структурні та функціональні особливості об'єкта кваліфікаційної роботи.

Другий розділ присвячено моделюванню об’єкта та предмету дослідження. Обгрунтовано методи та моделі які будуть використані для вирішення задачі. Складено специфікацію вимог до програмного забезпечення.

У третьому розділі розглядаються проєктні рішення обрані для розробки. Зазначено UML-діаграми, що моделюють роботу програмного забезпечення.

У четвертому розділі реалізується програмне забезпечення відповідно до сформованих вимог та за допомогою обраних технологій. Наведено результати виконаної роботи. Сформовано керівництво користувача та виконано навантажувальне тестування.

У висновках узагальнено результати роботи та визначено можливі перспективи подальшого розвитку проєкту.

Кваліфікаційна робота викладена на 88 сторінках машинописного тексту, складається із вступу, 4 розділів, загальних висновків, переліку джерел посилання з 25 найменувань та 3 додатків. Праця містить 10 таблиць та 33 рисунків.

Ключові слова: автоматизація розкладу, архітектура вебзастосунку, база даних, специфікація вимог, навчальний процес.

ABSTRACT

of the Bachelor's Thesis

"Automated System for Planning Training Schedules"

Student of group 409: Ivan Trunov

Supervisor: Doctor of Technical Sciences, Professor Alyona Shved

In today's world, the rapid development of modern digital technologies is having a significant impact on the automation of tasks and can substantially influence various aspects of human activity. In the field of education, lesson planning is one of the most complex organisational tasks and belongs to the class of NP-complete optimisation problems. There are many hard and soft constraints: lecturers' workloads, lecture hall capacity, subject-specific requirements, and so on. The creation of a specialised automated system remains an important and relevant task, as there is a lack of Ukrainian software solutions that combine flexible configuration options with Ukrainian localisation.

The goal of this work is to develop an automated system for planning lessons in order to improve the efficiency of educational process management.

The object of study is the process of automated timetable generation.

The subject of this work comprises the algorithms, methods and software tools used to automate timetable creation.

Qualification work consists of an introduction, four chapters, conclusions and a list of references.

The introduction highlights the relevance of developing an automated system for planning timetables for academic classes, and defines the aim, objectives, scope and subject of the research.

The first chapter analyses the subject area and existing solutions in the field of automated timetable planning. It outlines the advantages and disadvantages of existing software solutions. The structural and functional characteristics of the subject of this thesis are identified.

The second chapter is devoted to modelling the object and subject of the research. The methods and models to be used to solve the problem are justified. A specification of software requirements has been drawn up.

The third chapter examines the design solutions selected for development. UML diagrams modelling the operation of the software are provided.In the fourth chapter, the software is implemented in accordance with the established requirements and using the selected technologies. The results of the work carried out are presented. A user manual has been produced and load testing has been carried out.

The conclusions summarise the results of the work and identify possible prospects for the further development of the project.

The qualification work is presented in 88 pages of typed text and consists of an introduction, 4 chapters, general conclusions, a bibliography of 25 references, and 3 appendices. The work contains 10 tables and 33 figures.

Keywords: scheduling automation, educational process, web application architecture, database, software requirements specification.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – База даних

ОС – Операційна система

ПК – Персональний комп'ютер

ПЗ – Програмне забезпечення

СУБД – Система управління базами даних

API – Application Programming Interface (прикладний програмний інтерфейс)

CSP – Constraint Satisfaction Problem (задача задоволення обмежень)

ETP – Educational Timetabling Problem (освітня задача складання розкладу)

JWT – JSON Web Token (компактний стандарт токенів безпеки)

MRV – Minimum Remaining Values (евристика мінімального кількості життєздатних значень)

MVC – Model-View-Controller (архітектурний патерн «Модель-Представлення-Контролер»)

TSS – Timetable Scheduling Systems (системи планування розкладу)

UML – Unified Modeling Language (уніфікована мова моделювання)

XML – eXtensible Markup Language (розширювана мова розмітки)

ВСТУП

У сучасному світі стрімкий розвиток цифрових технологій суттєво впливає на автоматизацію задач у різних сферах діяльності людини. Складання розкладу навчальних занять є однією з найскладніших організаційних задач у сфері освіти та належить до класу NP-повних задач оптимізації. Це зумовлено необхідністю врахувати велику кількість жорстких та м'яких обмежень: навантаженість викладачів, місткість аудиторій та специфіки дисциплін.

Застосування сучасних технологій, таких як вебзастосунки, може значно полегшити цей процес і дозволити користувачам виконувати складання розкладу навчальних занять швидко і зручно.

Розробка власної автоматизованої системи планування розкладу навчальних занять дозволить:

– забезпечити доступ до ефективного розподілу ресурсів навчального закладу;

– зробити процес отримання інформації про розклад швидким і зручним.

Метою роботи є розробка автоматизованої системи планування навчальних занять для підвищення ефективності управління освітнім процесом.

Відповідно до мети визначено такі **завдання:**

– провести аналіз предметної області та існуючих програмних рішень;

– розробити специфікацію вимог до ПЗ;

– спроєктувати архітектуру вебзастосунку;

– розробити структуру бази даних відповідно до вимог;

– реалізувати серверну частину вебзастосунку;

– розробити клієнтську частину вебзастосунку;

– провести тестування системи та оцінити швидкість генерації розкладу.

Об'єкт – процес автоматизованого формування розкладу занять.

Предметом роботи є алгоритми, методи та програмні засоби автоматизації створення розкладу.

Незважаючи на існування рішень від таких компаній як «aSc Applied Software Consultants» (розробник aSc TimeTables) та «lalescu.ro» (система FET з відкритим кодом), більшість з них є високовартісними продуктами або ж мають складний інтерфейс, що потребує тривалого навчання. Це створює прогалину у наявності легких та адаптованих до українських стандартів систем, що зумовлює необхідність нової розробки.

Сфера застосування програмного забезпечення автоматизованого формування розкладу охоплює галузі, де постає задача впорядкування ресурсів (часу, аудиторій, людських ресурсів), спрямована на оптимізацію використання часу, персоналу (викладачів), аудиторій та навчальних планів.

Програмне забезпечення автоматизованого формування розкладу може бути адаптоване для вирішення задач управління персоналом, логістики, сфери послуг тощо.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Стан та тенденції розвитку систем автоматизованого планування навчального процесу

В сучасному інформаційному світі автоматизація стала однією з головних турбот розвитку у кожній сфері діяльності людини. В освіті ефективне управління закладом вищої або спеціалізованої освіти включає застосування систем автоматизації для організаційних процесів, серед яких однією з найскладніших і найтрудомісткіших процедур є формування освітньої діяльності. З кожним роком обсяги даних – кількість вибіркових курсів, кількість вимог і наявність спеціалізованих аудиторій, кількість підгруп і різних освітніх напрямків – зростають, і вимоги до академічного управління такого роду змінюються. Зі збільшенням потоків вхідних даних в систему традиційні методи планування – зокрема, табличні процесори (ТБ), такі як Microsoft Excel і Google Sheets втрачають свою ефективність. Недоліки цього підходу можна підсумувати наступним чином:

– висока ймовірність помилок, яка зумовлена тим що контроль за конфліктами інтересів повністю ведеться спеціалістом який займається формуванням розкладу, а не автоматично (виключити людський фактор неможливо);

– низька оперативність, адже при зміні зовнішніх обставин (звільнення викладача, хвороба персоналу тощо) неможливо швидко відформатувати графік через повільність ручного перерахунку спеціаліста;

– ізольованість даних, через яку викладачі та студенти змушені отримувати актуальну інформацію з докладенням суттєвих зусиль: за допомогою ручної передачі нового документа в електронному вигляді або ж його друку чи фото (єдиний інформаційний простір відсутній).

На основі вищезазначених проблем, значного розвитку набувають спеціалізовані системи автоматизації планування TSS (Timetable Scheduling

Systems) [1]. Аналіз наявного досвіду впровадження таких систем у закладах освіти дозволяє зробити висновок, що використання TSS значно підвищує ефективність менеджменту освітнього процесу.

Ручне керування розкладом є виснажливим і часто схильним до помилок. Адміністратори можуть витрачати цілі робочі тижні на створення плану без конфліктів, який задовольняє потреби кожного учасника навчального процесу. Система складання розкладів покликана скоротити час та зусилля необхідні від адміністратора. Використовуючи механізм планування, система самостійно аналізує дані студентів, викладачів та курсів. Цей перехід звільняє адміністраторів від виснажливої роботи та підвищує загальну ефективність. Продуктивність персоналу зростає, оскільки система бере на себе складніші процеси, а адміністратори витрачають час лише на необхідне налаштування автоматизації. Також, цифрова система управління розкладом не лише економить час, але й раціональніше використовує простір та ресурси, адже гарантує відсутність випадків подвійного бронювання аудиторій або ж накладок у графіках викладачів чи груп.

Спеціалізовані системи легко адаптуються до зміни кількості студентів, викладачів або ж виникнення непередбачуваних обставин (наприклад, викладач захворів). Це дозволяє без необхідності повної перебудови структури даних швидко моделювати нові сценарії розподілу навантаження, нові розклади, забезпечувати гнучкість та масштабованість навчального процесу.

Виявляючи та усуваючи проблеми за допомогою ПЗ, цифрове планування зменшує кількість людських помилок, захищає академічну доброчесність та забезпечує ефективну роботу. В результаті досягається надійне планування як для студентів, так і для працівників закладу.

Також, важливим аспектом розробки систем автоматичного планування розкладу є вибір платформи для їх реалізації. Оскільки, в сучасних умовах розробки велика увага приділяється кросплатформенності та масштабованості,

найбільш перспективним напрямком для розробки рішень TSS є вебтехнології. Переваги такого підходу до розробки включають:

– кросплатформенність та доступність;
– масштабованість;
– централізоване управління даними.

Вебзастосунки не потребують встановлення спеціалізованого програмного забезпечення (ПЗ), адже доступ здійснюється через будь-який сучасний браузер та з будь-якого пристрою. Впровадження нових функцій або ж зміна алгоритмів в свою чергу не потребують перевстановлення ПЗ користувачами системи, адже всі зміни відбуваються на серверній стороні. Усі зміни, які вносяться адміністратором чи методистом миттєво стають доступними для всіх клієнтів системи (учасників навчального процесу), що в свою чергу вирішує проблему ізольованості даних, яка притаманна для методів розробки розкладу за допомогою табличних процесорів.

Таким чином, сучасні тенденції розвитку систем автоматизованого планування навчального процесу свідчать про поступовий відхід від застарілих ручних методів на користь інтелектуальних вебзастосунків. Впровадження таких вебсистем дозволяє усунути конфлікти у графіках, забезпечити масштабованість та доступність даних у реальному часі для всіх учасників освітнього процесу [2, 3].

1.2 Аналіз існуючих аналогів та їх характеристика

Для розробки автоматизованої системи планування навчальних занять важливо розглянути існуючі аналоги, щоб спрогнозувати власне проєктування вебдодатка. Сьогоднішній ринок освітніх технологій представлений як і універсальними табличними процесорами, так і вузькоспеціалізованими системами на базі еволюційних алгоритмів. Для виконання порівняння та аналізу існуючих рішень було обрано системи, які представляють різні підходи до реалізації: комерційну систему aSc TimeTables [4], open-source проєкт FET [5] та традиційний метод, тобто табличний процесор Excel [6]. Такий аналіз необхідний

для вивчення сучасних підходів до реалізації автоматизації планування розкладу навчальних занять, оцінки функціонального наповнення конкурентів і відповідно, виявлення наявних у них переваг та недоліків, які б дозволили визначити можливості для створення власного і конкурентного вебсайту.

aSc TimeTables [4]

aSc TimeTables (рис. 1.1) – це комерційне ПЗ, розроблене компанією aSc Applied Software Consultants. Даний продукт пропонує автоматичне складання розкладу для закладів середньої, професійно-технічної та вищої освіти. Згідно інформації на офіційному сайті розробників, програма використовується у понад 170 країнах світу і являє собою досить потужний та гнучкий інструмент, який займає одну з провідних позицій на ринку TSS. Висока ефективність даного рішення зумовлена використанням складних евристичних методів.

Таблиця 1.1 – Опис aSc TimeTables

Назва	aSc TimeTables
Виробник	aSc Applied Software Consultants
Архітектура	Настільний застосунок, вебзастосунок, клієнт-сервер
Мова реалізації	C++, C#
Функції	– автоматичне генерування розкладу; – контроль конфліктів інтересів у реальному часі; – експорт та імпорт даних в різних форматах (Excel \ XML); – управління замінами викладачів; – управління лікарняними та вихідними днями; – система сповіщень через мобільний застосунок EduPage, який входить до екосистеми застосунків від компанії та являє собою хмарну програму для дистанційного навчання.
Переваги	– потужний інтелектуальний генератор розкладу; – швидка робота генерації; – наявність української локалізації; технічна підтримка.

Кінець таблиці 1.1

Недоліки	– застарілий та складний інтерфейс; – висока вартість платної ліцензії; – платна технічна підтримка; – високий поріг входження для нових користувачів; – основна орієнтація на Windows-клієнт та досить урізана функціональністю вебверсія для інших платформ.

Підсумовуючи, aSc TimeTables добре спроєктований інструмент з потужним алгоритмом автоматичної генерації, але його використання має суттєві обмеження для багатьох навчальних закладів через високу вартість ліцензійованого використання, тобто щорічними витратами на оновлення доступу до програмного продукту. Це робить систему недоступною для невеликих приватних шкіл, класів чи курсів, які потребують гнучкого інструменту без значних фінансових витрат, але мають обмежений бюджет. Також, незважаючи на широке функціональне наповнення, інтерфейс програми є застарілим і перевантаженим складними меню та налаштуваннями, змушуючи персонал витрачати значний час на навчання або ж залучати сторонніх фахівців для налаштування системи.

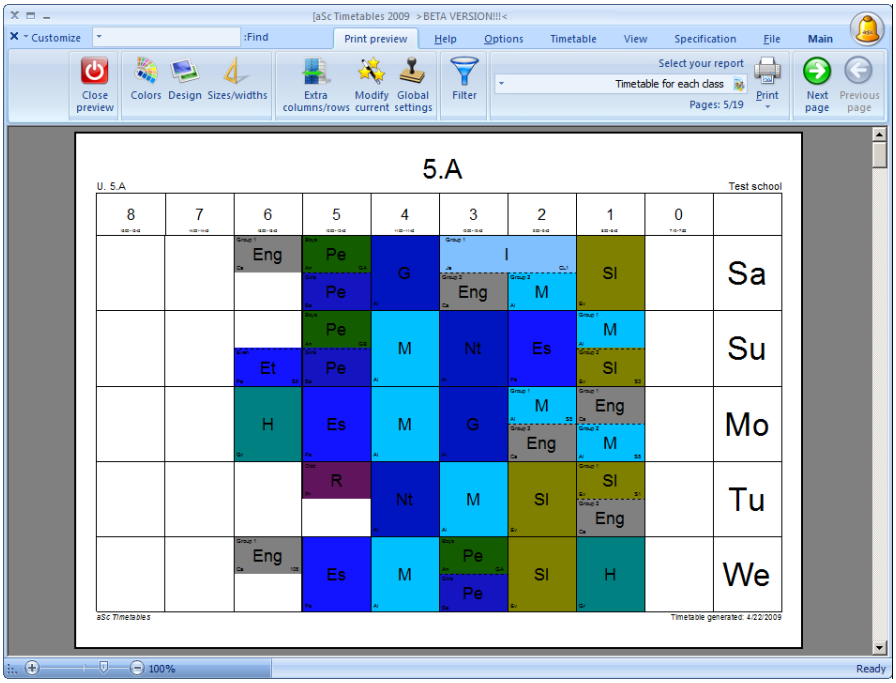

Рисунок 1.1 – Вигляд інтерфейсу застосунку aSc TimeTables

FET [5]

FET (рис. 1.2) – це безкоштовне ПЗ з відкритим кодом (Open Source), призначене для автоматичної генерації розкладу для закладів початкової, середньої чи вищої освіти. Програма написана на мові C++ із використанням кросплатформенної бібліотеки Qt і набрала свою популярність у минулому через свою прозорість та доступ до детальних ручних налаштувань алгоритму.

Таблиця 1.2 – Опис FET

Назва	FET
Виробник	Liviu Lalescu, Volker Dirr (Спільнота розробників)
Архітектура	Настільний застосунок (Windows або Linux)
Мова реалізації	C++
Функції	– автоматичне генерування розкладу з використанням еволюційного алгоритму; – широка підтримка часових та просторових обмежень; – експорт даних через XML, CSV, HTML.
Переваги	– велика кількість локалізацій через Open-Source основу; – безкоштовний та відкритий код; не потребує підключення до інтернету.
Недоліки	– не має хмарної синхронізації або спільної роботи онлайн; – дуже застарілий, складний інтерфейс; довга генерація розкладу, яка напряму залежить від технічного обладнання клієнта.

FET є взірцевим інструментом з погляду на його відкритість та налаштованість, але оскільки початковий випуск програми датується 2002 роком і після цього інтерфейс не отримував значних оновлень, він не відповідає сучасним стандартам UX/UI-дизайну. Робочий простір програми виглядає перенасиченим технічними термінами і складними діалоговими вікнами, що з

погляду звичайного користувача, без поглиблених технічних знань є дуже поганим досвідом.

Більше того, без вебверсії та нового способу візуалізації результатів, FET залишається обмеженим у використанні у сучасних умовах, де користувацький досвід та мобільність стали основними критеріями при виборі ПЗ. Сучасний користувач потребує від системи інтуїтивно зрозумілого інтерфейсу та можливість працювати з даними без глибоких технічних знань та багаторівневих інструкцій, з можливістю редагувати розклад з будь-якого пристрою без необхідності локального встановлення ПЗ. FET містить відносно небагато інструментів графічного представлення даних і може ускладнити швидкий аналіз конфліктів або завантаження на аудиторію та групу в реальному часі. Таким чином, незважаючи на потужний обчислювальний потенціал, FET поступово втрачає свою актуальність.

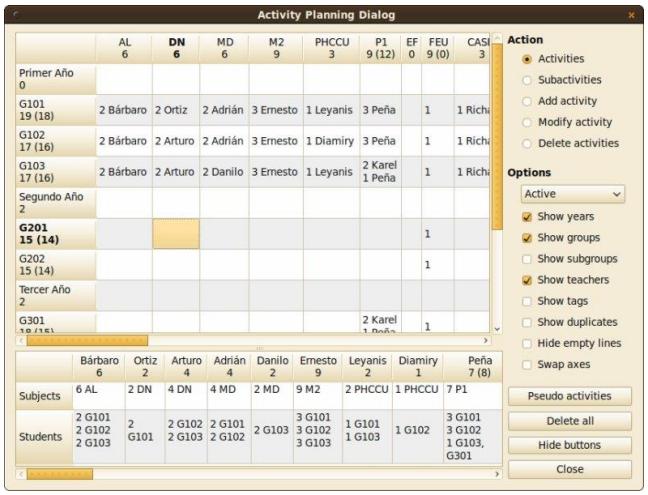

Рисунок 1.2 – Вигляд інтерфейсу застосунку FET

Microsoft Excel [6]

Excel (або ж його хмарний аналог Google Sheets) (рис. 1.3) – ТБ все ще залишаються одним із основних методів для формування розкладу в навчальних закладах, попри наявність спеціалізованого ПЗ. Хоча такі традиційні підходи до роботи з даними створюють неймовірно ефективну та індивідуалізовану структуру даних, це все ще дуже трудомістко і не пропонує інтелектуальної автоматизації.

Таблиця 1.3 – Опис Microsoft Excel

Назва	Microsoft Excel
Виробник	Microsoft Corporation
Архітектура	Настільний застосунок, SaaS (Office 365)
Мова реалізації	C++, C#
Функції	– табулювання даних; – широка візуалізація даних у вигляді діаграм; – формули.
Переваги	– доступність та гнучкість; – можливість довільного форматування даних; – підтримка різноманітних макросів; передвстановлено на більшості ОС (операційна система) сімейства Windows.
Недоліки	– відсутність алгоритму автоматичного генерування розкладу; – ручний контроль конфліктів інтересів; – висока складність використання з великими обсягами даних; високий ризик помилок через людський фактор.

ТБ, як Microsoft Excel вже давно використовуються для складання розкладу навчальних занять і залишаються традиційним методом у багатьох освітніх установах. Однак, через свої критичні недоліки – включаючи трудомісткість та залежність від людського фактора (адміністратор має самостійно відстежувати зайнятість кожного викладача та аудиторії), створює потребу до переходу від статичних систем на динамічні застосунки із автоматичною логікою. Адміністратор змушений вручну відстежувати тисячі перехресних зв'язків: зайнятість кожного викладача, доступність аудиторного фонду, специфіку навчальних планів та побажання студентських груп. Коли дані вводяться в таблицю, будь-яка механічна помилка може викликати конфлікти, які часто діагностуються вже під час навчального процесу. Також, хоча хмарні версії (Excel Online) частково вирішують проблему мобільності та доступності, але поки що не

пропонують таку ж зручність перегляду даних, яку пропонують більш спеціалізовані продукти. Таким чином, простота використання ТБ Excel нівелюється низькою адаптивністю та відсутністю автоматизації.

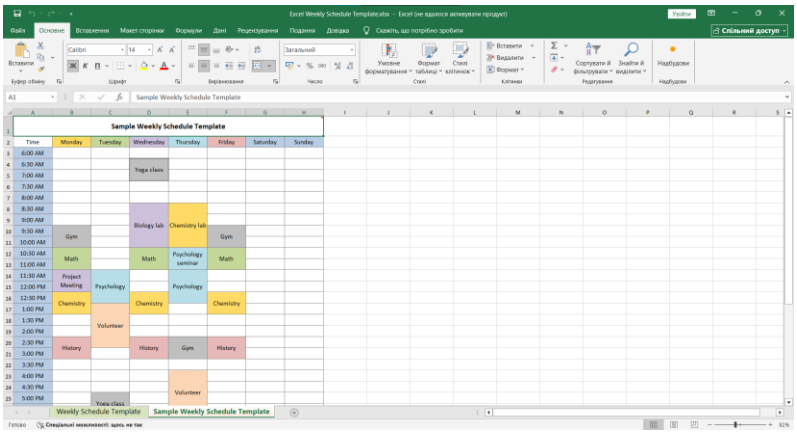

Рисунок 1.3 – Приклад складання розкладу занять у застосунку Excel

Згідно до виконаного аналізу існуючих рішень, можна зробити висновок, що поточний ринок представлений двома різними підходами до складання розкладу навчальних занять. aSc TimeTables, FET представляють комерційні та open-source проєкти, які володіють значним алгоритмічним потенціалом, але також потребують від користувача досить високої технічної бази або ж велику плату за ліцензійне використання. Також, універсальні інструменти типу Microsoft Excel є найбільш поширеним та доступним методом, але досить неефективним через повну відсутність автоматизації. Таким чином, існує чітко визначена проблема в розробці сучасного вебзастосунку, який би надавав користувачам інтуїтивно зрозумілий інтерфейс, який не потребував би спеціального навчання та містив у собі інтелектуальні інструменти перевірки колізій та автоматизації складання розкладу і можливістю отримання доступу до даних з будь-якого пристрою без необхідності встановлення додаткового ПЗ.

1.3 Аналіз структурних і функціональних особливостей

Очевидно, якщо планування навчальних сесій розглядати як автоматизований об'єкт, це багатофакторна система з жорсткими обмеженнями ресурсів. Основною особливістю об'єкта є одночасна синхронізація чотирьох

важливих сутностей: академічних груп, викладацького складу, аудиторій та часових слотів. Однак, зміна параметрів однієї сутності (наприклад, перенесення заняття викладача на інший час) викликає ланцюгову реакцію в розкладах груп та зайнятості аудиторій, тому для правильного функціонування системи слід враховувати два типи функціональних обмежень [7].

Жорсткі обмеження (Hard Constraints) – включають в себе фізичну неможливість перебувати у двох різних станах одночасно. Наприклад: викладач або ж група не можуть перебувати у двох місцях одночасно; відповідність типу аудиторії до вимог заняття, (наприклад наявність комп'ютерів для занять з програмування); місткість приміщення відносно кількості учнів. Якщо такі обмеження порушуються, розклад фактично є неправильним.

М'які обмеження (Soft Constraints) – включають в себе побажання щодо якості розкладу. Наприклад, сюди можна включити відсутність вікон у студентів, тобто виключення випадка коли їм потрібно чекати весь день, адже між першою і п'ятою парою вони не мають навантаження. Так само м'яким обмеженням є навантаженість викладачів та груп протягом тижня, наприклад, в більшості випадків прийнятним вважалось би рівномірне навантаження на кожен робочий день, а не 5 пар в понеділок і по одній парі на кожен інший день. Забезпечуючи оптимізацію таких обмежень, система гарантує якість розкладу та комфорт освітнього процесу.

Для реалізації описаних алгоритмів та взаємодії користувача з ними необхідно представити вебзастосунок у вигляді таких підсистем:

– модуль автентифікації та ролей користувачів, який має визначати функціональні можливості користувача згідно до його ролі;

– модуль управління нормативно-довідковою інформацією, який відповідає за структурне представлення даних і їх характеристик, а саме про викладачів, навчальних дисциплін, груп та аудиторій;

– програмний компонент автоматичної генерації який на основі заданої інформації та обмежень здійснює підбір оптимального варіанту розкладу;

– інформаційний модуль, який забезпечує візуалізацію розкладу для користувачів системи та надає можливість фільтрації і експорту даних.

Аналіз архітектурних особливостей

Реалізація системи у вигляді клієнт-серверного вебзастосунку дозволить розділити складну бізнес-логіку на серверній стороні, а стороні клієнта лишити лише візуальну складову. Така структура дозволить обійти проблему ізольованості даних, оскільки всі зміни в розкладі будуть миттєво синхронізуватися з БД і будуть відображатися в режимі реального часу для всіх учасників навчального процесу.

Використання сучасних вебтехнологій дозволяє зробити систему адаптивною і забезпечити відображення розкладу на пристроях з різною діагоналлю екрана коректно.

Висновки до розділу 1

У першому розділі кваліфікаційної бакалаврської роботи проведено аналіз предметної області, пов’язаної з автоматизацією планування розкладу навчальних занять. Проведено аналіз існуючих аналогів, таких як aSc TimeTables, FET та Microsoft Excel і визначено поточний стан та тенденції розвитку доступних рішень з цієї предметної області. Результати показали, що комерційні та Open Source рішення мають серйозне алгоритмічне наповнення, але з недоліками у вигляді високих ліцензійних витрат або застарілими інтерфейсами та високим порогом входу. Традиційні ТБ, своєю чергою, не забезпечують необхідного рівня автоматизації та контролю колізій.

Розкрито структурні та функціональні особливості об’єкта дослідження. Класифіковано обмеження на жорсткі (Hard Constraints) та м’які (Soft Constraints).

У результаті проведеного аналізу сформульовано функціональні особливості вебсайту, що розробляється та обгрунтовано вибір клієнт-серверної архітектури, яка дозволить ефективно розділити бізнес-логіку та інтерфейс користувача.

2 МОДЕЛЮВАННЯ ОБ'ЄКТУ ТА ПРЕДМЕТУ ДОСЛІДЖЕННЯ

2.1 Обґрунтування моделей та методів вирішення задачі

Використання алгоритмічного підходу до вирішення проблеми генерації розкладу або ж Educational Timetabling Problem (ETP) є ключовою функціональною особливістю системи. Вичерпний перебір усіх можливих комбінацій для розкладу є обчислювально неможливим, оскільки ETP за всіма критеріями належить до класу NP-повних задач оптимізації [8]. Через велику кількість обмежень і величезний простір пошуку рішень ця проблема вимагає застосування ефективних обчислювальних методів та адекватних математичних моделей.

Аналіз сучасних наукових публікацій підтверджує, що для автоматизації генерації розкладу доцільно поєднувати строгу математичну формалізацію задачі із сучасними наближеними методами пошуку. Найбільш ефективним методом до моделювання комп'ютерних систем такого типу є задача задоволення обмежень (Constraint Satisfaction Problem – CSP) [9]. Водночас для безпосереднього пошуку рішень у надвеликих просторах станів усе частіше залучають метаевристичні методи (meta-heuristics) – ітераційні процеси, що забезпечують оптимальний баланс між дослідженням простору пошуку (exploration) та експлуатацією знайдених якісних рішень (exploitation) [10]. Використання моделі CSP дозволяє досягти необхідного комбінаторного звуження задачі та чітко врахувати складні правила користувача. Модель CSP дозволяє розглядати процес складання розкладу не як хаотичний пошук, а як математичну модель, що складається з трьох основних компонентів:

– змінні (Variables), якими у випадку задачі складання розкладу виступають заняття, які потрібно призначити для певної навчальної групи;

– домени (Domains), які представляють у собі перелік всіх можливих значень для кожної змінної (всі доступні часові слоти, всі вільні аудиторії);

– обмеження (Constraints), якими є попередньо визначені в підрозділі 1.3 жорсткі (Hard) і м’які (Soft) обмеження за якими визначається чи є обране з домену значення допустимим для конкретної змінної.

Коротко кажучи, суть алгоритму CSP полягає у постійному звуженні простору пошуку. Кожного разу, коли алгоритм призначає заняття на час, цей час і аудиторія видаляються з доменів усіх інших пов'язаних змінних. Це дозволяє системі працювати лише з дійсними (активними) варіантами і мати нормальну швидкість виконання. Більше того, якщо під час процесу призначення виникає тупик і алгоритм не може знайти значення з доменів, яке відповідає обмеженням, застосовується метод повернення назад (backtracking). Метод повернення назад активується, якщо, наприклад, система потрапляє в ситуацію, коли немає доступної кімнати в потрібний час, алгоритм повертається назад і намагається змінити попереднє рішення, щоб знайти вихід з конфлікту.

Також, застосування CSP дозволить системі динамічно реагувати на дії користувача. Тобто, якщо адміністратор намагається вручну змінити розклад, алгоритм перераховує доступність доменів для інших сутностей і перевіряє цілісність розкладу та відсутність порушень обмежень.

Цей підхід дозволяє перетворити абстрактну задачу планування на чітку математичну модель. Використання CSP забезпечує, що система не просто генерує випадковий набір даних, а постійно задовольняє всі встановлені обмеження (жорсткі та м'які обмеження), забезпечуючи цілісність розкладу.

2.2 Огляд технологій

Оскільки проєктування автоматизованої системи планування розкладу навчальних занять потребує від обраного стеку високої продуктивності при виконанні ресурсозатратних алгоритмів, то при аналізі сучасних інструментів особливу увагу слід приділяти архітектурній гнучкості. Аналіз сучасних наукових публікацій у фахових виданнях свідчить, що для систем класу TSS (Timetable Scheduling Systems) найбільш доцільним є використання архітектури на основі

компонентно-орієнтованих технологій. Такий підхід забезпечує необхідну модульність системи, спрощує інтеграцію з базами даних та дозволяє реалізувати гнучкий інтерфейс користувача для управління великими масивами даних.

Відповідно до цих критеріїв знайдено дослідження архітектурних принципів існуючої системи «iZETA» [3]. Аналіз вказує на те, що вона має чітко виражену модульну структуру. Кожен підмодуль викликається головним модулем і виконує відносно незалежну функцію. Успадкувавши успішний досвід модульного розділення функцій, але враховуючи необхідність переходу до сучасних компонентно-орієнтованих вебтехнологій, для реалізації власного вебзастосунку було обрано стек технологій PERN, який дозволяє побудувати ефективну full-stack архітектуру:

– PostgreSQL;
– Express.js;
– React;
– Node.js.

Серверна частина (Backend)

Node.js [11] – це сучасне середовище виконання JavaScript, яке побудоване на рушії V8 для виконання високопродуктивних мережевих застосунків. Раніше JavaScript застосовувався лише для обробки даних користувача в браузері, але Node.js надає можливість виконувати JavaScript-скрипти на серверній частині та відправляти користувачеві результат їхнього виконання. Це робить дане середовище ідеальним для розробки масштабованих мережевих сервісів, що потребують високої швидкості обробки запитів.

Також, основною перевагою Node.js є використання подієво-орієнтованої моделі неблокуючого вводу-виводу (Event-driven non-blocking I/O), що дозволяє обробляти тисячі одночасних запитів без значних витрат системних ресурсів. Оскільки в проєкті наявна ресурсомістка частина генерації розкладу, Node.js дозволить виконувати перевірку колізій та запуск алгоритмів оптимізації у фоні, а не блокувати роботу іншим користувачам.

Таблиця 2.1 – Порівняння традиційних серверних платформ та Node.js

Характеристика	**Традиційні платформи (Apache/PHP)**	**Node.js**
Модель обробки	Багатопотокова (один потік на запит)	Однопотокова з циклом подій (Event Loop)
Ввід-вивід (I/O)	Блокуючий	Неблокуючий
Продуктивність при I/O	Високі витрати на перемикання контексту	Висока ефективність при великій кількості з’єднань

Express.js [12] – це бекенд-фреймфорк для вебзастосунків на основі Node.js, який виступає стандартним шаром для управління маршрутами та обробки HTTP-запитів. Використання Express.js передбачає наступні можливості:

– швидке створення ендпоінтів та спрощення розробки REST API;

– використання Middleware, яке виступає проміжним обробником для перевірки прав доступу користувачів, валідації вхідних даних;

– легка інтеграція з БД для виконання CRUD операцій (Create, Read, Update, Delete).

Використання Express.js дозволяє побудувати надійний міст між базою даних та фронтенд-частиною, забезпечуючи стабільну передачу даних навіть при великій кількості запитів.

База даних

Для управління базами даних обрано PostgreSQL [13] – реляційну систему управління базами даних (СУБД), яка забезпечує надійність, архітектурну гнучкість та суворе дотриманням стандартів SQL. У контексті розробки системи планування розкладу дані мають складну структуру та велику кількість перехресних зв'язків, тому PostgreSQL є оптимальним вибором завдяки підтримці складних запитів та механізмів забезпечення цілісності даних.

PostgreSQL використовується для збереження даних:

– даних користувачів (логіни, хешовані паролі, ролі);

– реєстри викладачів, навчальних дисциплін, аудиторій та часових слотів;

– сформовані розклади занять із підтримкою версіонування;

– логічні обмеження для генерації (графіки доступності викладачів, місткості аудиторій та інше).

Підключення Node.js до PostgreSQL здійснюється за допомогою Sequelize [14] – сучасної ORM-бібліотеки (Object-Relational Mapping) для Node.js. Використання Sequelize дозволяє взаємодіяти з БД на високому рівні абстракції, оперуючи об'єктами JavaScript замість написання сирих SQL-запитів. Сутності розкладу в свою чергу, мають велику кількість взаємозв'язків типу «один-до-багатьох» та «багато-до-багатьох», а Sequelize дозволяє декларативно описувати ці зв'язки та автоматично виконувати операції об'єднання таблиць (JOIN).

Клієнтська частина (фронтенд)

Інтерфейс застосунку розробляється за допомогою React [15] – безкоштовна і відкрита фронтенд-бібліотека JavaScript для створення інтерфейсів користувача. Бібліотека базується на компонентному підході, тому дозволяє розбивати складний інтерфейс на окремі ізольовані частини, які можна перевикористовувати.

Основні технічні переваги React:

– компонентний підхід, який спрощує керування окремими елементами інтерфейсу;

– швидка робота завдяки використанню віртуального DOM;

– можливість легко поєднувати API з іншими бібліотеками та фреймворками.

Для розробки адаптивного, привабливого інтерфейсу використовується TailwindCSS [16] – сучасний utility-first CSS-фреймворк. На відміну від класичних фреймворків (наприклад, Bootstrap), де надаються вже готові компоненти, Tailwind працює на рівні атомарних (утилітарних) класів, тому стилізація

елементів виконується безпосередньо в HTML/JSX, комбінуючи дрібні класи-цеглинки.

Таблиця 2.2 — Порівняння React з традиційними методами побудови інтерфейсів

Характеристика	**React (SPA підхід)**	**Традиційний Multi-Page (PHP/HTML)**
Швидкість відгуку	Обробка відбувається на стороні клієнта, тому відгук завжди миттєвий	Залежить від швидкості відповіді сервера
Користувацький досвід	Плавний, оскільки відсутня необхідність перезавантаження сторінок	Перезавантаження сторінки при кожній дії
Валідація колізій	У реальному часі в браузері	Переважно після відправки форми на сервер

Безпека та захист даних

Забезпечення конфіденційності та цілісності інформації є важливим аспектом розробки системи. Для безпечної автентифікації та авторизації користувачів обрано JWT (JSON Web Token) [17] – відкритий стандарт для створення токенів доступу. Після успішного входу користувача сервер генерує підписаний токен, який зберігається на стороні клієнта. Кожен наступний запит до API Express.js супроводжується цим токеном, що і дозволяє серверу ідентифікувати користувача без необхідності повторного введення логіна та пароля. Використання JWT забезпечує високу масштабованість системи, оскільки серверу не потрібно зберігати сесії в пам'яті, що особливо важливо для стабільної роботи архітектури розкладу при одночасному доступі багатьох користувачів.

Також, у системі PostgreSQL паролі користувачів ніколи не зберігаються у відкритому вигляді, тому використовується бібліотека для адаптивного хешування паролів Bcrypt [18]. Bcrypt використовує алгоритм хешування разом із

механізмом «солі» (salt) – додаванням випадкових даних до пароля перед його обробкою. Це робить систему стійкою до атак.

2.3 Специфікація вимог до програмного забезпечення

1. Призначення та межі проєкту

1.1 Призначення системи

Вебзастосунок призначений для автоматизації процесу планування розкладу навчальних занять в закладах освіти та забезпечує швидку генерацію розкладу з уникненням конфліктів, а також надає зручний інтерфейс з можливістю перегляду та редагування розкладу.

1.2 Узгодження

– проєкт відповідає вимогам поставлених у дипломній роботі та сучасних практиках розробки вебзастосунків;

– узгоджено використання алгоритму CSP (Constraint Satisfaction Problem) для автоматизації процесу планування розкладу навчальних занять;

– визначено необхідність забезпечення цілісності даних за допомогою реляційної БД.

1.3 Межі проєкту

Проєкт не включає розробку мобільного чи настільного застосунків, але передбачає адаптивну версію сайту для різних типів пристроїв. Визначено необхідність розробки API серверної частини та клієнтського інтерфейсу.

2. Загальний опис

2.1 Сфера застосування

Розроблюване ПЗ охоплює галузі, де постає задача впорядкування ресурсів (часу, аудиторій, людських ресурсів), спрямована на оптимізацію використання часу, персоналу (викладачів), аудиторій та навчальних планів. Розрахована на використання адміністраторами (методистами) навчальних закладів, викладачами та студентами для оперативного доступу до розкладу навчального процесу.

2.2 Характеристики користувачів

– адміністратори: здійснюють управління користувачами та налаштуваннями безпеки;

– методисти: відповідальні за введення даних про викладачів, аудиторії та групи, здійснюють запуск алгоритму генерації розкладу;

– викладач\студент: користувачі з правом доступу лише на перегляд розкладу.

2.3 Загальна структура і склад системи

Структурно система поділяється на такі основні компоненти:

– Frontend: React-застосунок;

– Backend: Node.js сервер на базі Express;

– база даних**:** PostgreSQL для структурованого і безпечного зберігання даних.

2.4 Загальні обмеження

Застосунок потребує наявності доступу до інтернету.

3. Функції системи

3.1 Авторизація та аутентифікація

3.1.1 Опис функції

Функція забезпечує безпечний доступ до системи за допомогою перевірки даних авторизації користувача. Введений пароль порівнюється з хешем наявним у базі даних після реєстрації користувача. Після успішного входу, система генерує сесійний токен доступу.

3.1.2 Вхідна і вихідна інформація

– вхідна: email та пароль користувача;

– вихідна: підписаний JWT токен доступу, дані про рівень доступу користувача в системі (його роль).

3.1.3 Функціональні вимоги

– система повинна хешувати паролі за допомогою алгоритму Bcrypt із використанням механізму «солі»;

– серверна частина повинна генерувати JWT токен для виконання подальших запитів до системи без необхідності повторної авторизації та аутентифікації;

– система повинна обмежувати доступ до закритих маршрутів системи для неавторизованих користувачів.

3.2 Управління довідниками

3.2.1 Опис функції

Функція дозволяє методистам наповнювати БД інформацією про об'єкти навчального процесу (аудиторії, викладачі, групи, дисципліни), які використовуються як домени та змінні в задачі CSP.

3.2.2 Вхідна і вихідна інформація

– вхідна: назви та характеристики аудиторій, викладачів, груп та дисциплін;

– вихідна: структуровані записи в БД, які зможуть використовуватися алгоритмом генерації розкладу.

3.2.3 Функціональні вимоги

– система повинна забезпечувати CRUD операції для всіх об'єктів навчального процесу;

– перевірка на унікальність записів;

– для врахування в жорстких обмеженнях об'єкти мають мати специфічні характеристики (місткість аудиторії, тип аудиторії, спеціальність викладача та інше).

3.3 Автоматичне планування розкладу

3.3.1 Опис функції

Основна функція системи, що реалізує алгоритм задоволення обмежень (CSP) для автоматичного планування розкладу навчальних занять за часовими слотами, аудиторіями і з уникненням конфліктів.

3.3.2 Вхідна і вихідна інформація

– вхідна: набір змінних (заняття з дисциплін), домени (часові слоти та аудиторії) та жорсткі і м’які обмеження;

– вихідна: згенерований розклад навчальних занять, де кожне заняття має унікальні комбінації часу та аудиторії.

3.3.3 Функціональні вимоги

– функція повинна гарантувати відсутність конфліктів: один викладач або ж група не можуть бути в один час в різних місцях;

– модуль повинен враховувати м'які обмеження для покращення якості створеного розкладу.

4. Вимоги до інформаційного забезпечення

4.1 Джерела і зміст вхідної інформації

– дані авторизації користувачів: логіни, пароль, ПІБ студентів та викладачів;

– дані про аудиторії, дисципліни та групи з викладачами, які вводяться методистами.

4.2 Нормативно-довідкова інформація

Довідники часових інтервалів (номери пар та час їх початку і завершення), типів занять (лекція, практика, семінар, залік, екзамен, інше).

4.3 Організація зберігання даних

– збереження даних відбувається за допомогою реляційної СУБД PostgreSQL;

– паролі користувачів зберігаються виключно у вигляді хеш-кодів;

5. Вимоги до технічного забезпечення

– підтримка протоколу HTTPS;

– середовище виконання Node.js 18.0+ і PostgreSQL 13.0+.

6. Вимоги до програмного забезпечення

6.1 Архітектура програмної системи

– клієнт-серверна модель;

– RESTful API для обміну даними у форматі JSON.

6.2 Системне програмне забезпечення

Рекомендовано ОС Linux (Ubuntu) на сервері, Node.js (версії 18+).

6.3 Мережне програмне забезпечення

– протокол HTTPS із сертифікатом SSL/TLS для шифрування каналу зв’язку;

– механізм JSON Web Token (JWT) для ідентифікації сесій користувачів.

6.4 Програмне забезпечення ведення інформаційної бази

Реляційна система PostgreSQL та ORM Sequelize для взаємодії програмного коду з базою даних та управління міграціями.

6.5 Мова і технологія розробки

– Frontend: JavaScript (бібліотека React), Tailwind CSS для стилізації;

– Backend: JavaScript (фреймворк Express.js).

7. Вимоги до зовнішніх інтерфейсів

7.1 Інтерфейс користувача

Адаптивний вебінтерфейс, коректно відображуваний на різних пристроях, з роздільною здатністю від 320x480 до 1920x1080 пікселів.

7.2 Апаратний інтерфейс

Спеціалізоване обладнання не потрібне; доступ здійснюється через стандартні периферійні пристрої введення/виведення**.**

7.3 Програмний інтерфейс

REST API для забезпечення можливості подальшої інтеграції з іншим ПЗ.

7.4 Комунікаційний протокол

HTTPS для всіх клієнт-серверних запитів.

8. Властивості програмного забезпечення

8.1 Доступність

Система має бути доступною 99% часу на рік. Допустимий час технічного обслуговування – поза межами робочих годин закладу.

8.2 Супроводжуваність

Використання модульної архітектури та використання npm-пакетів забезпечує легкість оновлення та розширення функціонала системи.

8.3 Переносимість

– можливість розгортання на локальних серверах із підтримкою Docker;

– використання будь-яких хмарних платформах (AWS, DigitalOcean, Azure).

8.4 Продуктивність

Час генерації розкладу для середнього обсягу даних не повинен перевищувати 60-120 секунд. Час відповіді інтерфейсу на запити перегляду – до 1 секунди.

8.5 Безпека

– аутентифікація користувачів через токен JWT;

– зхешування паролів за допомогою BCrypt.

9. Інші вимоги

Підтримка експорту сформованих документів у формати .xlsx та .pdf для друку.

Висновки до розділу 2

У другому розділі кваліфікаційної бакалаврської роботи сформовано технічний фундамент для подальшої розробки вебзастосунку автоматизованої системи планування розкладу навчальних занять. Доведено доцільність використання методів задоволення обмежень (CSP). Також, визначивши що для системи необхідно виконати умови високої продуктивності та масштабованості, обрано стек PERN (PostgreSQL, Express.js, React, Node.js). Визначено переваги подієво-орієнтованої моделі Node.js та обгрунтовано доцільність використання реляційної СУБД PostgreSQL.

Описано функціональні можливості системи: модулі авторизації, управління довідниками та автоматичного планування. Визначено технічні та програмні вимоги.

3 ПРОЄКТУВАННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ ПЛАНУВАННЯ РОЗКЛАДУ

3.1 Моделі даних

Моделювання вебсистеми, яка працюватиме на основі реляційної системи керування базами даних (СКБД) належить починати з формування структур даних – основних сутностей предметної області та зв'язків між ними. Для відображення структури цих програмних сутностей, використано UML-діаграму класів (розд. 3.4), а для моделювання саме структури бази даних використано – ER-діаграму (Entity-Relationship Diagram) [19].

Діаграма «сутність-зв'язок» (ERD) – це діаграма яка використовується для візуалізації логічної структури бази даних, показуючи об'єкти (сутності), які існують у системі, які вони мають характеристики (атрибути) та як вони взаємодіють між собою.

Сутність (Entity) уособлює концепцію реального світу, інформацію про яку необхідно зберігати в системі. Наприклад для розроблюваної вебсистеми такими сутностями будуть: навчальний заклад, аудиторія, викладач, навчальна дисципліна та інші. У реляційних БД кожна така сутність трансформується в окрему таблицю з атрибутами, які деталізують її. Такі атрибути можуть бути представлені базовими властивостями про сутність, наприклад для «Аудиторія» це будуть номер, місткість, тип (комп’ютерний клас, лекційна аудиторія чи інше). Також в ці атрубити входять ключі: первинний ключ (Primary Key), який призначений для унікального ідентифікування кожного запису таблиці та зовнішній ключ (Foreign Key), що використовується для зв’язку з іншими таблицями та для забезпечення реляційної цілісності.

Для моделювання логічної залежності однієї сутності від іншої використовується – зв'язок (Relationship). Зв’язок «Один-до-багатьох» (1:M) вказує на те, що один запис у першій таблиці може бути пов'язаний з багатьма записами в другій. В розроблюваній системі такий зв'язок існує між «Навчальний

заклад» та його сутностями «Аудиторія», «Викладач», «Навчальна група», оскільки один об'єкт навчального закладу містить в собі багато таких об'єктів, але кожен конкретний викладач, група чи аудиторія можуть належати лише одному певному закладу.

Зв’язок «Багато-до-багатьох» (M:N) використовується коли безліч екземплярів однієї сутності можуть взаємодіяти з безліччю екземплярів іншої. У спроєктованій системі такий зв'язок є між сутностями «Викладач» та «Предмет», оскільки один викладач може викладати декілька дисциплін, і водночас одну й ту саму дисципліну можуть вести різні викладачі. В реляційній моделі даних не можна на пряму створювати зв'язки «Багато-до-багатьох», тому цей зв'язок декомпонується на два залежні зв'язки типу «Один-до-багатьох» через сполучну таблицю «Навчальний план» в якій і відбувається співставлення викладача та предмета.

Зв’язок «Один-до-одного» (1:1) використовується для сутностей, які мають жорстку і однозначну відповідність. Такий зв'язок наявний між авторизованим користувачем системи (User) із роллю «Викладач» та його розширеною анкетою (Teacher). Це використано для відокремлення конфіденційних даних від загальної інформації про викладача.

Також, при моделюванні БД варто враховувати специфіки предметної області та розглядати необхідність впровадження додаткових системних стуностей. В рамках задачі автоматичної генерації розкладу навчальних занять користувачі мають необхідість у підтримці «версіонування» розкладу, тому варто додати сутність «Версія розкладу», яка зв'язком «Один-до-багатьох» відокремлюватиме довідникову інформацію від інших версій. Для забезпечення існування поняття «підгрупа» та оптимізації зв'язків між цими підгрупами в сутності «Навчальна група» (Group) реалізовано рекурсивний зв'язок типу 1:M замість відокремлення цієї стуності в окрему таблицю БД.

Особливу увагу варто приділити взаємодії користувачів із платформою на етапі реєстрації. Оскільки автоматична генерація розкладу досить

ресурсозатратна обчислювальна задача, на початкових етапах розгортання системи варто зробити контрольовану реєстрацію – з ручним затвердженням адміністраторами. Для цього доцільно додати сутність для обробки запитів на реєстрацію: «Заявка на реєстрацію закладу» (InstitutionRequest). Дана сутність є ізольованим буфером і дозволяє накопичувати вхідні дані від потенційних клієнтів, не створюючи при цьому передчасних записів у основних таблицях Institutions та Users.

Змодельована діаграма «сутність-зв'язок» (додаток А, рис. А1) з усіма сутностями та їх атрибутами, первинними і зовнішніми ключами та зв’язками детально відображає реляційну структуру застосунку і дозволяє значно полегшити подальшу генерацію таблиць у БД PostgreSQL.

3.2 Діаграма прецедентів

Для ефективного проєктування розроблюваної автоматизованої системи планування розкладу навчальних занять доцільно скористатись уніфікованою мовою моделювання (UML, Unified Modeling Language) [20] для створення діаграми прецендентів (Use Case Diagram). Дана діаграма використовується для графічного представлення взаємодії між користувачами системи та її функціями. Для простішого розуміння прийнято вважати що діаграма прецендентів показує на те, хто взаємодіє з системою і що саме ця система повинна робити.

Основними елементами які зображуються на діаграмі використання актори (Actors), прецденти (Use Case), межі системи (System Boundary).

Актори – це користувачі системи, прилади або ж інші системи які взаємодіють з програмою. На основі специфікації вимог до програмного забезпечення (розд. 2) можна визначити ролі, які будуть використовуватись у розроблювані системі і обмежуються в даний момент лише користувачами:

– адміністратор системи;
– диспетчер розкладу (методист);
– викладач;

– студент або ж гість.

Для досягнення мети визначених акторів зазначаються окремі функції, які виступають прецедентами на діаграмі і пов’язуються з ними за допомогою асоціацій (Association). Згідно до загальноприйнятого стандарту UML-діаграм «ISO/IEC 19505» [21] використовуються лише чотири стандартні типи зв’язків: асоціація (Association), включення (Include), розширення (Extend) та узагальнення (Generalization).

Для створення діаграми прецендентів (рис. 3.1) використано CASE-інструмент (Computer-Aided Software Engineering) інструмент Software Ideas Modeler [22], оскільки він повністю відповідає раніше згаданому стандарту UML 2.5.

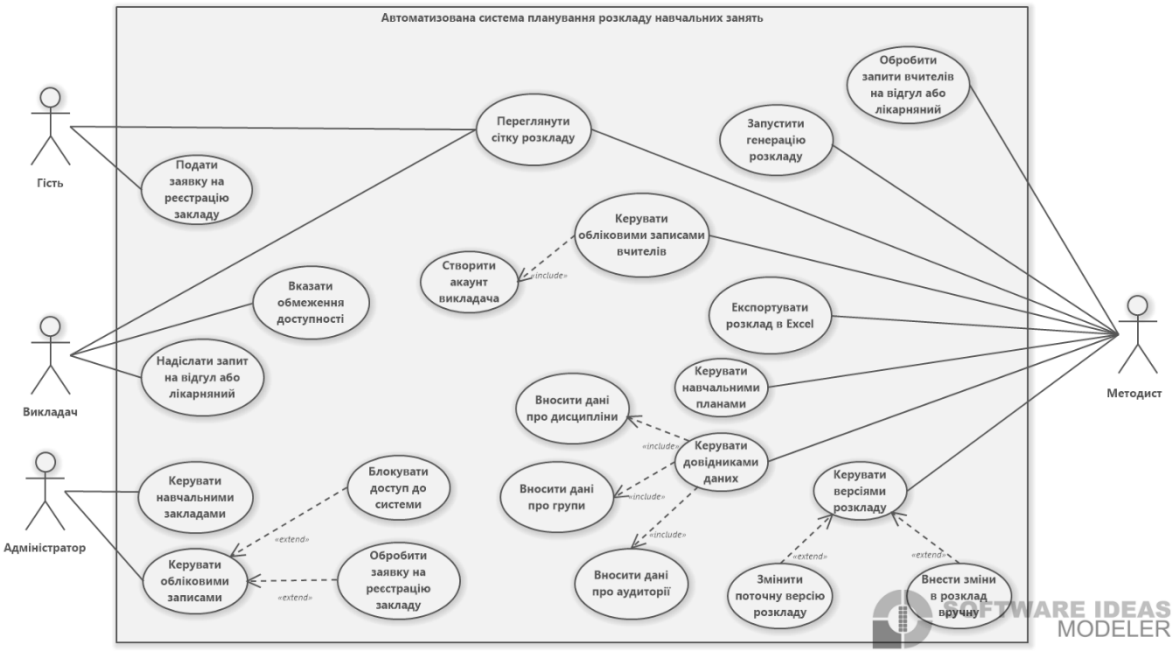

Рисунок 3.1 – Діаграма прецедентів

В розроблюваній системі згідно до діаграми прецедентів можна визначити майбутні можливості акторів. Гість виступає базовим рівнем доступу до системи і в більшості випадків таку роль матимуть студенти навчального закладу, яким необхідно лише переглянути актуальний розклад. У випадках коли майбутній методист ще не має повноцінного доступу до системи він може подати заявку на реєстрацію закладу будучи при цьому гостем.

Актор викладач вже є авторизованим користувачем чий розклад безпосередньо керується системою. Для цього рівня користувача відокремлюються наступні унікальні прецеденти:

– вказати обмеження доступності (незручні для роботи дні, побажання щодо навантаження протягом дня та інше);

– надіслати запит на відгул або лікарняний.

Ключовою роллю системи, яка має найбільше функціональне навантаження є методист. Даний користувач володіє повним спектром інструментів необхідних для організації навчального процесу в навчальному закладі. Серед прецендентів слід виділити можливість керувати довідковими даними та навчальними планами, які є основою необхідною для генерації розкладу.

Адміністратор в свою чергу забезпечує глобальну працездатність платформи і зосереджений на керуванні іншими користувачами. Прецендент «Керувати обліковими записами» має точки розширення (extend) – зв’язки які вказують на опціональну функціональність. Таким чином адміністратор має можливість за певних причин блокувати доступ до системи у інших користувачів та можливість схвалювати запити на додавання нових аккаунтів до системи надісланими неавторизованими гостями.

3.2 Сценарії використання

Тим часом як діаграма прецендентів візуально відображає сферу застосування рішення, показуючи акторів та варіанти використання з якими вони взаємодіють, для більш детального опису прецендентів прийнято формувати сценарії використаня (Use Case). Дані сценарії описують варіант можливої взаємодії користувача і системи під час використання певної функції, що і дозволяє змоделювати поведінку системи з точки зору кінцевого користувача.

Якщо посилатися на загальноприйняту методологію Алістера Коберна (Alistair Cockburn), описану в його книзі «Writing Effective Use Cases» [23], то формування для розроблюваної системи сценаріїв використання можна виконати

за повним форматом (Fully Dressed Format). Цей шаблон передбачає передбачає представлення сценаріїв використання у вигляді таблиці з наступним полями: Use Case Name (назва прецеденту), Scope (рамки), Level (Рівень), Primary Actor (основний актор), Stakeholders and interests (зацікавлені особи та їхні інтереси), Preconditions (передумови), Success guarantee (гарантія успіху), Main Success Scenario (основний сценарій успіху), Extensions (розширення), Special Requirements (спеціальні вимоги), Technology and Data Variations List (список технологічних та даних варіацій), Frequency of Occurrence (частота повторення). Для повноти розуміння поведінки системи достатньо представити сценарії використання для її основних функцій.

Первинною функцією будь-якої комп’ютерної системи є реєстрація нового користувача, тому слід перший сценарій зазначити саме для неї (табл 3.1). Описано основні кроки реєстрації методиста з детальним основним сценарієм успіху.

Таблиця 3.1 – Сценарій використання «Реєстрація методиста»

Usecase section	**Comment**
Use Case Name	Реєстрація методиста
Scope	System
Level	User Goal
Primary Actor	Гість
Stakeholders and interests	1) Гість – зацікавлений в отриманні доступу до системи для формування розкладу навчальних занять свого закладу. 2) Адміністратор – зацікавлений у валідації аккаунтів системи.
Preconditions	Користувач не був раніше зареєстрований в системі за вказаними даними.
Success guarantee	Створено та прив'язано обліковий запис методиста до згенерованого унікального ідентифікатора його закладу.
Main Success Scenario	1) користувач переходить на сторінку відправлення запиту на доступ;

Кінець таблиці 3.1

	2) система відображає форму для введення даних (назва закладу, ПІБ методиста, email та супровідний коментар); 3) користувач заповнює форму; 4) система перевіряє правильність введених даних та відправляє запит; 5) система виводить повідомлення про успішне відправлення та необхідність в очікуванні на розгляд заявки; 6) адміністратор переходить на панель керування і переглядає отримані дані; 7) система створює новий домен закладу та методиста; 8) система видаляє оброблену заявку та відправляє підтвердження реєстрації з тимчасовим паролем на вказаний email користувача.
Extensions	– користувач вказує існуючий email і система виводить повідомлення «Користувач з таким Email вже зареєстрований у системі»; – адміністратор вирішує не погоджувати вказану заявку.
Special Requirements	Створений заклад повинен мати унікальний ідентифікатор, який забезпечить ізоляцію даних.
Technology and Data Variations List	Тимчасовий пароль може бути згенерованим або ж введеним вручну адміністратором.
Frequency of Occurrence	5%

Перш ніж запускати генерацію розкладу, система повинна мати чітко сформовану реляційну базу обмежень з викладачами, предметами, навчальними групами та аудиторіями, тому важливою функцією методиста є можливість керування довідниковими даними (табл. 3.2).

Таблиця 3.2 – Сценарій використання «Керування довідниковими даними»

Usecase section	**Comment**
Use Case Name	Керування довідниковими даними
Scope	System
Level	User Goal

Кінець таблиці 3.2

Primary Actor	Методист
Stakeholders and interests	Методист – зацікавлений у структурованому введенні даних про викладачів, предмети, навчальні групи та аудиторії.
Preconditions	Користувач знаходиться на сторінкі введення довідникових даних.
Success guarantee	Довідникові сутності успішно додані, модифіковані або вилучені в базі даних.
Main Success Scenario	1) методист обирає довідник, яким бажає керувати та переходить на його сторінку; 2) система відображає обраний довідник у вигляді інтерактивної таблиці; 3) методист ініціює створення, редагування або видалення запису натиснувши відповідну кнопку керування; 4) система відкриває діалогове вікно з формою введення\редагування параметрів або підтвердження видалення; 5) методист підтверджує операцію натиснувши відповідну кнопку; 6) система виконує транзакцію з БД; 7) діалогове вікно закривається і відображається таблиця з оновленими даними.
Extensions	Модифікація або створення даних заблоковано через порушення унікальних полів.
Special Requirements	Система швидко обробляє CRUD запит.
Technology and Data Variations List	Оновлення таблиць без перезавантаження сторінки.
Frequency of Occurrence	40%

Якщо методист успішно вніс всі довідникові дані, він може перейти до центральної функції системи – запуску генерації розкладу (табл. 3.3). Зазначений сценарій використання описує взаємодію методиста з інтелектуальним ядром системи.

Таблиця 3.3 – Сценарій використання «Запуск генерації розкладу»

Usecase section	**Comment**
Use Case Name	Запуск генерації розкладу
Scope	System
Level	User Goal
Primary Actor	Методист
Stakeholders and interests	1) Методист – зацікавлений в отриманні згенерованої сітки занять без колізій, накладок між аудиторіями, групами та з урахуванням побажань викладачів. Викладач – зацікавлений у врахуванні його побажань під час генерації.
Preconditions	Методист успішно вніс усі довідникові дані та сформував навчальний план.
Success guarantee	Алгоритм успішно врахував матрицю обмежень і сформував розклад.
Main Success Scenario	1) методист переходить на фінальну сторінку майстра налаштувань; 2) методист запускає генерацію розкладу натисканням на відповідну кнопку; 3) система блокує інтерфейс методиста та відображає вікно логів, відправляє запит на запуск генерації до бекенду; 4) інтелектуальне ядро зчитує з БД матрицю жорстких та м'яких обмежень, запускає алгоритм пошуку з бектрекінгом; 5) система успішно знаходить комбінацію занять без конфліктів інтересів та з урахуванням обмежень; 6) система створює нову версію розкладу та зберігає її в БД; 7) система знімає блокування інтерфейсу та виводить повідомлення про успішну генерацію; система активує кнопку переходу до режиму перегляду створеного розкладу.
Extensions	Інтелектуальному ядру не вдалося знайти рішення через суперечливі обмеження.

Кінець таблиці 3.3

Special Requirements	Обчислюваннння на сервері повинні бути асинхронні та відбуватися в фоні.
Technology and Data Variations List	Базовий алгоритм може поєднуватися з евристиками MRV (Minimum Remaining Values) для прискорення знаходження результату.
Frequency of Occurrence	5%

Завершальним прецендентом і найбільш часто використовуваним є «Перегляд сітки розкладу» (табл. 3.4). На архітектурному рівні системи прийнято рішення, що студенти є неавторизованими користувачами, тому доступ до сітки розкладу вони отримують за публічним посиланням, яке надає методист закладу.

Таблиця 3.4 – Сценарій використання «Перегляд сітки розкладу»

Usecase section	**Comment**
Use Case Name	Перегляд сітки розкладу
Scope	System
Level	User Goal
Primary Actor	Гість (студент)
Stakeholders and interests	1) Студент – зацікавлений у швидкому та зручному перегляді актуального розкладу. 2) Методист – зацікавлений у легкому розповсюдженні розкладу через одне посилання.
Preconditions	Методист успішно згенерував версію розкладу та зазначив її як активну. Гість отримав публічне посилання на розклад закладу.
Success guarantee	Система успішно зчитала запит та візуалізувала дані у зручну сітку розкладу.
Main Success Scenario	1) студент переходить за наданим публічним посиланням; 2) система за отриманими параметрами пошуку знаходить необхідні дані в БД; 3) система візуалізує отримані дані в сітку розкладу та рендерить її на екрані користувача.
Extensions	Публічне посилання застаріле або розклад ще не опубліковано.

Кінець таблиці 3.4

Special Requirements	Сторінка показу сітки розкладу має бути адаптивною для коректного відображення на мобільних пристроях.
Technology and Data Variations List	Можливість фільтрації розкладу для певної групи або викладача. Доступ до експорту даних в ТБ Excel.
Frequency of Occurrence	90%

Зазначені сценарії використання відображають основні функціональні можливості системи та дозволяють спроєктувати типові дії користувачів ще до технічної реалізації застосунку. Це дозволяє покращити якість майбутньої розробки через врахування обмежень та вимог ще на етапі моделювання.

3.3 Діаграма класів

Для моделювання структури об'єктно-орієнтованих систем в нотації UML використовується діаграма класів (Class diagram). Дана діграма зображує структуру системи за допомогою моделювання її класів, властивостей, операцій і зв’язків між об’єктами. Згідно до стандарту UML 2.5 кожен клас відображується у вигляді прямокутника і включає назву класу, атрибути та операції. Кожен метод чи атрибут позначається рівнем доступу до нього в коді, а зв’язки між класами позначаються визначеними типами зв’язків – асоціація, успадкування, агрегація та композиція:

– асоціація показує що об'єкти одного класу пов'язані з об'єктами іншого класу;

– успадкування вказує на батьківський клас;

– агрегація вказує на просту належність одного класу до іншого;

– композиція вказує на жорстку залежність одного класу від іншого, коли життєвий цикл залежного класу повністю керується головним.

Такий підхід до моделювання дозволяє ще до реалізації отримати узагальнений вид системи, полегшити розробку.

Оскільки архітектура системи є модульною і відповідає патерну MVC (Model-View-Controller) [25], то доцільно розділити проєкт на логічні шари та продемонструвати основні класи системи (додаток А, рис. А2).

Діаграма класів відображає основні шари системи:

– моделі даних відображають структуру класів, які формують таблиці в БД для збереження інформації про навчальний заклад. Всі інші сутності перелічено в коментарях до даного шару;

– шар безпеки та даних відображає ключові класи «middleware», які надають можливості логіну та контролю прав доступу;

– в ядрі архітектури розкладу зосереджені ключові класи, які відповідають за бізнес-логіку застосунку;

– шар адміністрування та введення даних відображає класи, які відповідають за наповнення системи інформацією та керування навчальними закладами (SuperAdminController).

3.4 Діаграми діяльності

Діаграма класів використовується для представлення статичної структури системи, тобто показує з чого вона складається, а не як працює в динаміці. В UML для демонстрації покрокового процесу виконання певної задачі використовуються діаграми діяльності (Activity Diagram). За стандартом UML 2.5 діаграма діяльності складається з таких стандартних графічних фігур:

– початкова точка (Initial Node), яка зображується суцільною чорною точкою і позначає момент з якого починається процес;

– дія (Action), яка позначається прямокутником і представляє конкретний крок алгоритму;

– розгалуження (Decision Node), яке виконує роль умови з розгалуженням на різні подальші кроки алгоритму;

– паралельність (Fork / Join), яка позначається чорною лінією і позначає процеси які відбуваються паралельно;

– кінцева точка (Activity Final Node), яка позначається чорним колом і є завершенням всього процесу.

Одним із основних і складних елементів системи є процес реєстрації нового навчального закладу (додаток А, рис. А3). Початкове розгортання системи для нових клієнтів згідно до сценарію використання «Реєстрація методиста» відбувається за контролю адміністратора. Бекенд виконує лише первинну перевірку даних та створює об'єкт запиту (InstitutionRequest). Після чого прийняття рішення повністю делегується на адміністратора платформи, який може присвоїти статус Rejected або ж погодити заявку і таким чином передати бекенду команду на запуск розгортання середовища для нового закладу.

Система у разі погодження запиту створює нову відокремлену сутність закладу (Institution) і першого користувача з прив’язкою до даного закладу із найвищими локальними правами доступу (Methodist).

В кінці статус транзакції змінюється на Approved і система відправляє підтвердження реєстрації на вказаний при заповненні заявки email.

Найбільш технічно складним в системі є процес «Автоматична генерація розкладу» (додаток А, рис. А4). Даний процес реалізує роботу CSP-двигуна (Constraint Satisfaction Problem) і оперує великим масивом вхідних даних про аудиторії, навчальні плани, обмеження викладачів, вихідних зі святами та навчальних груп. Збір даних для генерації відбувається асинхронно, система виконує одночасне паралельне зчитування реляційних таблиць БД і формує єдину матрицю. Перед запуском циклу підбору, всі навчальні години сортуються за принципом «найбільш обмеженої змінної». Таким чином генерація спершу розподілить домени, які мають найбільше обмежень, що значно зменшить можливість виникнення тупикової ситуації під час генерації. Після цього відбувається пошук із поверненням і алгоритм намагається знайти до кожного домена валідний слот. Якщо ж встановлені обмеження не мають математичного розв’язку і система через це більше чотирьох раз поспіль виконує повернення назад активується механізм адаптивного послаблення умов. relaxationBuffer

інкрементується на одиницю дозволяючи алгоритму ігнорувати частину умов. Так процес циклічно повторюється поки не буде знайдено компромісний варіант розкладу.

У разі успішного завершення пошуку з поверненням система ініціює запис результатів у БД, видаляє попередній запис якщо такий є для поточної версії. Якщо ж після п'яти ітерацій (коли relaxationBuffer перевищує значення 4) алгоритму так і не вдалося знайти розклад, то процес переривається зі статусом 422 (Execution timeout).

Висновки до розділу 3

У третьому розділі кваліфікаційної бакалаврської роботи здійснено моделювання автоматизованої системи планування розкладу навчальних занять. Для графічного представлення взаємодії між користувачами системи та її функціями створено діаграму прецедентів. В розроблюваній системі згідно до діаграми прецедентів визначено акторів та їх можливості.

Для більш детального опису прецедентів сформовано сценарії використання. Сценарії описують варіанти можливої взаємодії користувача і системи під час використання функцій реєстрації методиста, керування довідниковими даними, запуску генерації розкладу та перегляд сітки розкладу.

Моделювання структури системи виконано у формі діаграми класів. Відображено структуру системи за допомогою моделювання її класів, властивостей, операцій і зв’язків між об’єктами. Оскільки архітектура системи є модульною і відповідає патерну MVC (Model-View-Controller), то проєкт було розділено на логічні шари та продемонстровано основні класи системи.

Для демонстрації покрокового процесу виконання задач реєстрації нового навчального закладу та автоматичної генерації розкладу використано діаграми діяльності (Activity Diagram).

Створені діаграми та сценарії використання є основною для подальшої розробки застосунку і дозволяють зрозуміти як працює система ще до її реалізації.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ ПЛАНУВАННЯ РОЗКЛАДУ

4.1 Розробка системи

Серверна частина автоматизованої системи складання розкладу занять реалізована за допомогою середовища виконання Node.js на мові програмування JavaScript і з дотриманням архітектурного патерну MVC. Виконує роль «внутрішнього двигуна» всієї системи, відповідає за обробку та збереження даних, виконання бізнес-процесів. Архітектуру можна поділити на наступні шари:

– моделі, які реалізовані за допомогою ORM-бібліотеки Sequelize та декларативно описують таблиці в БД;

– контролери уособлюють шар бізнес-логіки системи та виступають мостом між моделями та мережевими запитами користувачів;

– сервіси містять в собі складну логіку яка не входить до стандартних CRUD-операцій контролерів;

– роутери, які визначають ендпоінти REST API додатка, пов'язуючи відповідні URL-адреси та HTTP-методи (GET, POST, PUT, DELETE) із методами контролерів.

Робота з БД

Для збереження даних системи використано реляційну СУБД PostgreSQL з використанням ORM-бібліотеки Sequelize. Розроблювана система передбачає одночасний доступ великої кількості користувачів, тому на рівні конфігурації Sequelize було реалізовано механізм пулу з'єднань (рис. 4.1).

Така конфігурація встановлює верхню межу активних з'єднань і таким чином постійно підтримує визначену кількість підключень до PostgreSQL та одночасно захищає СУБД від перевантаження.

Для ізоляції даних між навчальними закладами використано принцип багатоорендності (Multi-tenancy). Це дозволяє використовувати єдиний екземпляр системи та БД для багатьох клієнтів одночасно. Програмна реалізація такого

підходу виконана за допомогою впровадження зовнішнього ключа «InstitutionId» до кожної сутності (рис. 4.2).

```
const sequelize = new Sequelize(
  process.env.DB_NAME,
  process.env.DB_USER,
  process.env.DB_PASSWORD,
  {
    host: process.env.DB_HOST,
    port: process.env.DB_PORT || 5432,
    dialect: 'postgres',
    logging: false,
    // CONNECTION POOLING CONFIGURATION:
    pool: {
      max: 10,        // Maximum number of connection instances in pool
      min: 0,         // Minimum number of connection instances in pool
      acquire: 30000, // Maximum time (ms) that pool will try to get connection before throwing error
      idle: 10000     // Maximum time (ms) that a connection can be idle before being released
    }
  }
);
```

Рисунок 4.1 – Конфігурація пулу з’єднань

```
indexes: [
  // Multi-column unique index: ensures unique classroom numbers within the same institution/branch
  {
    unique: true,
    fields: ['number', 'InstitutionId']
  }
]
```

Рисунок 4.2 – Використання зовнішнього ключа «InstitutionId»

Зазначений код є частиною моделі «Classroom» і окрім зовнішнього ключа, дана модель індексується за полем «number», щоб створити складену перевірку на унікальність і дозволити в різних навчальних закладах мати однакові назви кабінетів. Даний підхід застосовано до всіх ключових сутностей системи.

Безпека та автентифікація

Оскільки система чітко поділена на ролі та містить в собі велику кількість конфіденційних даних закладів та їх викладачів, значну увагу було приділено підсистемі безпеки. Збереження паролів користувачів виконано із використанням Bcrypt, щоб захистити облікові записи у випадку витоку даних (рис. 4.3).

```
const hashedPassword = await bcrypt.hash(methodistPassword, 10);

await User.create({
  fullName: methodistName.trim(),
  email: methodistEmail.trim().toLowerCase(),
  password: hashedPassword,
  role: 'Methodist',
  InstitutionId: institution.id,
  isActive: true
});
```

Рисунок 4.3 – Хешування паролю з використанням Bcrypt

Для управління сесіями користувачів використано стандарт JWT. Метод логіну в контролері «authController» генерує зашифрований токен, підписаний секретним ключем сервера та записує в нього ідентифікатор користувача, його роль і ідентифікатор навчального закладу до якого він прив'язаний. З цими метаданими токен передається клієнту і зберігається в локальному сховищі браузера для подальшого його використання в HTTP-запитах.

```
// SESSION LIFECYCLE STRATEGY:
// 30 days expiry if "Remember Me" is checked, otherwise 2 hours
const tokenExpiry = rememberMe ? '30d' : '2h';

const token = jwt.sign(
  {
    id: user.id,
    role: user.role,
    InstitutionId: user.InstitutionId
  },
  process.env.JWT_SECRET,
  { expiresIn: tokenExpiry }
);

res.json({
  token,
  user: {
    id: user.id,
    fullName: user.fullName,
    role: user.role,
    InstitutionId: user.InstitutionId
  }
});
```

Рисунок 4.4 – Генерація JWT

Життєвий цикл наданого токену контролюється згідно умов які вказує користувач. Якщо під час логіну було обрано опцію «rememberMe», то токен лишається валідним протягом 30 днів, інакше лише 2 години.

Захист ендпоінтів розробленого REST API відбувається за допомогою двох обробників «authMiddleware» (рис. 4.5) та «roleMiddleware». Модуль перевірки авторизації перевіряє вкладений в HTTP-запит токен за допомогою «jwt.verify()» і дешифрує згенерований при логіні JWT. Якщо валідація відбувається успішно, то мідлвар додає до об’єкта запиту дешифровані метадані користувача і дозволяє пройти далі по ланцюжку виконання до бізнес-контролерів. У випадку, якщо термін дії сесії закінчився, модуль безпеки генерує помилку та повідомляє клієнта про необхідність повторної авторизації користувача.

```
const authMiddleware = (req, res, next) => {
  const authHeader = req.headers.authorization;

  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return res.status(401).json({ message: 'Access denied. No token provided.' });
  }

  const token = authHeader.split(' ')[1];

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    // Injects userId, role, and institutionId directly into the request object
    req.user = decoded;

    next();
  } catch (error) {
    console.error('JWT Verification Error:', error.message);

    if (error.name === 'TokenExpiredError') {
      return res.status(401).json({ message: 'Token has expired. Please log in again.' });
    }

    return res.status(401).json({ message: 'Invalid token.' });
  }
};
```

Рисунок 4.5 – Модуль перевірки авторизації

Модуль перевірки ролі в свою чергу отримує масив ролей, які декларативно вказуються під час опису маршрутів у файлах роутерів та зіставляє їх із роллю, яка була перед цим вбудована в об'єкт запиту після дешифрування JWT в ««authMiddleware»». Наприклад, додавання обробника ролі до роутера який відповідає за доступ до генерації розкладу і доступний лише для методистів відбувається таким чином:

router.use(authMiddleware, roleMiddleware(['Methodist']));

Сервіс автоматичної генерації

Контролери які виступають мостом між моделями та мережевими запитами користувачів реалізовують стандартні CRUD-методи, тому більш доцільно розглянути сервіси, які містять головну складну логіку застосунку. Центральною функцією всієї системи є автоматична генерація розкладу (generatorService). Згідно діаграми діяльності створеної для даного процесу в третьому розділі, реалізація містить в собі розгортання CSP-двигуна. Три обов’язкові компоненти якими оперує дана задача реалізовані програмно: змінні (Variables), домени (Domains), обмеження (Constraints).

Масив навчальних занять (flatTasks), сформований на основі планових годин сутності Curriculum представляє змінні, які алгоритм повинен призначити на якусь позицію в розкладі (рис. 4.6).

```
let flatTasks = [];
const studentTotalLoad = {}; // Accurate sum of hours a physical student actually sits in class

for (const task of curriculumTasks) {
  const types = [
    { type: 'Lecture', pairs: task.lectureHours || 0 },
    { type: 'Laboratory', pairs: task.labHours || 0 },
    { type: 'Practice', pairs: task.practiceHours || 0 }
  ];

  for (const t of types) {
    for (let i = 0; i < t.pairs; i++) {
      flatTasks.push({
        type: t.type,
        GroupId: task.GroupId,
        SubjectId: task.SubjectId,
        TeacherId: task.TeacherId
      });

      const parentId = groupMap[task.GroupId];
      if (parentId) {
        // If task assigned to subgroup, it affects subgroup students
        studentTotalLoad[task.GroupId] = (studentTotalLoad[task.GroupId] || 0) + 1;
      } else {
        // If task assigned to parent group, it implicitly affects ALL its child subgroups too
        studentTotalLoad[task.GroupId] = (studentTotalLoad[task.GroupId] || 0) + 1;
        allGroups.forEach(g => {
          if (g.parentGroupId === task.GroupId) {
            studentTotalLoad[g.id] = (studentTotalLoad[g.id] || 0) + 1;
          }
        });
      }
    }
  }
}
```

Рисунок 4.6 – Формування навчальних занять

У БД змінна типу навчальний план містить інформацію про те який предмет призначено для групи та який викладач і скільки лекцій, практик чи лабораторних має провести протягом тижня. Для того щоб алгоритм міг працювати з цими даними така змінна в циклі розбивається на конкретні картки, які алгоритм може рухати по розкладу. Наприклад, якщо групі призначено в навчальному плані 3 лекції на тиждень, то цей навчальний план в циклі розбивається на три окремі картки і саме з ними працює алгоритм.

Також, для врахування випадків коли група ділиться на підгрупи та правильного розрахунку навантаження на студентів, в цей блок коду включено алгоритм розумного підрахунку навантаження «studentTotalLoad». Алгоритм аналізує згенеровану на поточному кроку циклу картку і якщо пара призначена конкретній підгрупі «if (parentId)», то навантаження інкрементується виключно для цієї підгрупи і ніяк не впливає на батьківську групу. Такий підхід дозволяє ще до початку розподілу занять знати обсяг навантаження на кожну групу чи підгрупу та на основі цих даних розрахувати індивідуальну щоденна норму, яку можна використовувати як м’яке обмеження при генерації і не змушувати студентів вчитись на максимум в якісь дні і нічого не робити в інші.

Домени які представляють позиції на які алгортим може призначити змінні, представляються у вигляді декартового добутку робочих днів (activeDays), часових інтервалів (activeSlots) та сумісних за місткістю і типом аудиторій (classrooms).

```
const allDays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'];
const activeDays = allDays.slice(0, version.daysPerWeek || 5);
const activeSlots = timeSettings.map(s => String(s.orderNumber));

const classrooms = await Classroom.findAll({ where: { InstitutionId: institutionId, isAvailable: true } });
```

Рисунок 4.7 – Формування доменів

Проте, повний перебір всіх доменів під час призначення змінних є занадто довгим та ресурсомістким процесом та змушує тримати в оперативній пам'яті сервера величезний масив. Для оптимізації такого перебору система формує простори доменів динамічно, прямо під час перебору (рис. 4.8).

```
function solve(index, relaxationBuffer) {
  if (index === totalTasks) return true;

  const task = flatTasks[index];
  const suitableRooms = roomCache[task.type] || classrooms;

  const shuffledDays = shuffleArray(activeDays);
  const shuffledSlots = shuffleArray(activeSlots);

  for (const day of shuffledDays) {
    if (systemBlockedKeys.has(day)) continue;

    for (const slot of shuffledSlots) {
      const key = `${day}-${slot}`;

      if (task.TeacherId && busyTeachers[key]?.has(task.TeacherId)) continue;
      if (busyGroups[key]?.has(task.GroupId)) continue;
      if (task.TeacherId && forbiddenTeacherSlots.has(`${task.TeacherId}-${day}-${slot}`)) continue;

      for (const room of suitableRooms) {
        if (hasConflict(task, day, slot, room.id, relaxationBuffer)) continue;

        if (!busyTeachers[key]) busyTeachers[key] = new Set();
        if (!busyGroups[key]) busyGroups[key] = new Set();
        if (!busyClassrooms[key]) busyClassrooms[key] = new Set();
```

Рисунок 4.8 – Динамічне формування доменів під час перебору змінних розкладу (початок методу Solve)

Початкові «allDays» та «activeSlots» випадково тасуються і перетворюються на масиви «shuffledDays» та «shuffledSlots». Це є важливим евристичним рішенням, яке дозволяє уникати ситуацій стандартного хронологічного перебору. Якби алгоритм завжди починав перебір з «Понеділка, 1-ї пари», то він формував би розклад який заповнює доступні слоти на перші дні тижня і майже нічого не залишає на інші дні. Щоб уникнути таких випадків можна було б скористатись механізмом покрокового повернення і змушувати системи переставляти призначені

на перші дні тижня заняття, але це б значно збільшило час генерації, оскільки довелось би проходитись по величезній кількості зайвих циклів. Випадковий порядок доменів дозволяє рівномірно розподілити заняття по всій сітці і навіть створює ймовірність знайти компромісний варіант розкладу з перших спроби проходження по циклу.

Механізм динамічного декартового добутку працює «на льоту» всередині рекурсивної функції за трьома етапами. На першому етапі цикл обирає випадковий день тижня і умовою «systemBlockedKeys.has(day)» прибирає дні, які заблоковані для змінних розкладу (вихідні або свята). Наступний цикл заходить в середину обраного дня та обирає слот (пару). Якщо викладач чи група вже зайняті для цієї пари іншим предметом, або якщо в цей час викладач не працює (forbiddenTeacherSlots), алгоритм за допомогою команди «continue» переходить до наступного слоту. На третьому рівні цикл «for (const room of suitableRooms)» перебирає кабінети в зазделегіть підготовленому масиві «suitableRooms» (який містить лише сумісні за типом та місткістю приміщення). Це дозволяє суттєво зменшити час роботи алгоритму. Після цього викликається функція «hasConflict», яка перевіряє чи не порушує така позиція в розкладі встановлені обмеження (рис. 4.9).

```
function hasConflict(task, day, slot, roomId, relaxationBuffer) {
  const key = `${day}-${slot}`;

  if (task.TeacherId && busyTeachers[key]?.has(task.TeacherId)) return true;
  if (busyClassrooms[key]?.has(roomId)) return true;
  if (busyGroups[key]?.has(task.GroupId)) return true;

  const parentId = groupMap[task.GroupId];
  if (parentId && busyGroups[key]?.has(parentId)) return true;
  if (busyGroups[key]) {
    for (const bookedGroupId of busyGroups[key]) {
      if (groupMap[bookedGroupId] === task.GroupId) return true;
    }
  }

  const targetCap = (studentDailyCap[task.GroupId] || 4) + relaxationBuffer;
  const currentTargetLoad = groupDailyLoadTracker[`${task.GroupId}-${day}`] || 0;
  if (currentTargetLoad >= targetCap) return true;

  if (parentId) {
    const parentCap = (studentDailyCap[parentId] || 4) + relaxationBuffer;
    const currentParentLoad = groupDailyLoadTracker[`${parentId}-${day}`] || 0;
    if (currentParentLoad >= parentCap) return true;
  }

  return false;
}
```

Рисунок 4.9 – Перевірка обмежень встановлених для розкладу (функція hasConflict)

На першому рівні перевіряється дотримання жорстких обмежень. Перевіряється чи не було вже зайнято викладача, кабінету або групи через оперативні масиви «busyTeachers», «busyClassrooms» та «busyGroups». Також відбувається контроль підгруп за картою «groupMap», де знаходиться «батьківська» група для підгрупи та перевіряється чи не зайнята вона зараз на спільному занятті. Так само відбувається і зворотній цикл який перевіряє не зайнята зараз якась підгрупа.

На другому рівні відбувається контроль м'яких обмежень, який відповідає за балансування навантаження студентів протягом тижня. Функція зчитує поточну кількість пар групи на обраний день і порівнює з нормою. Якщо ліміт пар на день перевищено, то функція такий варіант розміщення слоту не дозволяє.

Якщо обидва рівні перевірок було пройдено успішно, то функція повертає «false», і дозволяє призначення слоту (рис. 4.10).

```
if (task.TeacherId) busyTeachers[key].add(task.TeacherId);
busyGroups[key].add(task.GroupId);
busyClassrooms[key].add(room.id);

groupDailyLoadTracker[`${task.GroupId}-${day}`] = (groupDailyLoadTracker[`${task.GroupId}-${day}`] || 0) + 1;
const parentId = groupMap[task.GroupId];
if (parentId) {
  groupDailyLoadTracker[`${parentId}-${day}`] = (groupDailyLoadTracker[`${parentId}-${day}`] || 0) + 1;
} else {
  allGroups.forEach(g => {
    if (g.parentGroupId === task.GroupId) {
      groupDailyLoadTracker[`${g.id}-${day}`] = (groupDailyLoadTracker[`${g.id}-${day}`] || 0) + 1;
    }
  });
}

const matchingSlotSetting = timeSettings.find(s => String(s.orderNumber) === slot);
const timeSlotLabel = matchingSlotSetting
  ? `${matchingSlotSetting.startTime.substring(0, 5)} - ${matchingSlotSetting.endTime.substring(0, 5)}`
  : `Пара #${slot}`;

finalScheduleProposals.push({
  dayOfWeek: day,
  timeSlot: timeSlotLabel,
  type: task.type,
  GroupId: task.GroupId,
  SubjectId: task.SubjectId,
  TeacherId: task.TeacherId || null,
  ClassroomId: room.id,
  ScheduleVersionId: versionId,
  isOnline: false
});

if (solve(index + 1, relaxationBuffer)) return true;
```

Рисунок 4.10 – Збереження параметрів заняття та рекурсивний перехід (продовження методу Solve)

Спочатку ідентифікатори викладача, групи та аудиторії додаються методом «add» до сховищ зайнятості на цей конкретний час, і таким чином блокується цей

слот для призначення його іншому предмету. Також, відбувається інкремент денного навантаження та підбирається текстова назва часу пари. Фінальний об'єкт додається в загальний масив «finalScheduleProposals» і алгоритм робить крок уперед до наступної картки заняття – «solve (index + 1, relaxationBuffer)».

Якщо ж функція «hasConflict» знаходить конфлікт, то активується покрокове повернення назад (Backtracking) (рис. 4.11).

```
if (task.TeacherId) busyTeachers[key].delete(task.TeacherId);
busyGroups[key].delete(task.GroupId);
busyClassrooms[key].delete(room.id);

groupDailyLoadTracker[`${task.GroupId}-${day}`]--;
if (parentId) {
  groupDailyLoadTracker[`${parentId}-${day}`]--;
} else {
  allGroups.forEach(g => {
    if (g.parentGroupId === task.GroupId) {
      groupDailyLoadTracker[`${g.id}-${day}`]--;
    }
  });
}
finalScheduleProposals.pop();
```

Рисунок 4.11 – Покрокове повернення назад (продовження методу Solve)

Покрокове повернення назад полягає у скасуванні останніх дій для повернення до попереднього стабільного стану складання розкладу. Якщо рекурсивний виклик функції solve повертає значення false, це сигналізує про те, що поточний варіант пошуку зайшов у безвихідь. Таким чином комбінація ресурсів на попередніх кроках визначається як невдала. Алгоритм виконує зворотню операцію: вилучає викладача, групу та кабінет із відповідних об'єктів оперативної пам'яті та повністю видаляє останній сформований слот з кінця масиву за допомогою функції «pop()». Після цього рекурсія намагається підібрати інший вільний кабінет або часовий слот для поточної картки-заняття.

Останнім, але не менш важливим елементом алгоритму є захист обчислювального двигуна від потрапляння у нескінченні цикли. Це означає, що конфігурація обмежень встановлена так, що містить логічні суперечності. Використано ітераційний цикл адаптивного послаблення умов, який керується «relaxationBuffer» (рис. 4.12).

```
let success = false;
// Dynamic incremental relaxation matrix path
for (let relaxationBuffer = 0; relaxationBuffer <= 4; relaxationBuffer++) {
  Object.keys(busyTeachers).forEach(k => busyTeachers[k].clear());
  Object.keys(busyGroups).forEach(k => busyGroups[k].clear());
  Object.keys(busyClassrooms).forEach(k => busyClassrooms[k].clear());
  Object.keys(groupDailyLoadTracker).forEach(k => delete groupDailyLoadTracker[k]);
  finalScheduleProposals = [];

  if (solve(0, relaxationBuffer)) {
    success = true;
    console.log(`[CSP Engine Success]: Grid compiled natively with relaxation buffer: +${relaxationBuffer} pairs/day.`);
    break;
  }
}

if (!success) {
  return { success: false, placedLessons: 0, message: 'Execution timeout due to tight constraint grid.' };
}

await Schedule.destroy({ where: { ScheduleVersionId: versionId } });
await Schedule.bulkCreate(finalScheduleProposals);

return {
  success: true,
  placedLessons: totalTasks,
  message: 'Schedule grid successfully generated via student-centric Stochastic CSP.'
};
```

Рисунок 4.12 – Адаптивне послабленням умов генерації (кінець методу Solve)

З початку генерації лічильник «relaxationBuffer» дорівнює нулю і система намагається побудувати розклад з урахуванням всіх обмежень. Якщо конфігурація обмежень встановлена так, що ідеального рішення математично може не існувати, то покрокове повернення назад працювало би вічно. Тому, коли перша спроба функції «solve» повертає невдачу, скидаються всі накопичені дані, повністю очищає оперативні сховища занять і інкрементується «relaxationBuffer». З кожним новим колом цього циклу (максимум до 4) ліміт допустимих пар на день для студентів стає більшим і це дозволяє алгоритму спробувати «ущільнити» розклад. Як тільки функція «solve» знаходить правильну комбінацію і повертає, цикл переривається та система атомарно видаляє стару версію розкладу з БД і записує нову.

Якщо ж навіть при максимальному послабленні умов розклад скласти не вдалося, спроба генерації зупиняється (relaxationBuffer <= 4).

4.2 Керівництво користувача

Для пояснення методу взаємодії користувача з усіма ключовими моментами вебсистеми використовується керівництво користувача. У підрозділі детально описано інтерфейс розроблюваної системи, щоб дозволити новим користувачам швидко отримати необхідну для повноцінного користування застосунком

інформацію. Оскільки система орієнтована на декілька типів користувачів (адміністратор, методист, викладач та гість), то візуальний простір представлений для всіх цих ролей.

Реєстрація навчального закладу

Після переходу на головну сторінку платформи, користувач який зацікавився в її використанні може відправити запит на реєстрацію свого навчального закладу (рис. 4.13). Для цього він має заповнити відповідну реєстраційну форму і вказати назву свого закладу, тип цієї установи та електронну адресу з ПІБ майбутнього методиста, якому буде видано дозвіл на користування платформою. Усі поля є обов'язковими для заповнення. Також, доступна зміна локалізації між українською та англійською мовами. Після підтвердження форми запит на реєстрацію відправляється на розгляд до адміністрації сайту і користувач має очікувати на відповідь на вказаний емейл.

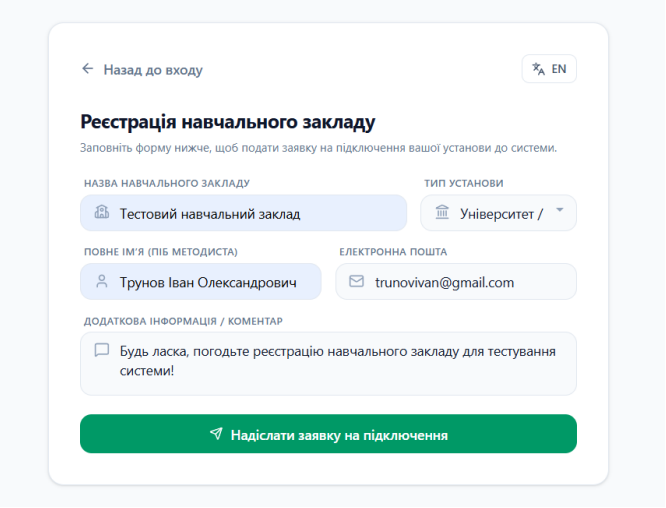

Рисунок 4.13 – Форма реєстрації навчального закладу

Панель адміністратора

Головна панель адміністратора (рис. 4.14) призначена лише для користувачів цієї ролі і містить всі необхідні для управління платформою інструменти. Адміністратор має доступ до перегляду всіх зареєстрованих закладів і засобів редагування даних про заклад і його штаб працівників, видалення як окремих акаунтів, так і всього закладу з системи. На даній сторінці додатково

представлені черга запитів на реєстрацію, в якій адміністратор може приймати рішення щодо додавання нових навчальних закладів в системи та черга на видалення, яка дає можливість відновити акаунт закладу або користувача, якщо цей запит був створений помилково.

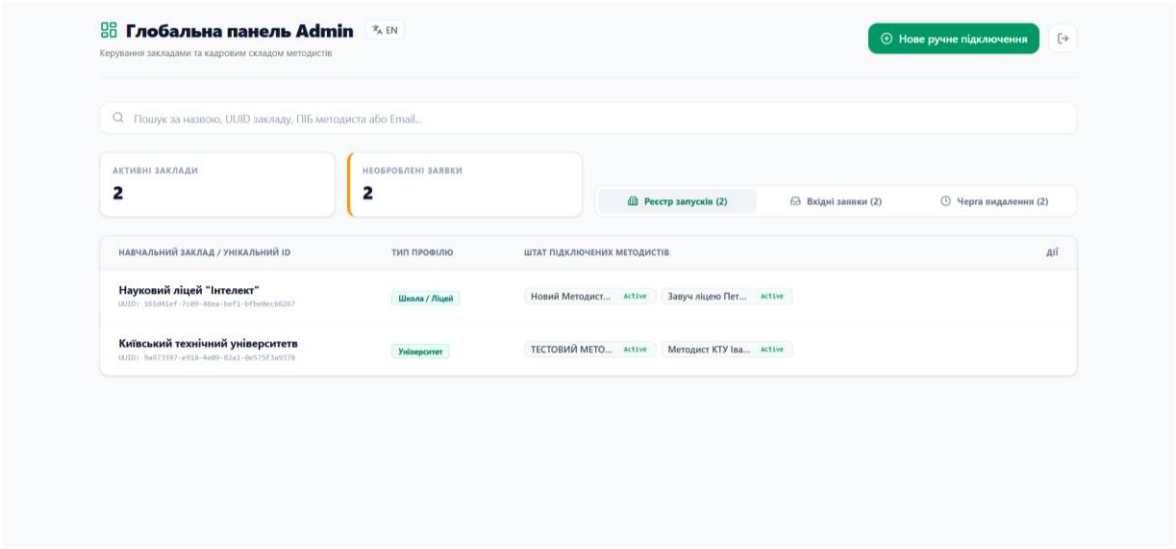

Рисунок 4.14 – Головна сторінка панелі адміністратора

Якщо у адміністратора виникає необхідність керування даними якогось навчального закладу системи, він може натиснути на відповідну кнопку, яка з’являється біля поля таблиці з переліком всіх зареєстрованих установ. Після цього адміністратор потрапляє на форму модифікації даних (рис. 4.15) із доступними полями з інформацією як про заклад, так і про його методистів. Адміністратор може виконувати операції зміни назви навчального закладу та його типу або ж обрати конкретного працівника зі штату і виконувати модифікацію його ПІБ, електронної пошти або ж заблокувати його можливість входу до системи. Також, адміністратор зберігає за собою можливість відправити навчальний заклад або ж його працівників у чергу на видалення (автоматичне видалення відбувається через 48 годин після потрапляння запиту в чергу).

Окрім цього, дана форма надає можливість за допомогою натискання кнопки «Додати нового» вручну ініціювати створення нового методиста обраного навчального закладу.

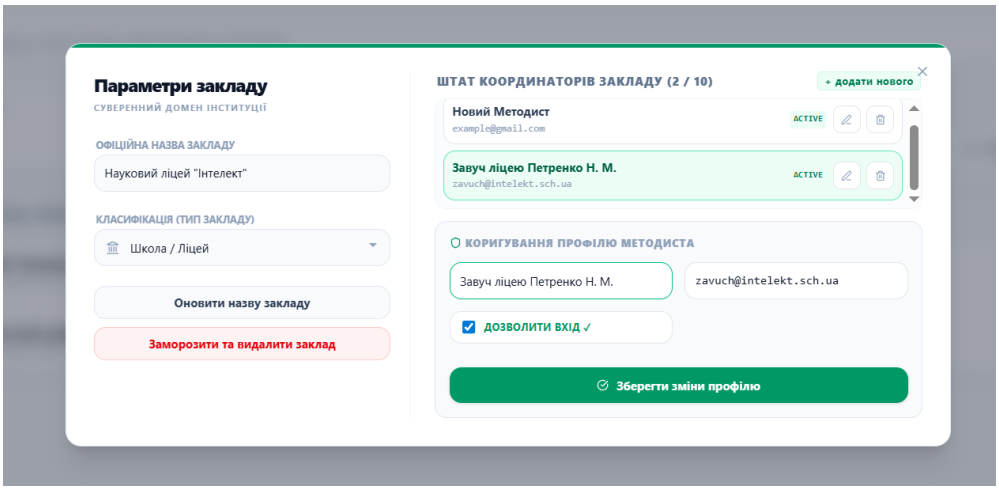

Рисунок 4.15 – Форма модифікації даних про навчальний заклад та його працівників

При переході на таблицю «Вхідні заявки» адміністратор може переглянути всі активні заявки і приймати рішення стосовно їх додавання в систему. Дана таблиця містить інформацію яку гість заповнює в формі реєстрації навчального закладу і відповідні кнопки для схвалення заявки та її відхилення (рис. 4.16).

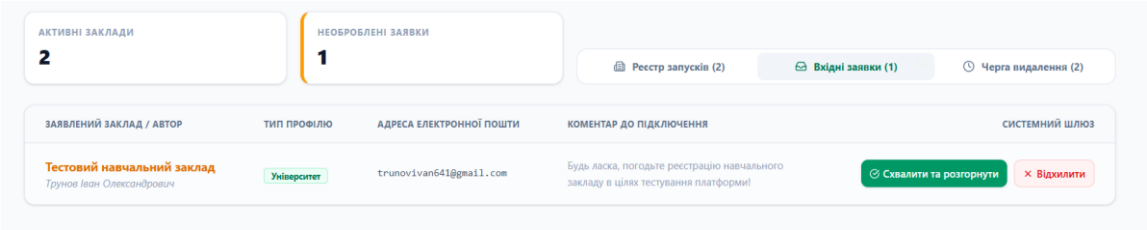

Рисунок 4.16 – Сторінка вхідних заявок панелі адміністратора

Якщо адміністратор вирішує погодити заявку, то він потрапляє після натискання кнопки на форму, яка дозволяє вручну змінити дані заявника, якщо це необхідно (рис. 4.17). Після погодження цієї заявки користувач отримує повідомлення на свій Email з назвою створеного закладу та логіном і паролем, який був згенерований або ж встановлений вручну адміністратором (рис. 4.18). Користувач завжди отримує в листі рекомендацію змінити даний тимчасовий пароль одразу після прешого входу в аккаунт.

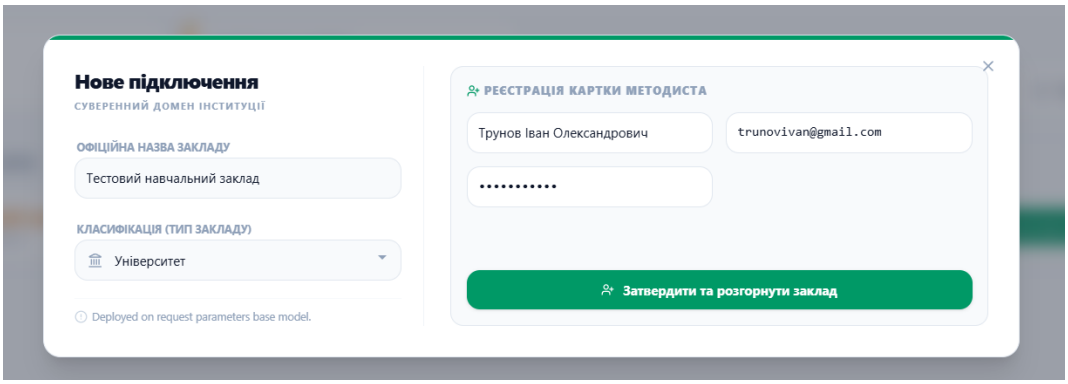

Рисунок 4.17 – Форма підтвердження заявки на реєстрацію нового закладу та його методиста

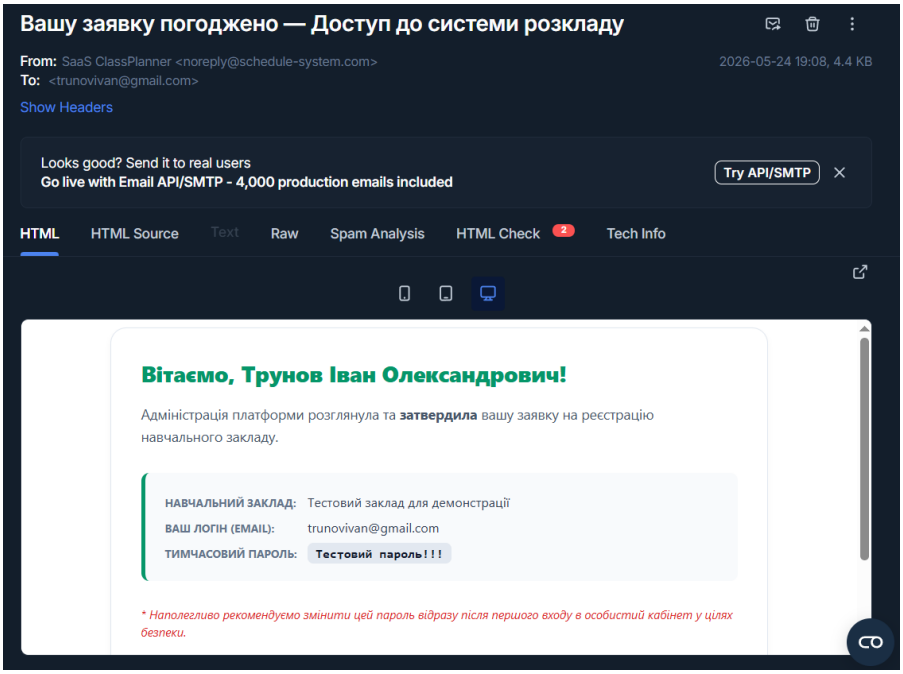

Рисунок 4.18 – Приклад листа-підтвердження реєстрації

Сторінка логіну

На головній сторінці системи, користувач окрім реєстрації нового навчального закладу, може перейти до сторінки логіну (рис. 4.19). Якщо користувач вже має аккаунт і отримав розсилку з тимчасовим паролем, він може скористатися цими даними для входу в систему. Також, в залежності від встановлення прапорця «Remember me» або ні, сесія збережеться на 30 днів чи 2 години відповідно. Сторінка є адаптивною для мобільних пристроїв, тому методисти і викладачі можуть користуватися цією точкою входу в систему і на смартфонах.

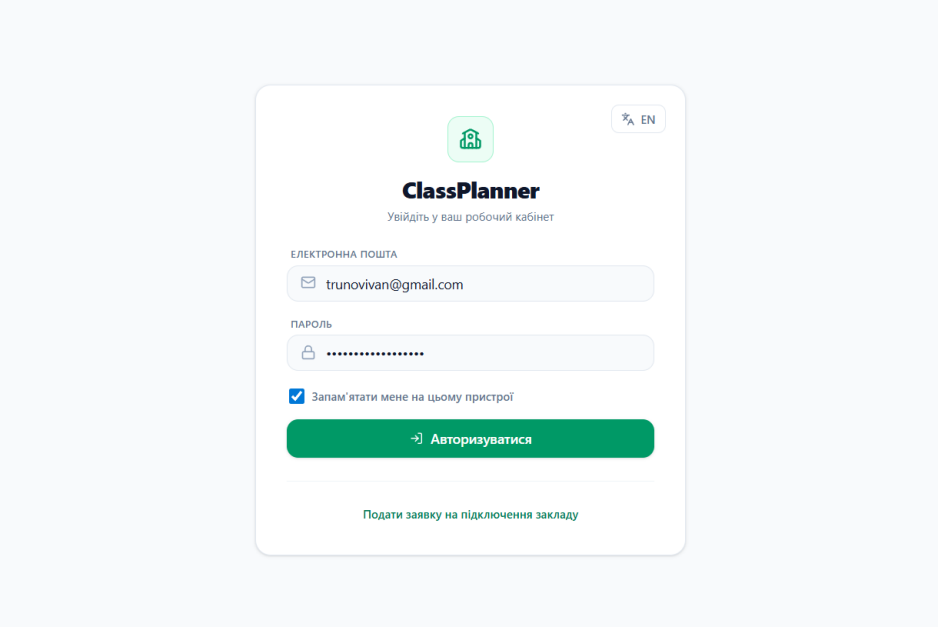

Рисунок 4.19 – Сторінка логіну

Панель методиста

Якщо залогінений користувач має роль «Методист», то система автоматично перенаправляє його на відповідну панель (рис. 4.20). Головна сторінка методиста містить інформацію про поточну версію розкладу і можливість скопіювати публічне посилання для доступу до сітки цього розкладу, яку методист може передати навчальним групам та викладачам. На головному меню розміщеному в лівій частині екрану методист може виконувати перехід між основними функціями його панелі.

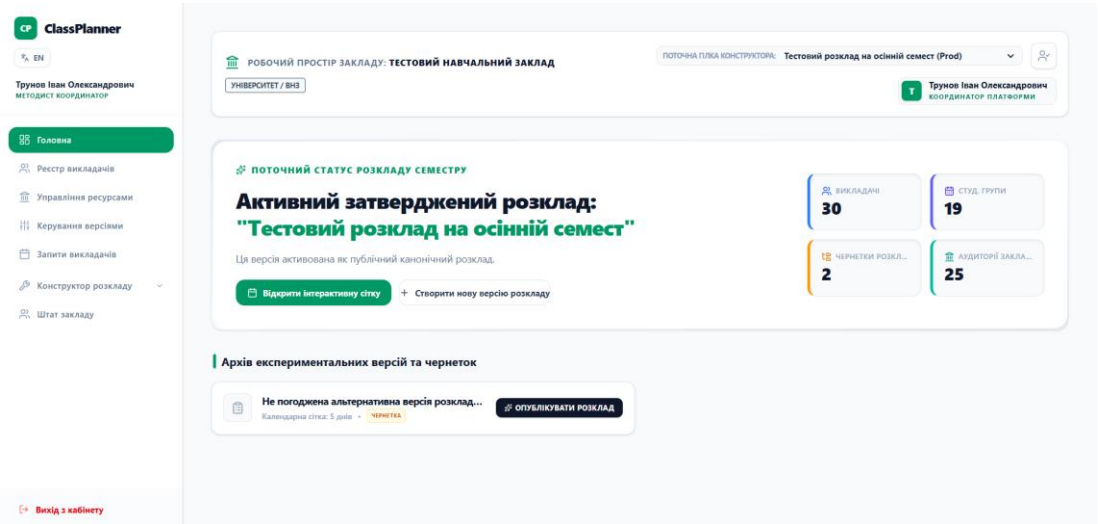

Рисунок 4.20 – Головна сторінка панелі методиста

Якщо методист обирає в головному меню «Реєстр викладачів», то він може виконати перехід до редагування обліковими записами викладачів його навчального закладу (рис. 4.21). Представлено цей перелік у вигляді таблиці з можливістю ручного заповнення ПІБ викладача, його електронної адреси, паролю для здійснення доступу до системи та кафедрою до якої він належить. Перелік викладачів не прикріплюється до конкретної версії розкладу і може перевикористовуватися між ними.

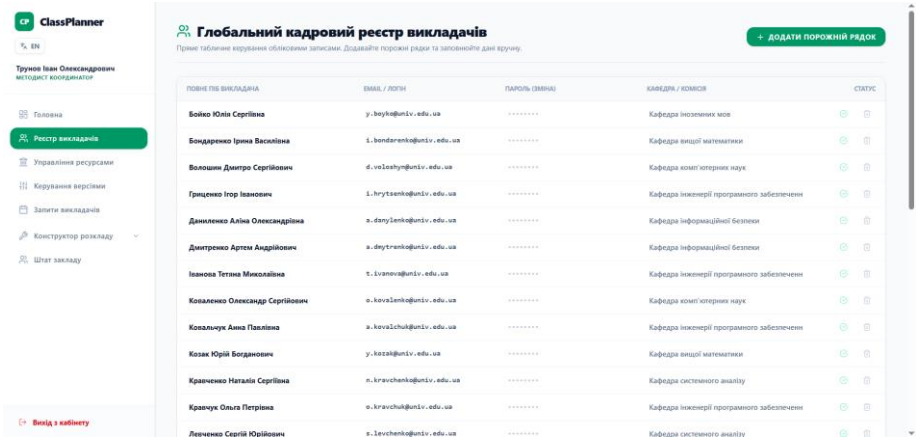

Рисунок 4.21 – Сторінка реєстру викладачів

При переході до пункту меню «Управління ресурсами» (рис. 4.22), методист отримує змогу до ручного заповнення даних про аудиторії, групи та навчальні плани, які необхідні для генерації розкладу. Представлення даних, як і для реєстру викладачів виконано у вигляді таблиці, але на відміну від викладачів відбувається прив’язка до поточної версії розкладу.

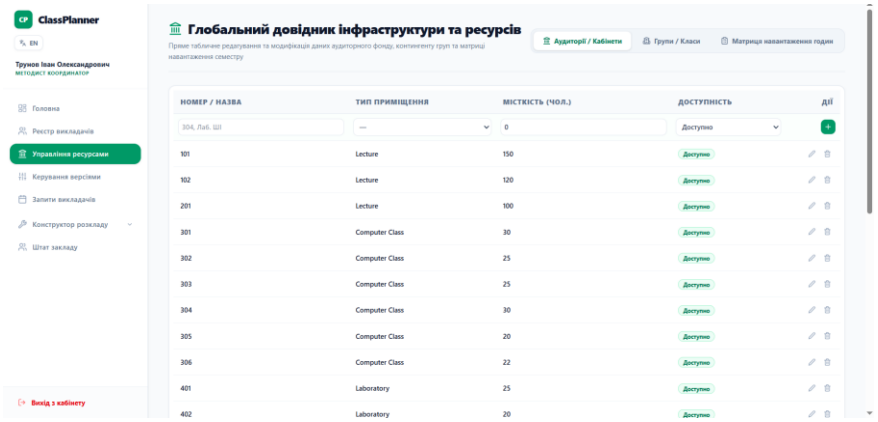

Рисунок 4.22 – Сторінка управління ресурсами

Якщо методист бажає розпочати генерацію розкладу, то він має звернутися до пункту меню «Конструктор розкладу». Дане меню складається з 7 кроків під час яких у методиста є право на фінальну корекцію даних перед генерацією. Тут створюється версія розкладу, перевіряються чи додаються дані про аудиторії, викладачів, групи та навчальні плани. Модифікація даних виконана у вигляді форми заповнення даних, а не у вигляді таблиці як для сторінок «управління ресурсами» та «реєстру викладачів» (рис. 4.23).

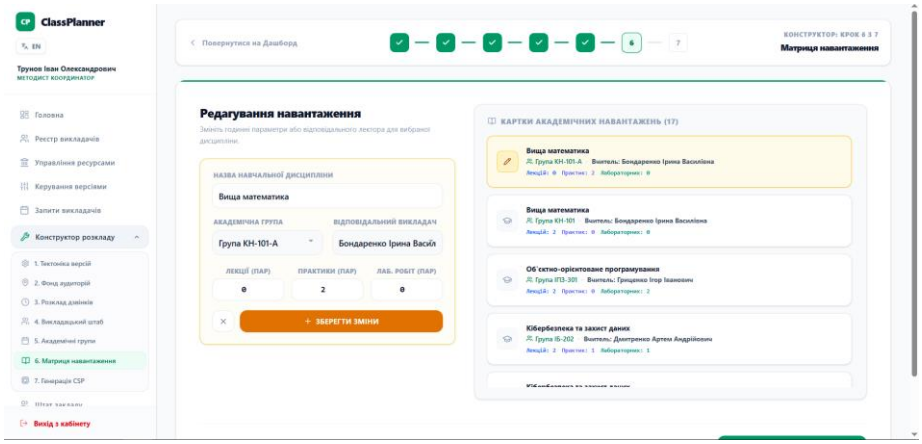

Рисунок 4.23 – Форма заповнення даних про навчальні плани

На фінальному кроці конструктора розкладу (рис. 4.24) методист отримує статистику про заповнені довідникові дані та може викликати генерацію розкладу. В інтерактивному вікні під час проходження генерації відбувається логування цього процесу, щоб методист міг відслідковувати як проходить генерація. Після успішного виконання автоматизованого складання розкладу, система рендерить кнопку за якою методист може перейти до перегляду розкладу.

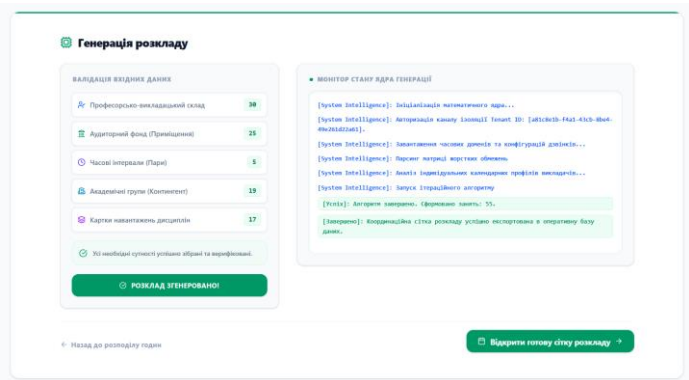

Рисунок 4.24 – Сторінка генерації розкладу

Сторінка перегляду розкладу

На сторінці перегляду розкладу (рис. 4.25) всі користувачі системи можуть переглядати згенеровану сітку за прив’язкою до групи, викладача або ж аудиторії. Методист після генерації розкладу може скопіювати публічне посилання, яке надає доступ до перегляду навіть для неавторизованих користувачів. Таким чином студенти не мають необхідності у створенні аккаунту і можуть просто переглядати розклад за публічним шляхом.

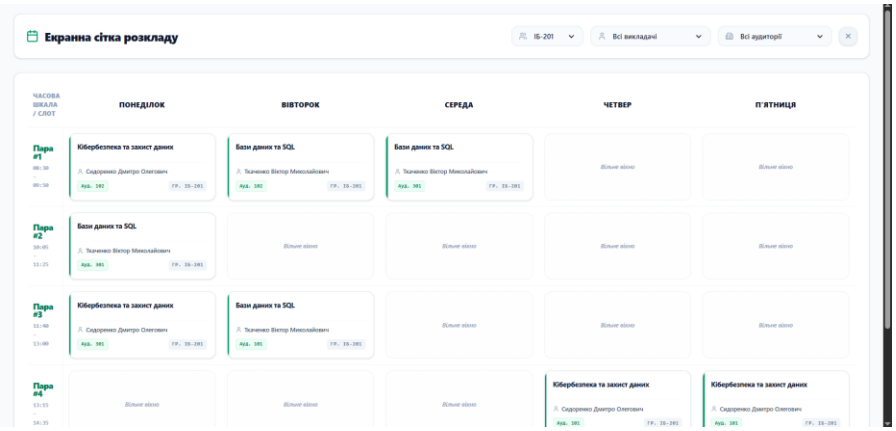

Рисунок 4.25 – Сторінка перегляду сітки розкладу

Сторінка повністю адаптована до використання на мобільних пристроях щоб студенти чи викладачі могли вільно переглядати розклад під час навчального дня (рис. 4.26).

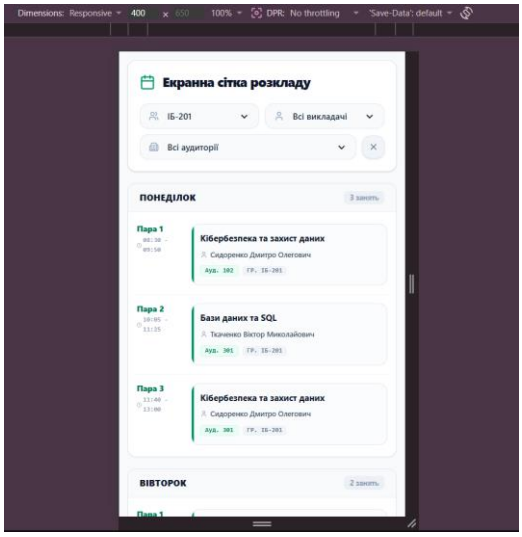

Рисунок 4.26 – Режим перегляду сторінки сітки розкладу на мобільних пристроях

4.3 Тестування застосунку

Для оцінки якості створеного застосунку та швидкості генерації розкладу модулем автоматизації доцільно виконати навантажувальне тестування. Згідно до міжнародного стандартну оцінки ефективності продуктивності та використання системних ресурсів (ISO/IEC 25010) [24] було спроєктовано серію тестів, спрямованих на фіксацію тимчасової поведінки сервера (Time-behaviour) та спостерення обсягів споживання оперативної пам'яті (Resource utilization) під впливом лінійного зростання обсягу вхідних даних. Тести змодельовані таким чином, щоб відтворити випадки від мінімального робочого навантаження до критичного режиму, за якого виникає ресурсний або логічний тупик в системі.

Зміну швидкості генерації від обсягу вхідних даних та навантаження процесору і оперативної пам’яті зафіксовано за допомогою логування в текстовий файл. Статистичні метрики збираються прямо в сервісі автоматичної генерації і дозаписуються в логфайл (рис. 4.27).

```
async function logPerformanceMetrics(metrics) {
  const logPath = path.resolve(process.cwd(), 'generation_metrics.txt');
  const timestamp = new Date().toISOString();

  const logEntry = [
    `================================================`,
    `TIMESTAMP:       ${timestamp}`,
    `INSTITUTION ID:  ${metrics.institutionId}`,
    `TOTAL VARIABLES: ${metrics.totalTasks} (Lesson Cards)`,
    `EXECUTION TIME:  ${metrics.durationSeconds} seconds`,
    `AVERAGE CPU:     ${metrics.cpuPercentage}%`,
    `PEAK RAM USAGE:  ${metrics.memoryUsedMB} MB`,
    `STATUS:          ${metrics.status.toUpperCase()}`,
    `================================================\n\n`
  ].join('\n');

  try {
    await fs.appendFile(logPath, logEntry, 'utf8');
  } catch (error) {
    console.error(`[Performance Logger Error]: Failed to write metrics to file: ${error.message}`);
  }
}
```

Рисунок 4.27 – Функція логування про метод автоматизованої генерації розкладу

Також, для відслідковування обсягу даних задіяних під час тесту, виконується логування загальної кількості змінних моделі CSP, тобто карт занять (totalTasks: totalTasksCount). Це головний показник масштабу вхідного масиву.

На основі отриманих даних логування було сформовано таблицю результатів (табл. 4.1).

Таблиця 4.1 – Результати навантажувального тестування

№	Обсяг даних	Час	Завантаження CPU, %	Пікове RAM	Статус
1	50	0,119	26,0	3,84мб	SUCCESS
2	100	0,129	109,6	6,09мб	SUCCESS
3	150	0,145	118,9	2,15мб	SUCCESS
4	200	0,155	150,7	3,02мб	SUCCESS
5	250	0,181	161,9	4,71мб	SUCCESS

Згідно до отриманих логів можна зробити висновок, що при збільшенні обсягу даних відсутнє лінійне збільшення часу виконання. Це зумовлено використанням інтелектуальної евристики. Замість сліпого перебору, алгоритм із самого початку фокусується на «найважчих» викладачах та групах, миттєво відсікаючи тупикові гілки пошуку.

Навантаження процесору вище 100% починаючи з другого тесту виникє через те, що важкі математичні розрахунки було повністю винесено в окремий системний потік за допомогою Worker Threads. Починаючи з другого тесту Node.js вийшов за рамки одного ядра процесора і задача виконувалась в кілька потоків. Низьке використання оперативної пам'яті в усіх тестах зумовлено покроковим поверненням назад. При вході в глухий кут алгоритм миттєво затирає за собою колекції тимчасових даних, тобто відбувається динамічне очищення пам’яті.

Виконані тести навантаження дозволили оцінити поведінку системи при різних обсягах даних та підтвердили її працездатність.

Висновки до розділу 4

В четвертому розділі кваліфікаційної бакалаврської роботи виконано реалізацію ПЗ та аналіз внутрішньої архітектури автоматизованої системи планування розкладу навчальних занять. Розроблено серверну частину (backend)

вебзастосунку на базі середовища Node.js та фреймворка Express.js із дотриманням архітектурного патерну MVC. Налаштовано механізм пулу з'єднань за допомогою ORM-бібліотеки Sequelize. Побудовано модуль безпеки та автентифікації, який використовує криптографічний алгоритм Bcrypt для безпечного збереження паролів користувачів.

Реалізовано CSP-двигун для автоматичного розрахунку сітки занять. Логіка генерації реалізована на основі рекурсивного алгоритму пошуку із покроковим поверненням назад та евристичному сортуванні вхідного масиву за принципом «найбільш обмеженої змінної» (MRV). Уникнення нескінченних рекурсивних циклів у випадку суперечливих вхідних обмежень відбувається механізмом адаптивного послаблення умов на основі інкрементного буфера relaxationBuffer.

Сформовано керівництво користувача (клієнтської частини React), яке візуалізує та описує інтерфейсні рішення вебзастосунку, побудованої за допомогою бібліотеки React.

Створено функціональний вебсайт, що забезпечує автоматизоване складання розкладу навчальних занять та виконано навантажувальне тестування для модуля генерації.

ВИСНОВКИ

Під час виконання кваліфікаційної бакалаврської роботи на тему «Автоматизована система планування розкладу навчальних занять» розроблено вебзастосунок який вирішує задачу автоматизації та оптимізації процесу формування розкладу в закладах освіти. Досягнено поставленої мети відповідно до визначених завдань.

Виконано аналіз предметної області, пов’язаної з автоматизацією планування розкладу навчальних занять. Досліджено існуючі аналоги, такі як aSc TimeTables, FET та Microsoft Excel. У результаті проведеного аналізу сформульовано функціональні особливості вебсайту, що розробляється та обгрунтовано вибір клієнт-серверної архітектури.

Здійснено моделювання структури системи за допомогою діаграми «сутність-зв'язок», діаграми прецедентів, сценаріїв використання, діаграм класів та діяльності. Систему реалізовано з використанням сучасних інтруменів. Врахувавши необхідність виконання умов високої продуктивності та масштабованості, вебзастосунок створено стеком PERN (PostgreSQL, Express.js, React, Node.js).

Програмно реалізовано клієнтську та серверну частини вебзастосунку з адаптивним та зручним інтерфейсом користувача, з інтуїтивно зрозумілим робочим простором та інструментом автоматичної генерації. Виконано навантажувальне тестування, яке підтвердило працездатність системи з різними обсягами вхідних даних. Розроблений вебсайт надає користувачам можливість адміністрування та розгортання навчального процесу для закладів освіти із зручним режимом доступу до довідкових даних, створення, редагування та перегляду розкладу навчальних занять.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Musa U. B., Oyelakin A. M. A Survey of Approaches for Designing Course Timetable Scheduling Systems in Tertiary Institutions. Information Technology. 2024. Vol. 3, № 1. P. 4-5.

2. M M. et al. The implementation of e-learning in the Arab Universities: Challenges and opportunities. Science. 2013. Vol. 3, № 1. P. 3-6.

3. Koliechkina L. et al. The Higher Educational Information System: Management of the Timetable Scheduling. CEUR Workshop Proceedings. 2023. Vol. 3641. P. 112-124.

4. aSc TimeTables. URL: https://www.asctimetables.com/ (Accessed:: 25.04.2026).

5. FET — Free Timetabling Software. URL: https://lalescu.ro/liviu/fet/ (Accessed:: 25.04.2026).

6. Microsoft Excel. URL: https://excel.cloud.microsoft/ (Accessed:: 25.04.2026).

7. Gnecco G. et al. Learning with mixed hard/soft pointwise constraints. IEEE Transactions on Neural Networks and Learning Systems. 2015. Vol. 26, № 9. P. 2019-2032.

8. Garey M. R., Johnson D. S. Computers and Intractability: A Guide to the Theory of NP-Completeness. San Francisco : W. H. Freeman and Company, 1979. 340 p.

9. Quantum advantage and CSP complexity. Journal of Combinatorial Theory, Series B. 2025. Vol. 170. URL: https://www.sciencedirect.com/science/article/pii/S0095895625000759 (Accessed: 25.04.2026)

10. Abdipoor S. et al. Meta-heuristic approaches for the University Course Timetabling Problem. Intelligent Systems with Applications. 2023. Vol. 19. URL: https://www.sciencedirect.com/science/article/pii/S0377042725006545 (Accessed: 25.04.2026).

11. Node.js — Run JavaScript Everywhere. URL: https://nodejs.org/ (Accessed:: 25.04.2026).

12. Express — Fast, unopinionated, minimalist web framework for Node.js. URL: https://expressjs.com/ (Accessed:: 25.04.2026).

13. PostgreSQL: The World's Most Advanced Open Source Relational Database. URL: https://www.postgresql.org/ (Accessed:: 25.04.2026).

14. Sequelize ORM for Node.js. URL: https://sequelize.org/ (Accessed:: 25.04.2026).

15. React — A JavaScript library for building user interfaces. URL: https://react.dev/ (Accessed:: 25.04.2026).

16. Tailwind CSS — Rapidly build modern websites without ever leaving your HTML. URL: https://tailwindcss.com/ (Accessed: 25.04.2026).

17. JSON Web Tokens — JWT.io. URL: https://jwt.io/ (дата звернення: 25.04.2026).

18. Bcrypt — adaptive hashing library. URL: https://www.npmjs.com/package/bcrypt (Accessed: 25.04.2026).

19. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley, 2004. 208 p.

20. What is an Entity Relationship Diagram (ERD)? Lucidchart. URL: https://www.lucidchart.com/pages/er-diagrams (Accessed: 25.04.2026).

21. ISO/IEC 19505-1:2012. Information technology — Object Management Group Unified Modeling Language (OMG UML) — Part 1: Infrastructure. Geneva : International Organization for Standardization, 2012.

22. Software Ideas Modeler — UML Diagram Tool and CASE Software. URL: https://www.softwareideas.net/ (Accessed: 25.04.2026).

23. Cockburn A. Writing Effective Use Cases. Boston : Addison-Wesley Professional, 2000. 270 p.

24. Krasner G. E., Pope S. T. A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. Journal of Object-Oriented Programming. 1988. Vol. 1, № 3. P. 26-49.

25. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. Geneva : International Organization for Standardization, 2011. 34 p.

ДОДАТОК А

Діаграми UML

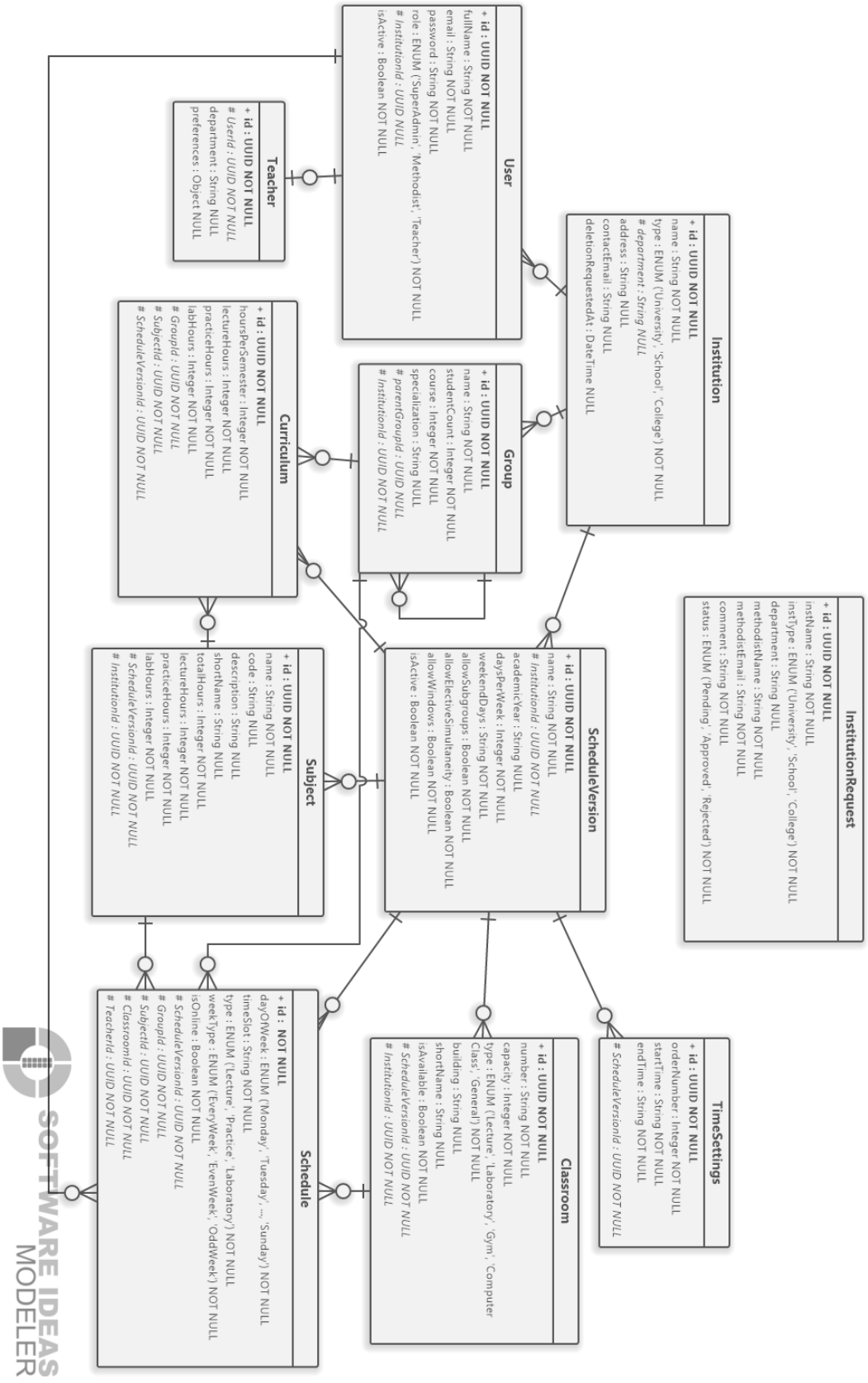

Рисунок А.1 – Діаграма «сутність-зв'язок»

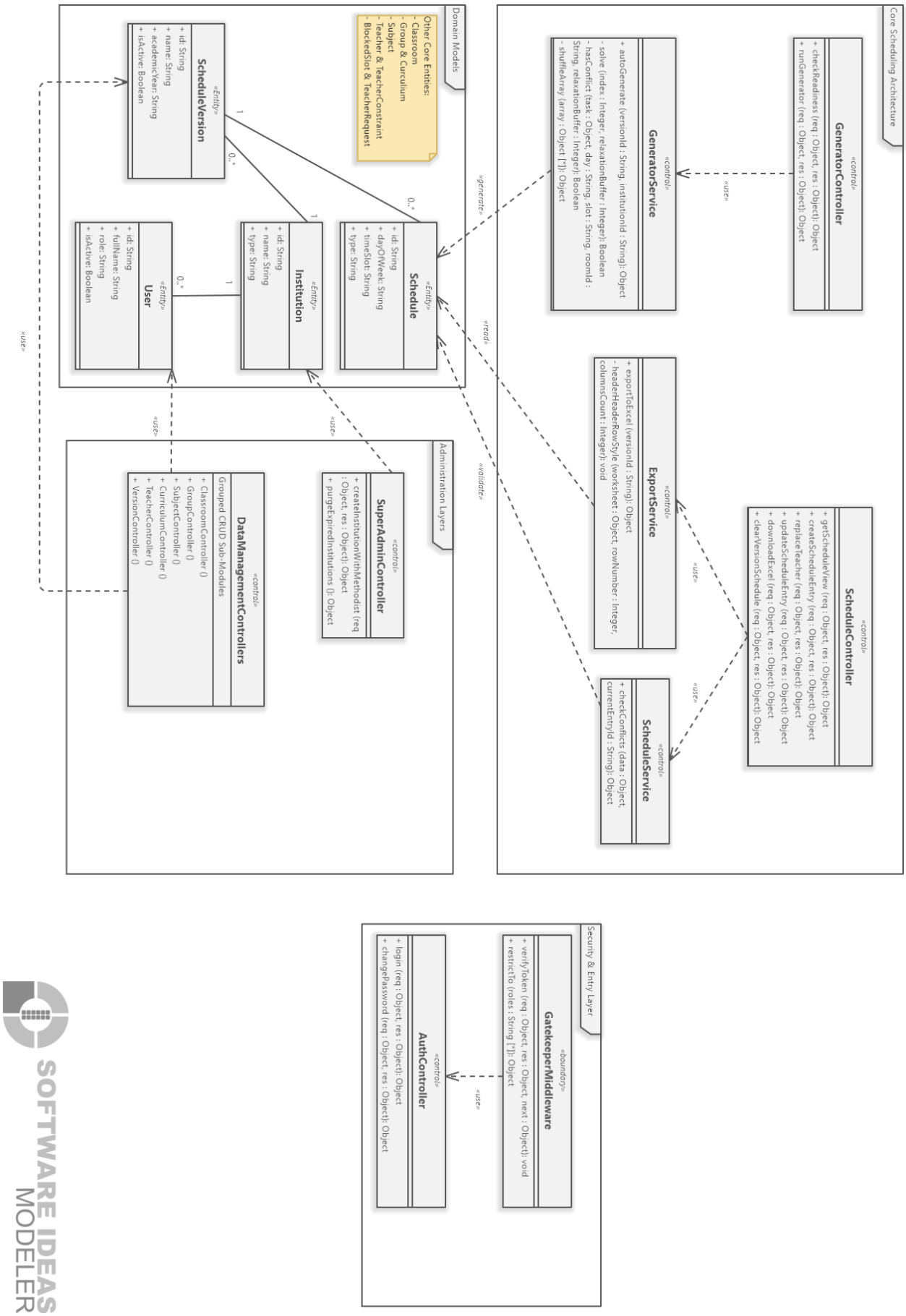

Рисунок А.2 – Діаграма класів

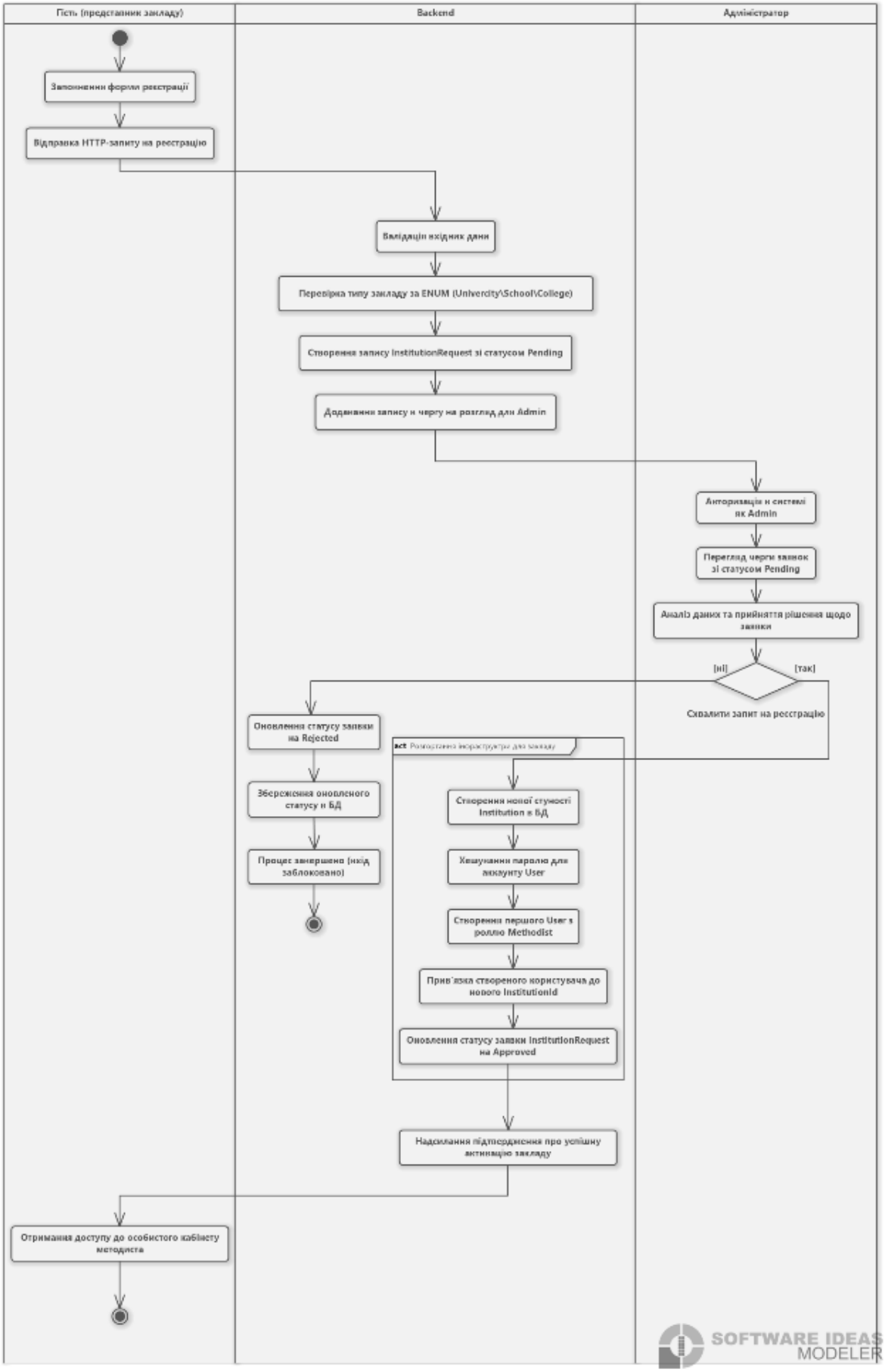

Рисунок А.3 – Діаграма діяльності «Реєстрація нового навчального закладу»

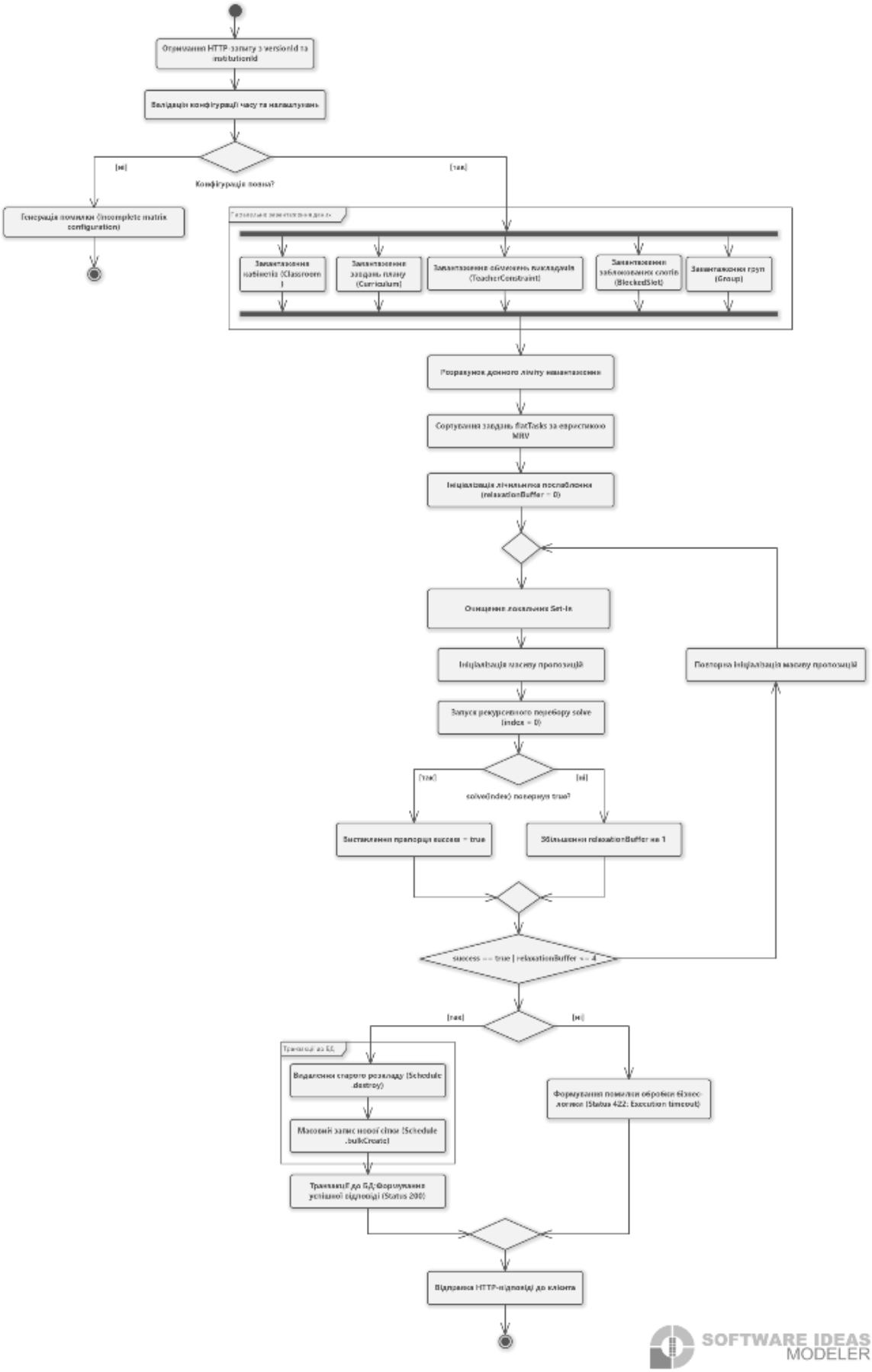

Рисунок А.4 – Діаграма діяльності «Автоматична генерація розкладу»

ДОДАТОК Б

Лістинг програмного модулю генерації розкладу

```
      const { Worker } = require('worker_threads');
      const path = require('path');
      const fs = require('fs').promises;
      const { performance } = require('perf_hooks');

      const {
        Group,
        Classroom,
        TimeSettings,
        Curriculum,
        Schedule,
        ScheduleVersion,
        TeacherConstraint,
        BlockedSlot
      } = require('../models');

      /**
       * Appends performance metrics report to a txt file.
       */
      async function logPerformanceMetrics(metrics) {
        const logPath = path.resolve(process.cwd(), 'generation_metrics.txt');
        const timestamp = new Date().toISOString();

        const logEntry = [
          `================================================`,
          `TIMESTAMP:       ${timestamp}`,
          `INSTITUTION ID:  ${metrics.institutionId}`,
          `TOTAL VARIABLES: ${metrics.totalTasks} (Lesson Cards)`,
          `EXECUTION TIME:  ${metrics.durationSeconds} seconds`,
          `AVERAGE CPU:     ${metrics.cpuPercentage}%`,
          `PEAK RAM USAGE:  ${metrics.memoryUsedMB} MB`,
          `STATUS:          ${metrics.status.toUpperCase()}`,
          `================================================\n\n`
        ].join('\n');

        try {
          await fs.appendFile(logPath, logEntry, 'utf8');
        } catch (error) {
          console.error(`[Performance Logger Error]: Failed to write metrics to
file: ${error.message}`);
        }
      }

      exports.runInWorker = (versionId, institutionId) => {
        return new Promise((resolve, reject) => {
```

```
          const workerPath = path.resolve(__dirname, 'generatorWorker.js');
          const worker = new Worker(workerPath, {
            workerData: { versionId, institutionId }
          });

          worker.on('message', (message) => {
            if (message.success) resolve(message.data);
            else reject(new Error(message.error));
          });
          worker.on('error', (error) => reject(error));
          worker.on('exit', (code) => {
            if (code !== 0) reject(new Error(`Worker thread stopped unexpectedly
with exit code ${code}`));
          });
        });
      };

      exports.autoGenerate = async (versionId, institutionId) => {
        const startTime = performance.now();
        const startCPU = process.cpuUsage();
        const startMemory = process.memoryUsage().heapUsed;

        let totalTasksCount = 0;
        let finalRelaxationBuffer = 0;
        let currentPhaseStatus = 'failed';

        try {
          const version = await ScheduleVersion.findByPk(versionId);
          const timeSettings = await TimeSettings.findAll({
            where: { ScheduleVersionId: versionId },
            order: [['orderNumber', 'ASC']]
          });

          if (!version || timeSettings.length === 0) {
            throw new Error('Incomplete version configuration matrix or time slots
missing');
          }

          const allDays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday'];
          const activeDays = allDays.slice(0, version.daysPerWeek || 5);
          const activeSlots = timeSettings.map(s => String(s.orderNumber));

          const classrooms = await Classroom.findAll({ where: { InstitutionId:
institutionId, isAvailable: true } });
          const curriculumTasks = await Curriculum.findAll({
            where: { ScheduleVersionId: versionId },
            include: [{ model: Group, where: { InstitutionId: institutionId } }]
          });
```

```
          let allConstraints = [];
          try { allConstraints = await TeacherConstraint.findAll({ where: {
ScheduleVersionId: versionId } }); } catch (e) {
            try { allConstraints = await TeacherConstraint.findAll().catch(() =>
[]); } catch (err) { allConstraints = []; }
          }

          let globalBlocks = [];
          try { globalBlocks = await BlockedSlot.findAll({ where: {
ScheduleVersionId: versionId } }); } catch (e) {
            try { globalBlocks = await BlockedSlot.findAll().catch(() => []); }
catch (err) { globalBlocks = []; }
          }

          const allGroups = await Group.findAll({
            where: { InstitutionId: institutionId },
            attributes: ['id', 'parentGroupId', 'studentCount']
          }).catch(() => []);

          const groupMap = {};
          const groupSizes = {};
          allGroups.forEach(g => {
            groupMap[g.id] = g.parentGroupId;
            groupSizes[g.id] = g.studentCount || 0;
          });

          const systemBlockedKeys = new Set();
          globalBlocks.forEach(b => { if (b.dayOfWeek)
systemBlockedKeys.add(b.dayOfWeek); });

          const forbiddenTeacherSlots = new Set();
          allConstraints.forEach(c => {
            if (c.TeacherId && c.dayOfWeek && c.timeSlot) {
              forbiddenTeacherSlots.add(`${c.TeacherId}-${c.dayOfWeek}-
${c.timeSlot}`);
            }
          });

          let flatTasks = [];
          const studentTotalLoad = {};

          for (const task of curriculumTasks) {
            const types = [
              { type: 'Lecture', pairs: task.lectureHours || 0 },
              { type: 'Laboratory', pairs: task.labHours || 0 },
              { type: 'Practice', pairs: task.practiceHours || 0 }
            ];

            for (const t of types) {
              for (let i = 0; i < t.pairs; i++) {
```

```
                flatTasks.push({
                  type: t.type,
                  GroupId: task.GroupId,
                  SubjectId: task.SubjectId,
                  TeacherId: task.TeacherId
                });

                const parentId = groupMap[task.GroupId];
                if (parentId) {
                  studentTotalLoad[task.GroupId] = (studentTotalLoad[task.GroupId]
|| 0) + 1;
                } else {
                  studentTotalLoad[task.GroupId] = (studentTotalLoad[task.GroupId]
|| 0) + 1;
                  allGroups.forEach(g => {
                    if (g.parentGroupId === task.GroupId) {
                      studentTotalLoad[g.id] = (studentTotalLoad[g.id] || 0) + 1;
                    }
                  });
                }
              }
            }
          }

          totalTasksCount = flatTasks.length;

          if (flatTasks.length === 0) {
            throw new Error('Curriculum requirements are empty.');
          }

          const totalDaysCount = activeDays.filter(d =>
!systemBlockedKeys.has(d)).length || 5;
          const studentDailyCap = {};
          Object.keys(studentTotalLoad).forEach(gId => {
            studentDailyCap[gId] = Math.ceil(studentTotalLoad[gId] / totalDaysCount)
+ 1;
          });

          allGroups.forEach(parent => {
            if (!parent.parentGroupId) {
              let maxChildCap = studentDailyCap[parent.id] || 0;
              allGroups.forEach(child => {
                if (child.parentGroupId === parent.id) {
                  if ((studentDailyCap[child.id] || 0) > maxChildCap) {
                    maxChildCap = studentDailyCap[child.id];
                  }
                }
              });
              if (maxChildCap > 0) studentDailyCap[parent.id] = maxChildCap;
            }
```

```
          });

          flatTasks.sort((a, b) => {
            const getScore = (id) => {
              if (!id) return 100;
              const constraintsCount = allConstraints.filter(cons => cons.TeacherId
=== id).length;
              return constraintsCount > 0 ? (100 - constraintsCount) : 100;
            };
            return getScore(a.TeacherId) - getScore(b.TeacherId);
          });

          const shuffleArray = (array) => {
            const arr = [...array];
            for (let i = arr.length - 1; i > 0; i--) {
              const j = Math.floor(Math.random() * (i + 1));
              [arr[i], arr[j]] = [arr[j], arr[i]];
            }
            return arr;
          };

          const roomCacheRandom = {};
          const roomCacheDeterministic = {};

          flatTasks.forEach(task => {
            if (roomCacheRandom[task.type]) return;
            const neededCapacity = groupSizes[task.GroupId] || 0;
            let rooms = classrooms.filter(r => {
              if ((r.capacity || 0) < neededCapacity) return false;
              if (task.type === 'Lecture') return r.type === 'Lecture' || r.type ===
'General';
              if (task.type === 'Laboratory') return r.type === 'Laboratory' ||
r.type === 'Computer Class';
              if (task.type === 'Practice') return r.type === 'General' || r.type
=== 'Lecture' || r.type === 'Computer Class';
              return true;
            });
            if (rooms.length === 0) rooms = classrooms.filter(r => (r.capacity || 0)
>= neededCapacity);

            roomCacheRandom[task.type] = [...rooms];
            rooms.sort((a, b) => (a.capacity || 0) - (b.capacity || 0));
            roomCacheDeterministic[task.type] = rooms;
          });

          let busyTeachers = {};
          let busyGroups = {};
          let busyClassrooms = {};
          let groupDailyLoadTracker = {};
          let finalScheduleProposals = [];
```

```
          let totalBacktracks = 0;
          let MAX_BACKTRACKS = 50000;

          function hasConflict(task, day, slot, roomId, relaxationBuffer) {
            const key = `${day}-${slot}`;
            if (task.TeacherId && busyTeachers[key]?.has(task.TeacherId)) return
true;
            if (busyClassrooms[key]?.has(roomId)) return true;
            if (busyGroups[key]?.has(task.GroupId)) return true;

            const parentId = groupMap[task.GroupId];
            if (parentId && busyGroups[key]?.has(parentId)) return true;
            if (busyGroups[key]) {
              for (const bookedGroupId of busyGroups[key]) {
                if (groupMap[bookedGroupId] === task.GroupId) return true;
              }
            }

            const targetCap = (studentDailyCap[task.GroupId] || 4) +
relaxationBuffer;
            const currentTargetLoad = groupDailyLoadTracker[`${task.GroupId}-
${day}`] || 0;
            if (currentTargetLoad >= targetCap) return true;

            if (parentId) {
              const parentCap = (studentDailyCap[parentId] || 4) + relaxationBuffer;
              const currentParentLoad = groupDailyLoadTracker[`${parentId}-${day}`]
|| 0;
              if (currentParentLoad >= parentCap) return true;
            }
            return false;
          }

          function solve(index, relaxationBuffer, isDeterministic) {
            totalBacktracks++;
            if (totalBacktracks > MAX_BACKTRACKS) return false;
            if (index === flatTasks.length) return true;

            const task = flatTasks[index];
            const suitableRooms = isDeterministic ?
(roomCacheDeterministic[task.type] || classrooms) : (roomCacheRandom[task.type] ||
classrooms);

            const daysOrder = isDeterministic ? activeDays :
shuffleArray(activeDays);
            const slotsOrder = isDeterministic ? activeSlots :
shuffleArray(activeSlots);

            for (const day of daysOrder) {
```

```
              if (systemBlockedKeys.has(day)) continue;

              for (const slot of slotsOrder) {
                const key = `${day}-${slot}`;

                if (task.TeacherId && busyTeachers[key]?.has(task.TeacherId))
continue;
                if (busyGroups[key]?.has(task.GroupId)) continue;
                if (task.TeacherId && forbiddenTeacherSlots.has(`${task.TeacherId}-
${day}-${slot}`)) continue;

                for (const room of suitableRooms) {
                  if (hasConflict(task, day, slot, room.id, relaxationBuffer))
continue;

                  if (!busyTeachers[key]) busyTeachers[key] = new Set();
                  if (!busyGroups[key]) busyGroups[key] = new Set();
                  if (!busyClassrooms[key]) busyClassrooms[key] = new Set();

                  if (task.TeacherId) busyTeachers[key].add(task.TeacherId);
                  busyGroups[key].add(task.GroupId);
                  busyClassrooms[key].add(room.id);

                  groupDailyLoadTracker[`${task.GroupId}-${day}`] =
(groupDailyLoadTracker[`${task.GroupId}-${day}`] || 0) + 1;
                  const parentId = groupMap[task.GroupId];

                  if (parentId) {
                    groupDailyLoadTracker[`${parentId}-${day}`] =
(groupDailyLoadTracker[`${parentId}-${day}`] || 0) + 1;
                  } else {
                    allGroups.forEach(g => {
                      if (g.parentGroupId === task.GroupId) {
                        groupDailyLoadTracker[`${g.id}-${day}`] =
(groupDailyLoadTracker[`${g.id}-${day}`] || 0) + 1;
                      }
                    });
                  }

                  const matchingSlotSetting = timeSettings.find(s =>
String(s.orderNumber) === slot);
                  const timeSlotLabel = matchingSlotSetting
                    ? `${matchingSlotSetting.startTime.substring(0, 5)} -
${matchingSlotSetting.endTime.substring(0, 5)}`
                    : `Slot #${slot}`;

                  finalScheduleProposals.push({
                    dayOfWeek: day, timeSlot: timeSlotLabel, type: task.type,
                    GroupId: task.GroupId, SubjectId: task.SubjectId, TeacherId:
task.TeacherId || null,
```

```
                  ClassroomId: room.id, ScheduleVersionId: versionId, isOnline:
false
                });

                if (solve(index + 1, relaxationBuffer, isDeterministic)) return
true;

                // ROLLBACK
                if (task.TeacherId) busyTeachers[key].delete(task.TeacherId);
                busyGroups[key].delete(task.GroupId);
                busyClassrooms[key].delete(room.id);
                groupDailyLoadTracker[`${task.GroupId}-${day}`]--;
                if (parentId) groupDailyLoadTracker[`${parentId}-${day}`]--;
                else {
                  allGroups.forEach(g => { if (g.parentGroupId === task.GroupId)
groupDailyLoadTracker[`${g.id}-${day}`]--; });
                }
                finalScheduleProposals.pop();
              }
            }
          }
          return false;
        }

        console.log(`[Phase 1]: Launching fast randomized CSP...`);
        MAX_BACKTRACKS = 40000;
        let success = false;

        for (let relaxationBuffer = 0; relaxationBuffer <= 4; relaxationBuffer++)
{
          finalRelaxationBuffer = relaxationBuffer;
          totalBacktracks = 0; busyTeachers = {}; busyGroups = {}; busyClassrooms
= {}; groupDailyLoadTracker = {}; finalScheduleProposals = [];

          if (solve(0, relaxationBuffer, false)) {
            success = true;
            currentPhaseStatus = 'success_phase_1_stochastic';
            console.log(`[Phase 1]: Success! Schedule generated via Stochastic CSP
with buffer +${relaxationBuffer}`);
            break;
          }
        }

        if (!success) {
          console.log(`[Phase 1 -> Failed]: Stochastic engine failed. ACTIVATING
PHASE 2 (Deterministic CSP)...`);
          MAX_BACKTRACKS = 200000;

          for (let relaxationBuffer = 0; relaxationBuffer <= 4;
relaxationBuffer++) {
```

```
              finalRelaxationBuffer = relaxationBuffer;
              totalBacktracks = 0; busyTeachers = {}; busyGroups = {};
busyClassrooms = {}; groupDailyLoadTracker = {}; finalScheduleProposals = [];

              if (solve(0, relaxationBuffer, true)) {
                success = true;
                currentPhaseStatus = 'success_phase_2_deterministic';
                console.log(`[Phase 2]: Success! Deterministic engine resolved tight
matrix constraints with buffer +${relaxationBuffer}`);
                break;
              }
            }
          }

          if (!success) {
            throw new Error('Execution timeout due to tight constraint grid.');
          }

          await Schedule.destroy({ where: { ScheduleVersionId: versionId } });
          await Schedule.bulkCreate(finalScheduleProposals);
          const endTime = performance.now();
          await logPerformanceMetrics({
            institutionId, versionId, totalTasks: totalTasksCount,
            durationSeconds: ((endTime - startTime) / 1000).toFixed(3),
            cpuPercentage: (((process.cpuUsage(startCPU).user +
process.cpuUsage(startCPU).system) / 1000) / (endTime - startTime) *
100).toFixed(1),
            memoryUsedMB: ((process.memoryUsage().heapUsed - startMemory) / 1024 /
1024).toFixed(2),
            status: currentPhaseStatus
          });

          return { success: true, placedLessons: flatTasks.length, message:
`Generated via ${currentPhaseStatus}` };

        } catch (error) {
          const endTime = performance.now();
          await logPerformanceMetrics({
            institutionId, versionId, totalTasks: totalTasksCount,
            durationSeconds: ((endTime - startTime) / 1000).toFixed(3),
            cpuPercentage: (((process.cpuUsage(startCPU).user +
process.cpuUsage(startCPU).system) / 1000) / (endTime - startTime) *
100).toFixed(1),
            memoryUsedMB: ((process.memoryUsage().heapUsed - startMemory) / 1024 /
1024).toFixed(2),
            status: `failed: ${error.message}`
          });
          return { success: false, placedLessons: 0, message: error.message };
        }
      };
```

ДОДАТОК В

Результати тестування

TIMESTAMP: 2026-05-26T20:56:31.169Z
INSTITUTION ID: ac25e25c-6d69-4489-9154-16840becc3ac
TOTAL VARIABLES: 50 (Lesson Cards)
EXECUTION TIME: 0.119 seconds
AVERAGE CPU: 26.0%
PEAK RAM USAGE: 3.84 MB
STATUS: SUCCESS

TIMESTAMP: 2026-05-26T20:56:47.235Z
INSTITUTION ID: ac25e25c-6d69-4489-9154-16840becc3ac
TOTAL VARIABLES: 100 (Lesson Cards)
EXECUTION TIME: 0.129 seconds
AVERAGE CPU: 109.6%
PEAK RAM USAGE: 6.09 MB
STATUS: SUCCESS

TIMESTAMP: 2026-05-26T20:57:10.070Z
INSTITUTION ID: ac25e25c-6d69-4489-9154-16840becc3ac
TOTAL VARIABLES: 150 (Lesson Cards)
EXECUTION TIME: 0.145 seconds
AVERAGE CPU: 118.9%
PEAK RAM USAGE: 2.15 MB
STATUS: SUCCESS

TIMESTAMP: 2026-05-26T20:57:18.397Z
INSTITUTION ID: ac25e25c-6d69-4489-9154-16840becc3ac
TOTAL VARIABLES: 200 (Lesson Cards)
EXECUTION TIME: 0.155 seconds
AVERAGE CPU: 150.7%
PEAK RAM USAGE: 3.02 MB
STATUS: SUCCESS

TIMESTAMP: 2026-05-26T20:58:12.173Z
INSTITUTION ID: ac25e25c-6d69-4489-9154-16840becc3ac
TOTAL VARIABLES: 250 (Lesson Cards)
EXECUTION TIME: 0.181 seconds
AVERAGE CPU: 161.9%
PEAK RAM USAGE: 4.71 MB
STATUS: SUCCESS