НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

На правах рукопису
УДК 519.86

До захисту допущено
Завідувач кафедри ММСА
_____Оксана ТИМОЩУК
«____»_____2025 р.

Магістерська дисертація

**на здобуття ступеня магістра
за освітньо-професійною програмою «Системний аналіз і управління» зі спеціальності 124 «Системний аналіз»
на тему: «Інтелектуальна система автоматизованого складання розкладу з використанням адаптивних гібридних метаевристик»**

Виконав:
студент ІІ курсу, групи КА-41мп
Гнідобор Сергій Михайлович _____

Науковий керівник:
завідувач кафедри ММСА, к.т.н., доц.
Тимощук Оксана Леонідівна _____

Консультант з нормоконтролю:
доцент кафедри ММСА, к.ф.-м.н.,
Статкевич Віталій Михайлович _____

Рецензент:
к.е.н., доцент, старший науковий співробітник відділу прикладної інформатики Інституту телекомунікацій і глобального інформаційного простору НАН України
Просянкіна-Жарова Тетяна Іванівна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ — 2025 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

Рівень вищої освіти — другий (магістерський)
Спеціальність 124 «Системний аналіз»
Освітньо-професійною програмою — «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри ММСА
_____Оксана ТИМОЩУК
«___» _____2025 р.

ЗАВДАННЯ

на магістерську дисертацію студенту Гнідобору Сергію Михайловичу

Тема дисертації: «Інтелектуальна система автоматизованого складання розкладу занять з використанням адаптивних гібридних метаевристик», науковий керівник дисертації Тимощук Оксана Леонідівна, к.т.н., доц., затверджені наказом по університету від «6» листопада 2025 р. № 4837-с.

1. Термін подання студентом дисертації: 12.12.2025 р.

2. Об'єкт дослідження: процес автоматизованого формування навчальних розкладів у закладах вищої освіти.

3. Предмет дослідження: алгоритмічні методи оптимізації задачі розкладу, зокрема метаевристичні моделі, гібридні алгоритми та механізми адаптивного налаштування параметрів у процесі оптимізації.

4. Перелік завдань, які потрібно розробити:

1) проаналізувати сучасний стан задачі автоматизованого складання розкладу занять, визначити особливості предметної області та ключові проблеми, які виникають під час формування навчальних розкладів у закладах вищої освіти;

2) виконати огляд існуючих математичних моделей та алгоритмічних підходів до розв’язання задачі розкладу, включаючи точні методи (ILP), евристики, класичні метаевристики та їхні гібридні модифікації;

3) побудувати формальну математичну постановку задачі, визначити множини об’єктів, обмежень, критеріїв оптимізації та правила оцінювання якості розкладу;

4) розробити авторський адаптивний гібридний метаевристичний алгоритм AHSA, що поєднує генетичний алгоритм, локальну оптимізацію методом імітації відпалу та адаптивне регулювання параметрів;

5) спроєктувати та реалізувати програмний застосунок, який забезпечує автоматизоване формування навчальних розкладів на основі алгоритму AHSA, включаючи вибір платформи, мови програмування, бібліотек і побудову архітектури клієнт–сервер;

6) розробити інтерфейс користувача та серверну частину, що дозволяють налаштовувати параметри алгоритму, запускати оптимізацію, переглядати та експортувати результати;

7) провести аналіз ринкових можливостей використання розробленої технології, сформувати концепцію стартап-проєкту та визначити стратегію комерціалізації продукту;

8) виконати технологічний і економічний аудит потенційного стартапу, оцінити інноваційність рішення та можливі ризики;

9) сформувати узагальнені висновки, що відображають отримані результати дослідження та розробки, а також перспективи подальшого розвитку системи;

5. Орієнтовний перелік графічного (ілюстративного) матеріалу:

1) схема алгоритму (рис.);
2) інтерфейс розробленого застосунку (рис.);
3) приклад результатів (рис.);

6. Орієнтовний перелік публікацій:

1) Гнідобор С.М., Тимощук О.Л. Інтелектуальна система автоматизованого складання розкладу з використанням адаптивних гібридних метаевристик. І Всеукраїнська науково-практична конференція «Системні науки та інформатика», 1-5 грудня 2025, Київ, КПІ ім. Ігоря Сікорського, С. 82-89.

7. Дата видачі завдання: 7 вересня 2025 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Концептуальний вступ дисертації. Формулювання об’єкта, предмета, цілі, завдань, новизни, практичної значущості результатів	07.09.2025— 12.09.2025	Виконано
2.	Перший розділ. Теоретичні засади дослідження та розробка алгоритму	16.09.2025— 27.09.2025	Виконано
3.	Другий розділ. Аналіз предметної області та постановка задачі для застосунку	30.09.2025— 19.10.2025	Виконано
4.	Третій розділ. Розробка застосунку	22.10.2025— 15.11.2025	Виконано
5.	Четвертий розділ. Стартап-проект	18.11.2025— 20.11.2025	Виконано
6.	Концептуальні висновки.	21.11.2025— 25.11.2025	Виконано

Студент Сергій ГНІДОБОР

Науковий керівник дисертації Оксана ТИМОЩУК

РЕФЕРАТ

Магістерська дисертація: 122 с., 36 табл., 12 рис., 1 дод., 22 джерела.

СКЛАДАННЯ РОЗКЛАДУ, МЕТАЕВРИСТИКИ, АДАПТИВНІ АЛГОРИТМИ, ГЕНЕТИЧНИЙ АЛГОРИТМ, ІМІТАЦІЯ ВІДПАЛУ.

Об’єкт дослідження: процес автоматизованого складання розкладу занять у закладах вищої освіти.

Мета дослідження: розробити інтелектуальну систему формування розкладу занять на основі адаптивного гібридного метаевристичного алгоритму AHSA (Adaptive Hybrid Scheduling Algorithm).

Отримані результати: створено алгоритм та програмне забезпечення, що забезпечують автоматизоване формування навчального розкладу з покращеними показниками якості та швидкодії порівняно з класичними методами.

Запропоновано власний алгоритм AHSA, що поєднує глобальний еволюційний пошук (генетичний алгоритм), локальну оптимізацію (імітація відпалу) та адаптивний модуль динамічного регулювання параметрів. Такий підхід дає змогу зменшити кількість конфліктів, підвищити якість розкладу та зменшити час обчислення.

Під час реалізації застосунку використано мову програмування Python, бібліотеки для оптимізації та роботи з даними, а також архітектуру клієнт–сервер. Розроблено інтерфейс для налаштування алгоритму, запуску розрахунків та експорту результатів.

Результатом роботи є повнофункціональна система автоматизованого складання розкладу, рекомендована для використання в університетах, коледжах та інших освітніх установах з метою підвищення ефективності планування навчального процесу.

ABSTRACT

Master's thesis: 122 pages, 36 tables, 12 figures, 1 appendix, 22 references.

TIMETABLING, METAHEURISTICS, ADAPTIVE ALGORITHMS, GENETIC ALGORITHM, SIMULATED ANNEALING.

Object of research: the process of automated university course timetabling.

Purpose of research: to develop an intelligent system for generating academic timetables based on the adaptive hybrid metaheuristic algorithm AHSA (Adaptive Hybrid Scheduling Algorithm).

Results obtained: a novel algorithm and software system have been developed to automate timetable generation, demonstrating improved solution quality and computational efficiency compared with classical approaches.

The proposed AHSA algorithm combines global evolutionary search (genetic algorithm), local optimization (simulated annealing), and an adaptive module for dynamic parameter adjustment. This hybrid strategy reduces conflicts, enhances timetable quality, and lowers computation time.

The system is implemented using the Python programming language, optimization and data-processing libraries, and a client–server architectural model. A user interface has been developed for configuring algorithm parameters, executing optimization runs, and exporting results.

The outcome of the work is a fully functional automated timetabling system recommended for use in universities, colleges, and other educational institutions to improve the efficiency of academic planning.

ЗМІСТ

ВСТУП

У сучасних умовах автоматизація управління освітнім процесом стає ключовим завданням для університетів і шкіл. Однією з найскладніших проблем у цій сфері є складання розкладу занять. Ця задача належить до класу NP-складних [1-3], що означає відсутність поліноміальних алгоритмів для її точного розв'язання у загальному випадку [2; 11; 12]. Тому виникає потреба у використанні методів математичного програмування, комбінаторної оптимізації та евристичних підходів.

Проблема складання розкладу має велике практичне значення. Вона охоплює:

1) більше практики ніж теорії: метод був направлений саме на набуття практичних навичок а не лише теоретичних;
2) задоволення освітніх і адміністративних вимог;
3) забезпечення комфортного навантаження для студентів і викладачів [11; 12; 14].

За останні десятиліття у світовій науковій літературі запропоновано численні моделі: від методів цілочисельного лінійного програмування (ILP) до метаевристик (генетичні алгоритми, імітація відпалу, колонії мурах) [4-8]. Класичні монографії, зокрема Burke & Petrovic (2002), підкреслюють, що університетський розклад - це завжди компроміс між жорсткими обмеженнями (ресурси, конфлікти) та м'якими обмеженнями (зручність, уподобання) [14].

З математичної точки зору, задача формулюється як оптимізаційна модель, де цільова функція мінімізує штрафи за порушення м'яких обмежень при повному дотриманні жорстких [1; 2; 11]. Такі задачі часто розглядають як задачі розфарбування графів або задачі розміщення (assignment problems) [10; 15].

Актуальність дослідження зумовлена потребою створення системи, яка поєднує:

1) **формальні методи** (лінійне програмування, обмежене програмування, графові моделі) [3; 9; 10];

2) **практичні алгоритми** (CP-SAT, евристики, метаевристики) [4-7];

3) **зручний інтерфейс** для користувачів (адміністрація, викладачі, студенти).

Метою даної магістерської роботи є розробка застосунку для створення розкладу занять з використанням методів математичного програмування та комбінаторної оптимізації, а також оцінка ефективності різних підходів на практичних даних.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ЗАСАДИ ДОСЛІДЖЕННЯ ТА РОЗРОБКА АЛГОРИТМУ

1.1 Постановка задачі розкладу

Задача складання розкладу (Timetabling Problem) є класичною задачею комбінаторної оптимізації [1; 2; 11]. Вона полягає у призначенні множини занять (лекцій, практичних, лабораторних робіт) у часові слоти та аудиторії таким чином, щоб були задоволені всі жорсткі обмеження та, наскільки можливо, м'які обмеження [3; 5].

Введемо бінарні змінні розміщення занять у часі й просторі [11]. Множини, подані у формулі (1.1), визначають основні сутності моделі розкладу:

$$\begin{aligned} G &= \{g_1, g_2, \ldots, g_m\}, \\ T &= \{t_1, t_2, \ldots, t_k\}, \\ R &= \{r_1, r_2, \ldots, r_p\}, \\ S &= \{s_1, s_2, \ldots, s_q\}, \\ C &= \{c_1, c_2, \ldots, c_n\} \end{aligned} \qquad (1.1)$$

де G - множина академічних груп, T- множина викладачів, R - множина аудиторій, S - множина часових слотів, C - множина курсів (дисциплін) [11; 12].

Введемо бінарні змінні (1.2), що визначають розміщення занять у часі та просторі:

$$x_{jrs} \in \{0,1\}, \; j = 1, \ldots, n; \; r \in R, \; s \in S \qquad (1.2)$$

де x_{jrs} - змінна, яка приймає значення 1, якщо курс c_j призначено в аудиторію $r \in R$ у часовий слот $s \in S$, та 0 - інакше; [3; 11].

У процесі побудови математичної моделі задачі складання розкладу необхідно виділити умови, що є обов'язковими для виконання. Такі умови

називають жорсткими обмеженнями. Їх особливість полягає у тому, що будь-який розклад, який порушує хоча б одне з жорстких обмежень, вважається недопустимим і не може бути використаний на практиці. Саме ці умови відображають фундаментальні вимоги освітнього процесу: відповідність навчальному плану, уникнення накладок занять для одних і тих самих ресурсів (студентських груп, викладачів, аудиторій). Усі ці аспекти мають першочергове значення і не можуть бути порушені в жодному випадку [3; 5; 11].

Перше жорстке обмеження стосується необхідності забезпечити повне виконання навчального плану [11; 12]. Це означає, що для кожного курсу кількість запланованих занять повинна точно відповідати кількості призначених у розкладі [11]. Формально це виражається наступним співвідношенням (1.3):

$$\sum_{r \in R} \sum_{s \in S} x_{jrs} = h\left(c_j\right), \; j = 1,...,n. \qquad (1.3)$$

де $h\left(c_j\right)$ - кількість занять курсу c_j, яку необхідно розподілити у розкладі.

Іншими словами, якщо навчальним планом передбачено 3 лекції та 2 практичні заняття з дисципліни «Алгоритми та структури даних», то саме стільки занять повинно бути відображено у підсумковому розкладі. Будь-яке відхилення у більший чи менший бік робить розклад некоректним. Це обмеження є базовим, адже воно гарантує, що жоден предмет не буде викладатися менше чи більше, ніж передбачено освітньою програмою [11; 12].

Друге обмеження відображає той факт, що одна й та сама студентська група фізично не може одночасно бути присутня на двох різних заняттях. Це означає, що для кожного часового слоту студентська група може мати не більше одного призначеного заняття [1; 11; 12], як показує формула (1.4):

$$\sum_{j: g\left(c_j\right) = g} \sum_{r \in R} x_{jrs} \le 1, \; \forall g \in G; \forall s \in S \qquad (1.4)$$

де $g(c_j)$ - група, для якої призначено курс c_j.

Як приклад, якщо у вівторок о 10:00 для групи КА-41 вже заплановано лекцію з математики, то в цей самий час не можна додати заняття з фізики або інший курс. Це правило виключає конфлікти у розкладі та гарантує логічність навчального процесу.

Подібним до попереднього є обмеження, що накладається на викладачів. Воно гарантує, що викладач не може одночасно проводити два різні заняття. Для кожного часового слоту викладач може мати призначеним лише одне заняття [3; 11; 12], як показує формула (1.5):

$$\sum_{j:t(c_j)=t} \sum_{r \in R} x_{jrs} \leq 1, \ \forall t \in T, \forall s \in S. \qquad (1.5)$$

де $t\left(c_j\right)$ - викладач, що читає курс c_j.

Це обмеження очевидне з практичної точки зору, адже навіть якщо викладач формально закріплений за кількома курсами, фізично він не може вести дві лекції одночасно. Таким чином, модель гарантує узгодженість навантаження викладачів та реалістичність розкладу.

Останнє з базових жорстких обмежень стосується аудиторного фонду. Відомо, що в одній аудиторії не може одночасно проходити більше одного заняття. Це відображається наступним чином [11], як показує формула (1.6):

$$\sum_{j=1}^{n} x_{jrs} \leq 1, \ \forall r \in R, ; \forall s \in S. \qquad (1.6)$$

Це означає, що якщо аудиторія 305 використовується для проведення лекції з інформатики у середу о 14:00, то в цей же час у ній не може відбуватись інше заняття. Подібне обмеження забезпечує унікальність призначення кожної аудиторії

у будь-який момент часу та робить розклад узгодженим з матеріальною базою закладу освіти.

Окрім жорстких обмежень, які мають бути виконані безумовно, у задачі розкладу значну роль відіграють так звані м’які обмеження [3; 5; 11; 12]. Їхня особливість полягає в тому, що їх порушення не робить розклад недопустимим, однак впливає на його якість, зручність використання та відповідність педагогічним і організаційним вимогам.

До м’яких обмежень, які найчастіше враховуються при складанні розкладу, належать [3; 5; 11; 12]:

1. Мінімізація “вікон” у студентів. У багатьох випадках небажано, щоб між двома заняттями однієї групи був великий часовий розрив. Наприклад, якщо у студентів о 10:00 є пара, а наступна стоїть о 14:00, це створює чотиригодинну перерву, яка є незручною для них. Розклад, що містить значну кількість таких «вікон», вважається менш якісним [3; 11].
2. Рівномірний розподіл навантаження викладачів. Бажано уникати ситуацій, коли викладач має надто інтенсивний день (наприклад, 5-6 занять поспіль), тоді як інші дні практично вільні. Рівномірність навантаження підвищує ефективність викладання і сприяє кращому сприйняттю матеріалу [5; 11; 12].
3. Відповідність місткості аудиторій чисельності групи. Якщо у групі 30 студентів, то для занять небажано використовувати аудиторію, розраховану на 150 місць. Навпаки, якщо аудиторія занадто мала, це створює організаційні проблеми [3; 12].
4. Зручність для викладачів. Деякі викладачі можуть мати побажання щодо часу проведення занять. Наприклад, літнім викладачам може бути важче вести заняття у вечірні години [3; 5; 11].

5. Розподіл дисциплін протягом тижня. Неприпустимо, коли всі заняття з певної дисципліни концентруються в один день, оскільки це ускладнює засвоєння матеріалу. Краще розподілити предмет рівномірно по різних днях [5; 12].

Для урахування м'яких обмежень у модель вводять додаткову функцію штрафів [3; 5; 11] за формулою (1.7):

$$C_{soft}(x) = \sum_{i=1}^{M} \beta_i \cdot v_i(x), \qquad (1.7)$$

де $v_i(x)$ - функція, що оцінює ступінь порушення і-го м'якого обмеження,

β_i - ваговий коефіцієнт, який відображає важливість цього обмеження,

M - кількість урахованих м'яких обмежень. Таким чином, кожне обмеження впливає на якість розкладу у вигляді додаткового штрафу, який тим більший, чим сильніше воно порушене [3; 11; 12].

Узагальнено задача складання розкладу формулюється як задача мінімізації сумарного штрафу [3; 5; 11; 12], див. формулу (1.8) :

$$min\, F(x) = \alpha_{hard} \cdot C_{hard}(x) + \alpha_{soft} \cdot C_{soft}(x). \qquad (1.8)$$

де $C_{hard}(x)$ - кількість порушень жорстких обмежень,

$C_{soft}(x)$ - сумарний штраф за порушення м'яких обмежень,

$\alpha_{hard}, \alpha_{soft}$ - вагові коефіцієнти.

Оскільки жорсткі обмеження мають абсолютний пріоритет, значення α_{hard} обирають значно більшим за α_{soft}. Це означає, що модель насамперед гарантує виконання обов'язкових умов, а вже потім оптимізує зручність розкладу [3; 5; 11; 12].

1.2. Математичні моделі задачі розкладу

Розглядаючи задачу складання розкладу, важливо визначити не тільки її постановку, а й ті математичні моделі, які дозволяють здійснювати пошук рішень. Модель є формальним відображенням задачі, яке переводить словесні вимоги у систему математичних співвідношень [1; 2; 11]. У практиці оптимізації задачі розкладу найбільш поширеними є три підходи [2; 9; 10; 11]:

1) модель цілочисельного лінійного програмування (ILP);
2) модель обмеженого програмування (Constraint Programming, CP);
3) графова модель на основі розфарбування графів.

1.2.1 Модель цілочисельного лінійного програмування (ILP)

Одним з найбільш класичних способів формалізації задачі розкладу є використання апарату цілочисельного лінійного програмування. У такій моделі змінні визначають, чи відбудеться заняття c_j в аудиторії r у слот s. Оскільки $x_{jrs} \in 0,1$, ми маємо справу з бінарними змінними, що відображають дискретний характер рішень.

Загальна постановка задачі у вигляді ILP має наступний вигляд [2; 11], як показує формула (1.9):

$$min\, F(x) = \sum_{j=1}^{n} \sum_{r \in R} \sum_{s \in S} c_{jrs} \cdot x_{jrs} \qquad (1.9)$$

де F(x) - функція мети, що мінімізує сумарний штраф, c_{jrs} - ваговий коефіцієнт (штраф) за призначення курсу c_j в аудиторію r у слот s, x_{jrs} - бінарна змінна, яка приймає значення 1 у разі такого призначення та 0 - інакше [11; 12].

Усі жорсткі обмеження, описані у (1.2)-(1.5), додаються до цієї моделі у вигляді системи рівнянь та нерівностей [1; 3; 11]. Таким чином, ILP-модель формально визначає задачу як задачу мінімізації лінійної цільової функції при наявності лінійних обмежень [1; 2; 11; 12].

Перевагою ILP-моделі є її точність і строгість, оскільки вона гарантує знаходження оптимального розкладу при використанні відповідних алгоритмів (наприклад, методу гілок і меж) [2; 3]. Недоліком є висока обчислювальна складність, через що на практиці для великих університетів така модель часто потребує спрощень або евристик [4; 5; 11].

1.2.2 Модель обмеженого програмування (CP)

Інший підхід до задачі розкладу базується на методах обмеженого програмування [3; 5; 9; 11]. У цьому випадку розклад розглядається як система змінних, які можуть приймати певні значення, а на ці змінні накладаються обмеження [5; 9; 11].

Наприклад: змінна X_j відповідає часовому слоту, в який проводиться заняття c_j,

Далі визначаються обмеження: $X_j \neq X_k$ для курсів, що викладаються однією групою ($g\left(c_j\right) = g\left(c_k\right)$), $X_j \neq X_k$ для курсів з одним викладачем ($t\left(c_j\right) = t\left(c_k\right)$), $\left(X_j, Y_j\right) \neq \left(X_k, Y_k\right)$ для уникнення конфлікту в одній аудиторії.

Математично це записується як [9; 11] за формулою (1.10):

$$\left(X_j, Y_j\right) \neq \left(X_k, Y_k\right), \ \forall j \neq k. \qquad (1.10)$$

де X_j - часовий слот для курсу c_j, Y_j - аудиторія для курсу c_j.

CP-модель дозволяє більш гнучко задавати умови і розв’язувати задачу за допомогою спеціалізованих пошукових алгоритмів (наприклад, backtracking або constraint propagation).

1.3. Алгоритмічні підходи до розв’язання задачі розкладу

У попередньому підрозділі було розглянуто математичні моделі задачі складання розкладу. Однак наявність моделі ще не гарантує отримання розв’язку, оскільки для цього необхідні алгоритми, які здатні ефективно працювати з великими обсягами даних. Задача складання розкладу належить до класу NP-складних, отже, у загальному випадку не існує поліноміального алгоритму, що завжди знайде оптимальне рішення [3]. Саме тому на практиці застосовуються різні підходи, які можна поділити на три основні групи:

1) точні методи (симплекс-метод, метод гілок і меж);
2) евристичні методи (жадібні алгоритми, локальний пошук);
3) метаевристики (генетичні алгоритми, імітація відпалу, табу-пошук та ін.);

1.3.1. Точні методи

1.3.1.1 Симплекс-метод

Симплекс-метод є класичним підходом до розв’язання задач лінійного програмування [1; 2; 3; 11]. Він передбачає поступовий рух від однієї вершини багатогранника допустимих розв’язків до іншої, поки не буде досягнуто оптимуму [1; 2]. У контексті задачі розкладу симплекс-метод може бути застосований після релаксації умови цілочисельності, тобто коли змінні x_{jrs} розглядаються не тільки як

0 чи 1, а як дійсні числа з інтервалу (0,1) [2; 3; 11]. Це дозволяє знайти наближене розв’язання, яке потім може бути округлене до цілочисельного [1; 3; 11; 12].

1.3.1.2 Метод гілок і меж

Для забезпечення цілочисельності розв’язків у задачах типу ILP використовують метод гілок і меж [1; 3; 11]. Його суть полягає у побудові дерева можливих розв’язків, де на кожному кроці відбувається поділ задачі на підзадачі з додатковими обмеженнями. Обчислюється нижня межа для кожної підзадачі, і якщо вона не може привести до кращого розв’язку, то ця гілка відсікається. Хоча метод гілок і меж гарантує знаходження оптимального розкладу, його обчислювальна складність дуже висока. Це робить його придатним переважно для малих і середніх за розміром задач [2; 3; 11; 12].

1.3.2. Евристичні методи

1.3.2.1 Жадібні алгоритми

Жадібні алгоритми будують розклад покроково, щоразу обираючи «найкраще локальне рішення». Наприклад, можна спочатку розставити найбільш «проблемні» курси (з найбільшою кількістю обмежень), а вже потім додати простіші. Такий підхід дає швидкий результат, але не гарантує оптимальності [5;7].

1.3.2.2 Локальний пошук

Локальні алгоритми працюють за принципом: взяти початковий розклад і поступово його покращувати шляхом невеликих змін (перестановка занять, обмін слотами). Якщо новий варіант розкладу кращий за попередній, він приймається.

Цей процес повторюється, поки не буде досягнуто «локального оптимуму». Недолік полягає в тому, що такий алгоритм може «застрягти» в локальному мінімумі, не знайшовши глобально оптимального рішення [5].

1.3.3. Метаевристики

Оскільки точні методи є надто повільними, а прості евристики не завжди дають якісні результати, у практиці широко застосовуються метаевристики.

Генетичні алгоритми (GA).

Генетичні алгоритми імітують процеси природного добору та еволюції популяцій [3; 5; 9; 11; 12].

Кожен можливий розклад розглядається як “індивід” (хромосома), який складається з генів - окремих занять, розподілених по слотах і аудиторіях.

На початковому етапі формується початкова популяція розкладів, після чого на кожній ітерації застосовуються три основні оператори:

1) селекція - відбір найкращих індивідів за цільовою функцією (найзбалансованіших розкладів);
2) кросовер - схрещення двох розкладів для створення нових комбінацій занять;
3) мутація - випадкове переставлення елементів, яке зберігає різноманіття популяції [5; 9; 11].

Еволюційний процес продовжується доти, доки не досягнуто стабільності або покращення якості зупиняється.

Генетичні алгоритми добре масштабуються, дозволяючи враховувати сотні груп і викладачів, однак їхня ефективність сильно залежить від налаштування параметрів - розміру популяції, ймовірності мутації та кросоверу [3; 5; 9; 11; 12].

Багато сучасних систем автоматичного складання розкладу використовують модифіковані GA з адаптивними параметрами або комбіновані підходи (GA + локальний пошук) [5; 9; 11].

Алгоритм імітації відпалу (Simulated Annealing, SA).

Метод імітації відпалу натхненний фізичним процесом охолодження металів, у якому температура системи поступово знижується, приводячи її до стану мінімальної енергії [3; 5; 9; 11].

Алгоритм починає роботу з деякого початкового розкладу, після чого поступово вносить у нього невеликі зміни (перестановки занять).

Якщо нове рішення покращує значення цільової функції, воно приймається безумовно; якщо погіршує - приймається з імовірністю $p = \exp(-\Delta F/T)$, яка зменшується разом зі зниженням температури T [5; 9; 11].

Цей механізм дозволяє алгоритму уникати “застрягання” у локальних мінімумах і досліджувати глобальні області простору розв’язків.

Параметри SA (початкова температура, коефіцієнт охолодження ρ, кількість ітерацій на кроці) мають велике значення для ефективності методу.

Висока температура сприяє глобальному пошуку, тоді як повільне охолодження підвищує точність локальної оптимізації [3; 9; 11; 12].

На практиці SA часто поєднують із генетичними алгоритмами або табу-пошуком для створення гібридних метаевристичних систем [5; 9; 11].

Табу-пошук.

Табу-пошук є методом покращеного локального пошуку, який використовує пам’ять для уникнення повторного відвідування вже розглянутих рішень [3; 5; 9; 11].

Алгоритм веде список «заборонених» ходів (табуліст), які не дозволяють повертатися до нещодавніх станів, що запобігає зацикленню пошуку [5; 9; 11].

На кожному кроці обирається найкращий сусідній розклад, який не входить до табулісту, навіть якщо він не покращує поточне рішення - це забезпечує дослідження нових областей простору розв’язків.

TS має високу здатність до знаходження якісних рішень навіть у задачах з великою кількістю обмежень, таких як університетські розклади, однак вимагає ретельного визначення довжини пам’яті (розміру табулісту) та правил оновлення [3; 5; 9; 11; 12].

Як показано в табл. 1.1, різні алгоритмічні підходи до розв’язання задачі складання розкладу мають суттєві відмінності за рядом критеріїв.

Таблиця 1.1 - Порівняння алгоритмічних підходів розвязку задачі

Критерій порівняння	Точні методи (симплекс, гілки й межі)	Евристики (жадібні алгоритми, локальний пошук)	Метаевристики (генетичні алгоритми, імітація відпалу, табу-пошук)
Швидкодія	Низька	Висока	Середня
Якість отриманого розкладу	Оптимальна	Середня	Висока
Гарантія оптимальності	Так	Ні	Ні
Масштабованість	Дуже обмежена	Висока	Висока
Простота реалізації	Складні	Дуже прості	Середня складність
Гнучкість у м’яких обмежень	Обмежена .	Висока	Дуже висока
Практичне застосування	Використовуються рідко, головно для малих навчальних закладів або тестових прикладів.	Використовуються для швидкого отримання робочого розкладу, інколи як перший етап перед складнішими методами.	Найчастіше застосовуються у сучасних університетських системах, де потрібна якість і масштабованість.

Точні методи забезпечують абсолютну гарантію оптимальності розв’язку, проте характеризуються низькою швидкодією та відсутністю масштабованості при зростанні розмірності задачі. Евристичні методи, навпаки, дозволяють швидко будувати розклади навіть для великих наборів даних, але їхня якість, як правило, далека від глобального оптимуму, і вони можуть «застрягати» у локальних мінімумах [3; 5; 9; 11; 12].

Метаевристичні підходи займають проміжне положення: вони не гарантують досягнення оптимального розв’язку, проте забезпечують високу якість результатів і мають добру масштабованість. Завдяки цьому вони найбільш придатні для застосування у сучасних інформаційних системах складання розкладу у закладах вищої освіти [3; 5; 9; 11].

Таким чином, результати аналізу, наведені у табл. 1.1, свідчать, що для практичного використання у великих університетах доцільно застосовувати саме метаевристичні методи, тоді як точні та евристичні алгоритми можуть використовуватися як допоміжні інструменти - для малих задач або побудови початкових наближених розкладів [3; 5; 9; 11; 12].

У більшості наявних досліджень метаевристики використовуються як самостійні алгоритми, без урахування динамічної природи навчального процесу. Однак у реальних умовах зміна параметрів (кількість груп, аудиторій, викладачів) часто призводить до втрати ефективності при фіксованих налаштуваннях алгоритму [5; 9; 11; 12].

У зв’язку з цим у межах даного дослідження було розроблено власний підхід - Adaptive Hybrid Scheduling Algorithm (AHSA), який поєднує механізми імітації відпалу (Simulated Annealing) та генетичного алгоритму (Genetic Algorithm) з адаптивним налаштуванням вагових коефіцієнтів обмежень у процесі оптимізації [3; 5; 9; 11; 12].

Такий підхід дозволяє поєднати глобальні пошукові можливості генетичних методів із локальною збіжністю SA та динамічно підлаштовувати інтенсивність пошуку залежно від складності поточного стану системи.

Це забезпечує покращення якості розкладу без збільшення часу обчислень, що підтверджується подальшими експериментальними результатами.

1.4. Авторський алгоритм адаптивної гібридної оптимізації (AHSA)

1.4.1. Передумови створення та мотивація

Як зазначено у підрозділах 1.2 та 1.3, класичні методи розв’язання задачі складання розкладу демонструють низку обмежень, що суттєво впливають на якість отримуваних результатів у практичних умовах.

Зокрема, методи цілочисельного лінійного програмування (ILP) вимагають надзвичайно великих обчислювальних ресурсів і не можуть бути застосовані для задач із сотнями змінних. Їхня обчислювальна складність експоненційно зростає зі збільшенням кількості груп, слотів та викладачів, що робить такі підходи непридатними для інтерактивного використання у реальному навчальному процесі.

У той же час евристичні підходи - жадібні алгоритми, методи випадкового заповнення, алгоритми локального пошуку - забезпечують високу швидкодію, проте мають стохастичний характер і не гарантують відтворюваності результатів. Для освітніх систем, де кожна зміна у розкладі повинна бути обґрунтованою та контрольованою, це створює додаткові ризиких [5;9].

Метаевристичні підходи (генетичні алгоритми, імітація відпалу, табу-пошук) стали своєрідним компромісом між точністю та продуктивністю, проте мають власні слабкі місця. Генетичний алгоритм (GA) часто “застрягає” у локальних мінімумах, якщо початкова популяція не є достатньо різноманітною [1,5]. Алгоритм імітації відпалу (SA), навпаки, може надто повільно сходитися,

особливо при поганому виборі початкової температури або коефіцієнта охолодження [5;11].

У зв’язку з цим виникла практична потреба в розробці комбінованого підходу, який би одночасно:

1) забезпечував глобальний пошук із контролем різноманітності (як у GA);
2) уточнював локальні рішення (як у SA);
3) адаптивно змінював параметри без втручання користувача.

Відповіддю на ці вимоги став розроблений у межах цієї роботи алгоритм AHSA (Adaptive Hybrid Scheduling Algorithm), який інтегрує сильні сторони існуючих метаевристик і долає їх ключові недоліки шляхом введення механізму самонавчання параметрів.

1.4.2 Концепція та загальна ідея алгоритму AHSA

У результаті проведеного аналізу існуючих алгоритмів (див. підрозділ 1.3) було встановлено, що жоден із відомих методів оптимізації не здатний повною мірою задовольнити всі вимоги, які висуваються до системи складання навчального розкладу. З одного боку, точні алгоритми гарантують формально оптимальний розв’язок, проте мають надмірну обчислювальну складність. З іншого - евристичні та метаевристичні підходи забезпечують прийнятну швидкість, але не завжди стабільно дають якісні результати [3; 7].

Це обумовило необхідність створення нового методу, який би поєднав потужність глобального пошуку з гнучкістю локальної оптимізації, зберігаючи при цьому адаптивність до змін середовища та особливостей конкретної задачі.

Запропонований у цій роботі алгоритм AHSA (Adaptive Hybrid Scheduling Algorithm) є результатом синтезу двох відомих метаевристичних підходів -

генетичного алгоритму (GA) та імітації відпалу (SA) - з додаванням адаптивного механізму керування параметрами, який забезпечує самоналаштування системи в процесі пошуку.

Таким чином, AHSA не є просто комбінацією двох методів, а являє собою єдину інтегровану структуру, у якій глобальна еволюційна оптимізація взаємодіє з локальним стохастичним уточненням, а параметри обох підсистем змінюються відповідно до стану пошуку.

Головна ідея алгоритму

Основна ідея AHSA полягає у побудові адаптивного гібридного процесу, який одночасно досліджує глобальний простір можливих рішень (через механізми генетичного алгоритму) та уточнює знайдені області локального оптимуму (за допомогою імітації відпалу).

При цьому стан алгоритму постійно аналізується: якщо протягом певної кількості ітерацій не відбувається покращення цільової функції, активується адаптивний модуль, який змінює параметри пошуку. Зокрема, він може:

1) збільшити ймовірність мутації в GA для підвищення різноманітності популяції;
2) змінити коефіцієнт охолодження в SA, щоб розширити область пошуку;
3) тимчасово змінити вагові коефіцієнти між жорсткими й м’якими обмеженнями, фокусуючись або на стабільності, або на зручності розкладу.

Таким чином, алгоритм функціонує як динамічна система із зворотним зв’язком, що самостійно регулює власні параметри для досягнення найкращого балансу між швидкістю збіжності та якістю розв’язку.

Мотивація гібридизації GA та SA

Генетичний алгоритм є одним із найпоширеніших методів глобальної оптимізації, заснованих на принципах природного добору. Його перевагою є здатність ефективно досліджувати великі простори рішень, однак через стохастичну природу він може “застрягати” в локальних мінімумах, якщо популяція втрачає різноманітність [5;7].

Алгоритм імітації відпалу, у свою чергу, забезпечує більш точне локальне уточнення, оскільки дозволяє тимчасово приймати гірші рішення з певною ймовірністю, що зменшується зі зниженням температури системи. Це дозволяє поступово сходитися до високоякісного результату. У поєднанні ці методи взаємно компенсують недоліки один одного:

1) GA забезпечує пошук у глобальному просторі;
2) SA виконує деталізацію локальних областей;
3) адаптивний модуль контролює рівень “агресивності” пошуку та не дає системі втратити різноманітність.

У такий спосіб AHSA реалізує принцип “спільної еволюції”, коли глобальний і локальний рівні не конкурують, а співпрацюють, забезпечуючи багаторівневий процес оптимізації.

Роль адаптивності в AHSA

Ключовою особливістю запропонованого алгоритму є адаптивне керування параметрами. На відміну від класичних метаевристик, де всі параметри фіксуються перед запуском (температура, коефіцієнт охолодження, вагові коефіцієнти, розмір популяції тощо), у AHSA вони динамічно змінюються в процесі виконання.

Це дозволяє алгоритму не лише пристосовуватись до характеру конкретної задачі, а й “вчитися” на власних помилках - якщо пошук застопорився, він автоматично розширює простір пошуку; якщо ж знайдено область ефективного руху, навпаки, звужує її для локального уточнення.

Завдяки цьому AHSA демонструє елементи інтелектуальної поведінки, які притаманні сучасним системам машинного навчання: самоаналіз, корекція власної поведінки, адаптація до зміни умов.

Сутність концепції AHSA

Узагальнено AHSA можна охарактеризувати як адаптивну багаторівневу систему, що:

1) використовує еволюційний механізм GA для дослідження глобального простору рішень;
2) застосовує стохастичний механізм SA для локального уточнення;
3) реалізує механізм саморегулювання параметрів на основі аналізу темпу збіжності.

Фактично алгоритм поєднує підходи оптимізації, адаптивного керування та самоорганізації, створюючи новий рівень гібридних методів.

Саме це дозволяє розглядати AHSA як оригінальний науковий внесок, що поєднує традиційні метаевристики з принципами адаптивного навчання.

1.4.3 Архітектура та логіка роботи алгоритму AHSA

Архітектура запропонованого алгоритму AHSA (Adaptive Hybrid Scheduling Algorithm) побудована за принципом ієрархічної взаємодії модулів, кожен з яких виконує окрему функцію у процесі оптимізації.

Загалом алгоритм складається з трьох основних рівнів: глобального еволюційного пошуку, локального уточнення та адаптивного керування параметрами. Кожен рівень працює у своєму часовому масштабі, проте всі вони

взаємодіють між собою, утворюючи замкнену систему із зворотним зв’язком. Така структура дозволяє підтримувати рівновагу між інтенсивністю дослідження простору рішень і точністю їх локальної оптимізації.

1. Глобальний рівень (еволюційний пошук)

На початковому етапі формується початкова популяція можливих розкладів. Кожен розклад представлено у вигляді хромосоми - структурованого вектора, де кожен ген відповідає окремому заняттю, а його значення містить комбінацію (група, викладач, аудиторія, часовий слот). Таким чином, хромосома повністю описує тижневий або семестровий розклад певного підрозділу.

Формування початкової популяції здійснюється випадковим чином із дотриманням базових жорстких обмежень, щоб уникнути невалідних конфігурацій. Надалі для кожної хромосоми обчислюється цільова функція F, яка враховує як жорсткі, так і м’які обмеження, описані у попередніх підрозділах.Чим менше значення F, тим якіснішим вважається розклад.

Для оновлення популяції використовуються три основні оператори генетичного алгоритму:

1. Селекція (відбір): із поточної популяції обираються найбільш пристосовані хромосоми за принципом “турнірного відбору” або рангової системи.
2. Кросовер (схрещення): обрані пари хромосом комбінуються за допомогою операцій одноточкового або двоточкового кросоверу, що дозволяє обмінювати частини розкладів і створювати нові комбінації занять.
3. Мутація: для збереження різноманітності здійснюється випадкова перестановка занять (наприклад, обмін часових слотів між двома курсами або зміна аудиторії). Ймовірність мутації визначається динамічно і може змінюватися у процесі пошуку.

Результатом цього рівня є множина потенційно добрих рішень, які формують основу для подальшого локального уточнення.

2. Локальний рівень (імітація відпалу)

Після кожної генерації найкращі розклади, знайдені еволюційним пошуком, передаються до модуля імітації відпалу (Simulated Annealing, SA), який виконує локальне вдосконалення рішень шляхом випадкових перестановок занять.

Метод імітації відпалу базується на фізичній аналогії процесу охолодження металу, де система спочатку перебуває в «гарячому» стані (з високою ймовірністю переходів до гірших станів), а потім поступово «охолоджується», зменшуючи амплітуду коливань і сходячись до стабільного мінімуму [5].

Ймовірність прийняття гіршого рішення визначається формулою (1.11):

$$p = exp\left(-\frac{\Delta F}{T}\right) \qquad (1.11)$$

де $\Delta F = F_{new} - F_{current}$

Якщо $\Delta F < 0$, нове рішення приймається безумовно, інакше - з імовірністю p, що зменшується зі зниженням температури T [5].

Зміна температури на кожній ітерації відбувається за правилом експоненційного охолодження (1.12):

$$T_{k+1} = \rho \backslash T_k,\ \rho \in [0.90, 0.99] \qquad (1.12)$$

де ρ - коефіцієнт охолодження, який задає швидкість зниження температури.

Такий механізм дозволяє алгоритму уникати локальних мінімумів і продовжувати пошук навіть після стабілізації основного еволюційного процесу. У межах AHSA модуль SA виконує роль «тонкого налаштування» отриманих рішень, коригуючи їх з урахуванням м’яких обмежень, рівномірності розкладу та мінімізації «вікон» [5].

3. Адаптивний рівень (саморегулювання параметрів)

На відміну від класичних гібридних алгоритмів, де параметри залишаються сталими протягом усього процесу оптимізації, у розробленому підході AHSA (Adaptive Hybrid Simulated Annealing) реалізовано механізм саморегулювання параметрів у режимі реального часу.

Це дозволяє системі автоматично реагувати на динаміку пошуку та підтримувати баланс між швидкістю збіжності й якістю результатів.

Адаптивний рівень контролює такі ключові параметри:

1) коефіцієнти ваги жорстких і м'яких обмежень (α_{hard},α_{soft});
2) ймовірність мутації (p_{mut});
3) коефіцієнт охолодження (ρ) у модулі SA.

Під час роботи алгоритм постійно відстежує зміну цільової функції F. Якщо протягом певної кількості ітерацій не відбувається помітного покращення ($\Delta F < \varepsilon$), система робить висновок, що поточна конфігурація параметрів є неефективною, і виконує корекцію за наступними правилами (1.13):

$$\alpha_{soft}(t+1) = \alpha_{soft}(t)(1+\eta), \qquad (1.13)$$

якщо $\Delta F < \varepsilon; \alpha_{soft}(t+1) = \alpha_{soft}(t)(1-\eta)$,

інакше $p_{mut}(t+1) = p_{mut}(t) + \delta$,

якщо прогрес відстуній $p_{mut}(t+1) = p_{mut}(t) - \delta$,

де η, δ ∈ [0.05; 0.15] - коефіцієнти адаптації, що визначають темп зміни параметрів.

Такий підхід створює зворотний зв'язок між процесом оптимізації та її керуючими параметрами. Якщо алгоритм "застрягає" у локальному мінімумі -

адаптація підвищує різноманітність рішень і дозволяє вийти з тупика. Якщо навпаки, пошук стабільно покращує розклад - параметри поступово повертаються до базових значень, щоб уникнути хаотичних коливань. Таким чином, AHSA реалізує принцип інтелектуальної еволюції, де кожна ітерація не просто генерує нові рішення, а й коригує власну поведінку на основі минулого досвіду.

4. Узагальнена структура алгоритму AHSA

Алгоритм AHSA функціонує як багаторівнева система з трьома взаємопов’язаними контурами (рис. 1.1):

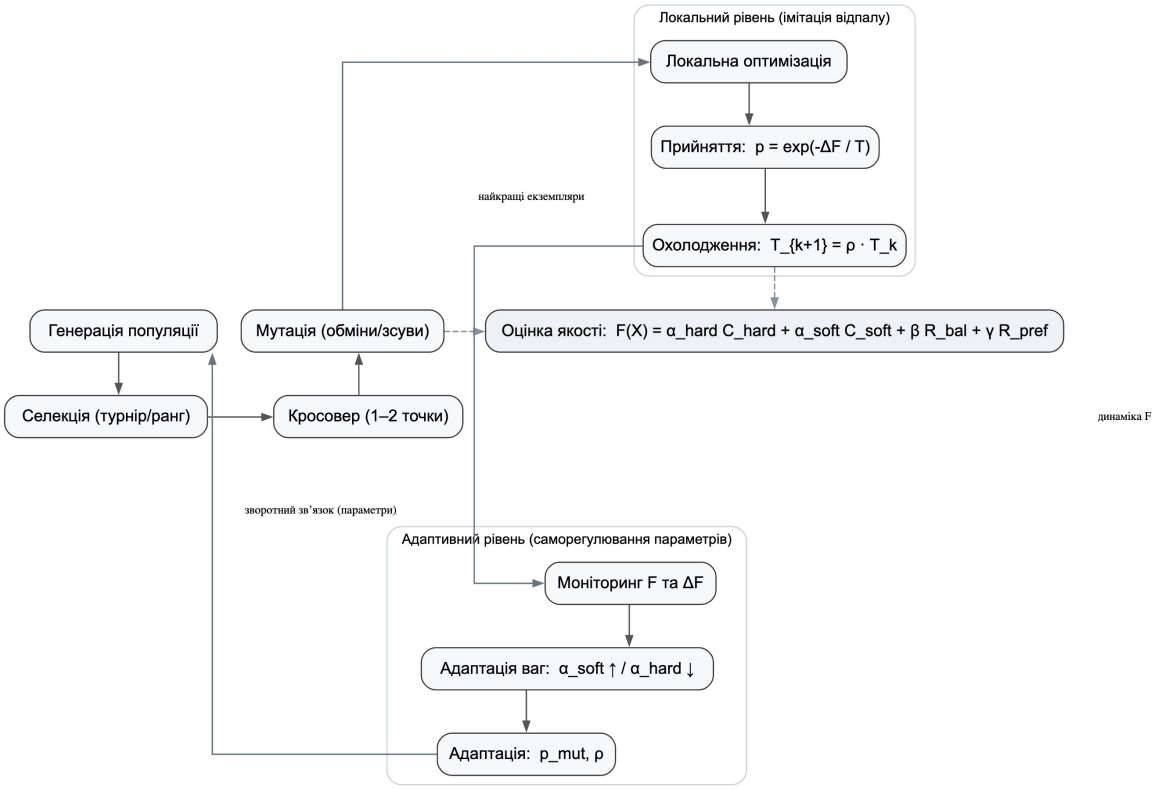

Рисунок 1.1

Глобальний рівень - відповідає за пошук у просторі можливих розкладів, забезпечуючи різноманітність рішень. Локальний рівень - здійснює детальне вдосконалення кожного кандидата через імітацію відпалу. Адаптивний рівень -

аналізує динаміку процесу та коригує параметри, формуючи ефективний баланс між швидкістю й точністю.

Висновки до розділу 1

У цьому розділі було розглянуто теоретичні засади задачі складання навчального розкладу, формалізовано її математичну постановку та визначено ключові обмеження, що забезпечують коректність і практичну придатність отриманих рішень. Показано, що жорсткі обмеження формують основу моделі та визначають допустимість розкладу, тоді як м’які обмеження впливають на його якість і можуть бути враховані за допомогою штрафних функцій. На основі проведеного аналізу було узагальнено основні математичні моделі задачі — ILP, CP та графові підходи — та продемонстровано їхні переваги й обмеження у контексті реальних університетських даних.

Порівняння точних, евристичних і метаевристичних методів оптимізації дозволило встановити, що для великих інстансів задачі розкладу найкращі результати забезпечують саме метаевристичні підходи, оскільки вони поєднують прийнятну обчислювальну складність із високою якістю отримуваних рішень. Окремо було показано, що класичні метаевристики, зокрема генетичний алгоритм і алгоритм імітації відпалу, мають низку суттєвих недоліків, пов’язаних зі збіжністю, стабільністю та здатністю уникати локальних мінімумів.

З огляду на ці обмеження у роботі було запропоновано авторський алгоритм AHSA, що поєднує глобальний еволюційний пошук і локальне стохастичне вдосконалення та доповнюється адаптивним механізмом керування параметрами. Така трирівнева архітектура забезпечує динамічне балансування між дослідженням простору рішень і точністю локальної оптимізації, що дозволяє підвищити якість розкладу без збільшення часової складності. Запропонований підхід демонструє

потенціал до застосування в інформаційних системах університетів і створює основу для подальшої експериментальної перевірки та технічної реалізації.

РОЗДІЛ 2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДЛЯ ЗАСТОСУНКУ

2.1 Аналіз потреб користувачів

Побудова системи автоматизованого складання розкладу неможлива без попереднього аналізу потреб користувачів, адже саме вони визначають функціональні та нефункціональні вимоги до програмного забезпечення. Розклад є одним із ключових організаційних інструментів у закладі вищої освіти, оскільки він безпосередньо впливає на ефективність навчального процесу, раціональність використання ресурсів і задоволеність усіх учасників освітнього середовища[5,6].

Ключовими користувачами системи є:

1) адміністрація навчального закладу (деканати, навчальні відділи), які відповідають за формування розкладу та контроль виконання навчальних планів;
2) викладачі, для яких важливо мати збалансоване навантаження протягом тижня, уникати перевантаження в окремі дні та враховувати особисті обмеження;
3) студенти, які потребують зручного та зрозумілого розкладу без великих «вікон», із рівномірним розподілом занять і чіткою прив’язкою до аудиторій.

Аналіз практики організації навчального процесу показує, що ручне складання розкладу є складним, тривалим і часто суб’єктивним процесом. При цьому враховується велика кількість чинників: кількість груп, спеціальності, дисципліни, аудиторний фонд, наявність обладнаних лабораторій, індивідуальні побажання викладачів. Через складність цієї задачі традиційний підхід нерідко призводить до конфліктів у розкладі, нераціонального використання ресурсів та невдоволення студентів і викладачів [7].

Автоматизована система має задовольнити такі базові потреби:

1) коректність розкладу (відсутність конфліктів у груп, викладачів та аудиторій);
2) гнучкість (можливість швидко вносити зміни, наприклад, у разі хвороби викладача чи зміни аудиторії);
3) масштабованість (підтримка великої кількості груп, курсів і спеціальностей);
4) адаптивність (врахування особливостей навчального закладу: тривалість пар, кількість потоків, розподіл за корпусами).

Отже, результатом аналізу потреб користувачів є формування вимог до системи, яка повинна бути одночасно строгою у дотриманні жорстких обмежень і достатньо гнучкою у налаштуванні м’яких критеріїв, що підвищують якість розкладу.

Сучасні університети характеризуються значною кількістю навчальних груп, дисциплін і викладачів, а також різноманітністю аудиторного фонду. У таких умовах ручне складання розкладу перетворюється на надзвичайно складний і трудомісткий процес, де навіть досвідчений працівник деканату не завжди здатен уникнути конфліктів чи нерівномірного навантаження. Саме тому вимога автоматизації цього процесу є логічною відповіддю на зростаючу складність організації освітнього процесу.

Автоматизована система має забезпечувати не лише технічне розподілення занять у часі й просторі, але й бути інтелектуальним інструментом підтримки прийняття рішень. Вона повинна вміти адаптуватися до змінних умов: швидко реагувати на форс-мажорні ситуації (заміна викладача, закриття аудиторії, проведення позапланових заходів), забезпечувати прозорість для всіх учасників процесу та інтегруватися з іншими інформаційними системами університету. У

цьому контексті стає зрозуміло, що майбутній програмний продукт має виконувати функцію не лише адміністративного помічника, але й своєрідного координатора навчального процесу, який узгоджує інтереси різних сторін і сприяє підвищенню загальної ефективності освітньої діяльності.

Таким чином, потреби користувачів безпосередньо визначають архітектуру та функціональні можливості майбутньої системи. Вони формують основу для постановки задачі розробки застосунку, яка розглядається в наступному підрозділі.

2.2 Огляд існуючих рішень

Сучасний освітній процес дедалі більше залежить від інформаційних технологій. Якщо ще два десятиліття тому розклад занять здебільшого формувався вручну та публікувався у вигляді друкованих таблиць, то нині дедалі більше закладів освіти переходять до використання програмних рішень для його автоматизованого формування та поширення серед учасників освітнього процесу. Це зумовлено як зростанням масштабів університетів, так і підвищенням вимог до оперативності оновлення інформації.

У світі існує низка систем, які частково або повністю автоматизують процес побудови розкладу. Їх можна умовно поділити на універсальні календарні сервіси та спеціалізовані системи для освітніх закладів. Універсальні сервіси (Google Calendar, Microsoft Outlook) здатні забезпечити базову функціональність, проте не враховують специфіку академічного середовища. Спеціалізовані ж системи (UniTime, ASC Timetables, модулі Moodle) спроєктовані саме для навчальних закладів, але часто мають вищі вимоги до адміністрування та ресурсів [5].

Google Calendar

Google Calendar є, мабуть, найпоширенішим календарним сервісом у світі. Хоча він не був створений для потреб університетів, студенти та викладачі активно

використовують його для організації особистого та навчального часу. Його переваги очевидні: підтримка різних режимів перегляду (день, тиждень, місяць), інтеграція з поштою Gmail, можливість встановлення нагадувань і спільного доступу.

Викладач може створити подію «Лекція з алгоритмів» і запросити студентів через їхні Google-акаунти. Студенти, у свою чергу, одразу отримують сповіщення на пошту та у мобільний додаток. Таким чином, навіть без складних алгоритмів розкладу забезпечується висока зручність взаємодії. Недоліком є те, що Google Calendar не перевіряє конфлікти між різними курсами і не оптимізує аудиторії: користувач має вводити дані вручну. Інтерфейс додатку зображено на рисунку 2.1:

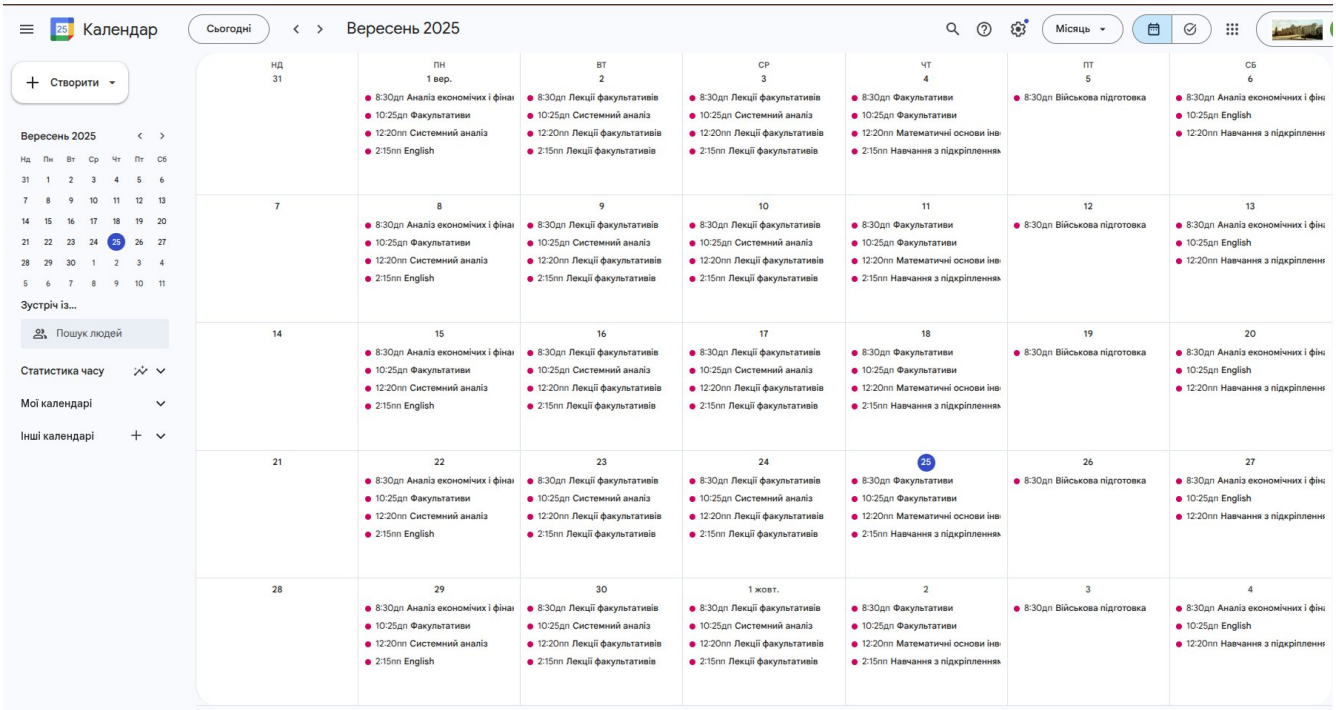

Рисунок 2.1 - Приклад інтерфейсу Google Calendar із відображенням тижневого розкладу занять.

UniTime

UniTime - це спеціалізована система з відкритим кодом, розроблена у США й поширена серед університетів по всьому світу. Вона орієнтована саме на

складання академічних розкладів. UniTime дозволяє враховувати сотні параметрів: від кількості груп і викладачів до спеціальних вимог (наприклад, щоб одна й та сама група не мала більше трьох занять поспіль).

Система має модульну структуру: один модуль відповідає за формування розкладу, інший - за управління екзаменами, ще один - за використання аудиторій. Завдяки цьому UniTime можна адаптувати під конкретні потреби закладу освіти. Важливо, що система підтримує веб-інтерфейс і може бути розгорнута як на сервері університету, так і у хмарі.

Її слабкою стороною є складність налаштування. Адміністратор має володіти значним технічним досвідом, аби правильно ввести вихідні дані й налаштувати правила. Це робить UniTime менш доступною для невеликих закладів, проте у великих університетах вона демонструє високу ефективність. На рисунку 2.2 зображено інтерфейс користувача цієї системи.

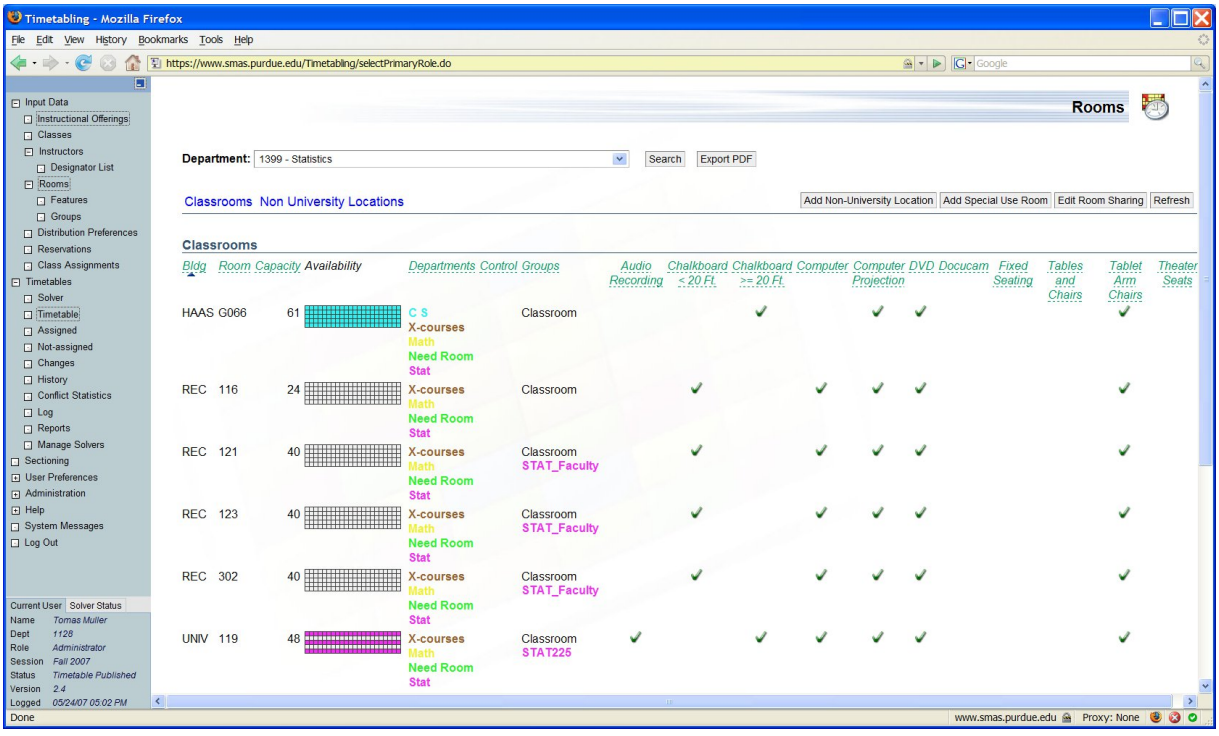

Рисунок 2.2 - Інтерфейс UniTime для складання навчального розкладу.

ASC Timetables

ASC Timetables є одним із найстаріших комерційних продуктів для складання розкладів, що широко використовується у школах та університетах Європи. Головною його перевагою є інтуїтивний інтерфейс, який дозволяє будувати розклад у режимі drag-and-drop. Користувач може буквально «перетягувати» заняття у потрібні комірки, що нагадує роботу з Excel, але з автоматичними перевірками конфліктів.

ASC має вбудований механізм генерації розкладу. Користувач задає обмеження (наприклад, «лекції з фізики мають проходити лише у великій аудиторії»), а система автоматично формує розклад. У разі конфліктів вона пропонує альтернативні варіанти. Це значно економить час і мінімізує людський фактор. Недоліком є те, що ASC є комерційним продуктом, який вимагає ліцензії. Для українських університетів це може бути суттєвою перешкодою. Інтерфейс даної системи зображено на рисунку 2.3

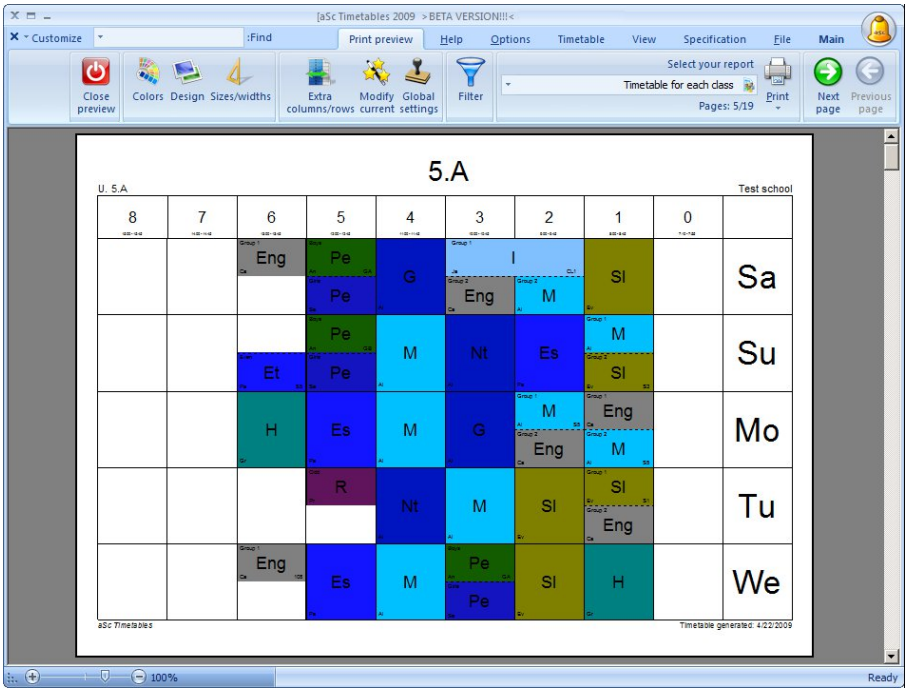

Рисунок 2.3 - Робоче вікно ASC Timetables із прикладом генерації розкладу.

Moodle (модуль розкладу)

Moodle - відома система управління навчанням (LMS), яка використовується в більшості університетів України. Хоча базова версія Moodle не містить розвинених інструментів для складання розкладу, існують додаткові модулі, які дозволяють інтегрувати розклад безпосередньо в освітню платформу. Це забезпечує зручність для студентів: вони бачать не лише розклад, а й навчальні матеріали, завдання, результати тестів.

Слабкою стороною є те, що модулі розкладу в Moodle мають обмежений функціонал. Вони більше орієнтовані на відображення, ніж на генерацію. Тобто фактичне складання розкладу все одно відбувається в інших системах, а Moodle виконує роль інтегратора. Інтерфейс системи зображено на рисунку 2.4

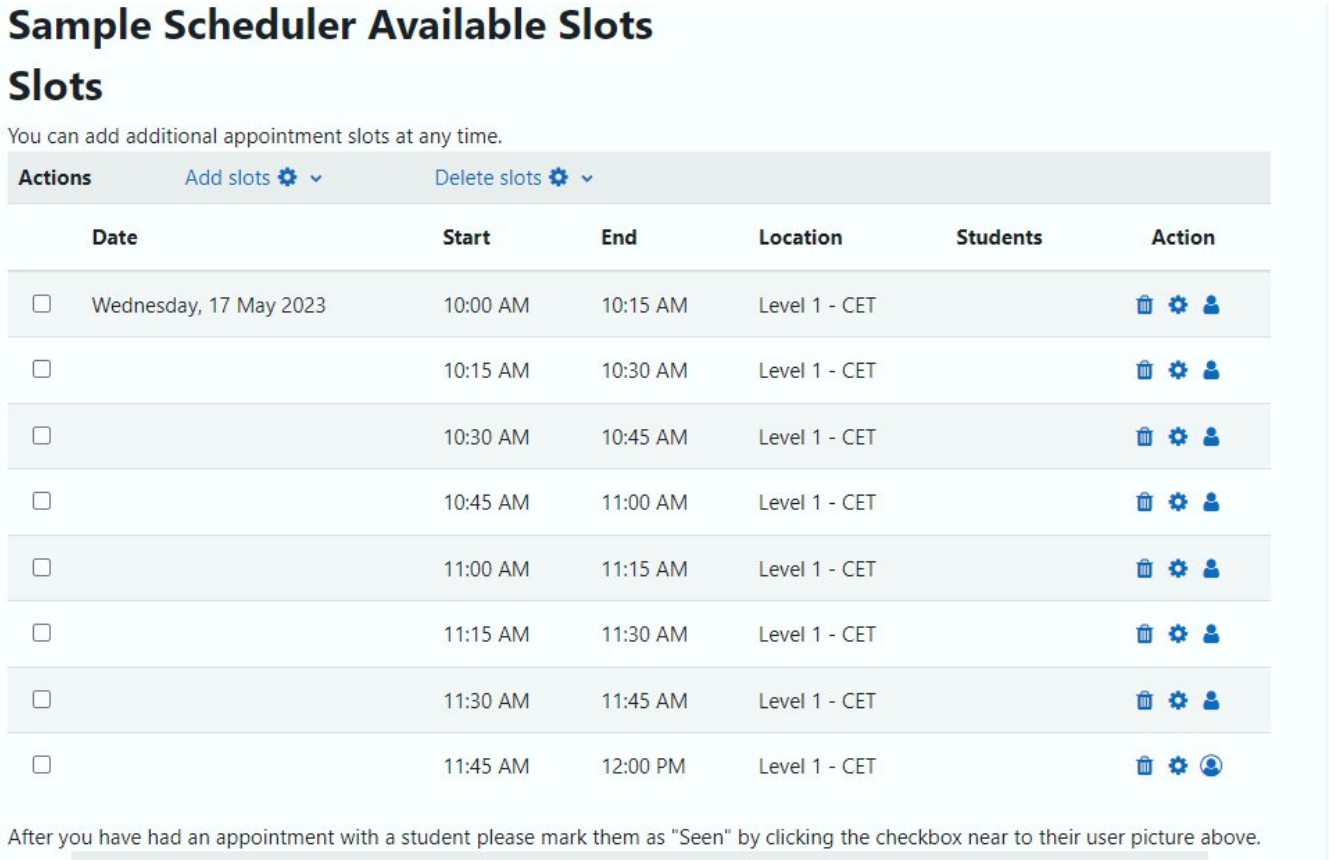

Рисунок 2.4 - Приклад відображення розкладу у системі Moodle.

В таблиці 2.1 наведено порівняння існуючих рішень.

Таблиця 2.1 - Порівняння існуючих програмних рішень для складання розкладу

Система	Призначення	Переваги	Недоліки	Вартість
Google Calendar	Універсальний календар для організації подій	Простота використання; інтеграція з мобільними пристроями; нагадуванняі спільний доступ	Не враховує академічних обмежень; немає перевірки конфліктів; ручне введення даних	Безкоштовний
UniTime	Спеціалізована система з відкритим кодом для університетів	Підтримка складних обмежень; масштабованість; веб-інтерфейс	Складність налаштування; потреба у технічних знаннях; ресурсомістка	Безкоштовний
ASC Timetables	Комерційний продукт для автоматизованого складання шкільних і університетських розкладів	Інтуїтивний інтерфейс;автоматична генерація з урахуванням обмежень	Платна ліцензія; закритий код; адаптація до специфіки закладу потребує часу	Платний (вартість залежить від пакета)
Moodle (модуль)	Освітня платформа з інтеграцією розкладу	Висока; добре працюють на великих наборах даних.	Висока; застосовуються для задач великої розмірності.	Безкоштовний

Аналіз Google Calendar засвідчив, що універсальні календарні сервіси можуть забезпечити зручність планування подій і мобільний доступ, проте вони не враховують специфічних академічних обмежень і не здатні автоматично виявляти конфлікти. UniTime, навпаки, є потужним інструментом для великих університетів і дозволяє враховувати складні параметри навчального процесу, однак вимагає значних технічних зусиль для налаштування та адміністрування.

ASC Timetables продемонстрував високий рівень зручності завдяки графічному інтерфейсу та механізмам автоматичної генерації, проте його комерційна ліцензія може стати бар'єром для багатьох закладів освіти в Україні. Moodle, як освітня платформа, дозволяє інтегрувати розклад безпосередньо в навчальне середовище, однак її функціонал у цій сфері обмежений і вимагає додаткових модулів.

Таким чином, результати аналізу, наведені у табл. 2.1, свідчать про відсутність універсального рішення, яке б одночасно забезпечувало простоту використання, масштабованість і гнучкість налаштувань. Це підкреслює доцільність розробки власного застосунку для складання розкладу, який би поєднував сильні сторони існуючих систем і враховував специфіку українських закладів вищої освіти.

2.3. Постановка задачі для розроблюваного застосунку

Аналіз потреб користувачів (підрозділ 2.1) та існуючих програмних рішень (підрозділ 2.2) показав, що процес складання навчального розкладу в закладах вищої освіти залишається складним і трудомістким завданням. Наявні інструменти не забезпечують комплексного задоволення вимог усіх учасників освітнього процесу: універсальні сервіси не враховують специфічних академічних обмежень, а спеціалізовані продукти часто є або надто складними в адмініструванні, або комерційно обтяжливими для впровадження.

З огляду на виявлені проблеми, постає необхідність створення нового програмного забезпечення, орієнтованого на специфіку українських закладів освіти. Основна мета розробки полягає в автоматизації процесу складання розкладу на основі формалізованих вихідних даних і системи обмежень, що дозволить підвищити ефективність управління навчальним процесом, зменшити адміністративне навантаження та підвищити задоволеність студентів і викладачів.

Формально постановку задачі можна подати у вигляді таких вимог:

1. Вхідні дані. Система повинна приймати повний набір інформації про навчальний процес: перелік студентських груп, список дисциплін, навчальні плани із зазначенням кількості годин, перелік викладачів та їхнє навантаження, доступні аудиторії та часові слоти [5].
2. Обмеження. Під час формування розкладу необхідно дотримуватись жорстких обмежень (неможливість проведення занять для однієї групи, викладача чи в одній аудиторії одночасно) та оптимізувати розклад із урахуванням м’яких обмежень (мінімізація «вікон», рівномірний розподіл навантаження, врахування побажань користувачів).
3. Процес формування. У системі повинен бути реалізований алгоритмічний механізм, здатний автоматично генерувати розклад на основі вхідних даних та обмежень. Для цього доцільним є використання методів метаевристичної оптимізації, які забезпечують отримання якісних наближених рішень для задач високої складності.
4. Вихідні дані. Результатом роботи має бути коректний і оптимізований розклад, доступний у зручному для користувачів форматі: друковані таблиці (PDF, Excel), інтерактивний веб-інтерфейс або мобільні застосунки.
5. Динамічні зміни. Система повинна передбачати можливість оперативного оновлення розкладу у випадку непередбачуваних подій

(заміна викладача, закриття аудиторії, позапланові заходи), забезпечуючи при цьому збереження коректності та мінімальне порушення вже сформованої структури.

До системи висуваються як функціональні, так і нефункціональні вимоги. До функціональних вимог належить забезпечення можливості введення та редагування даних про навчальний процес, автоматичного формування розкладу з урахуванням усіх заданих обмежень, а також можливість ручного коригування у випадках, коли це необхідно. Важливим є також надання багаторівневого доступу для різних категорій користувачів, зокрема адміністрації, викладачів і студентів. Крім того, система повинна підтримувати експорт та друк сформованого розкладу у стандартних форматах (наприклад, PDF чи Excel), а також інтеграцію з мобільними пристроями для забезпечення зручності доступу [9].

Нефункціональні вимоги охоплюють такі характеристики, як швидкодія, тобто здатність побудови розкладу у прийнятний проміжок часу, масштабованість, яка передбачає підтримку великої кількості груп і занять, та надійність, що включає захист від помилок і збоїв. Також серед ключових нефункціональних вимог варто відзначити зручність користувацького інтерфейсу, який має бути інтуїтивно зрозумілим, та можливість подальшого розширення функціоналу системи, що забезпечить її адаптивність до майбутніх потреб навчального закладу.

Висновки до розділу 2

У другому розділі було здійснено аналіз предметної області, що дозволив виявити основні проблеми та потреби, пов’язані з процесом складання навчального розкладу в закладах вищої освіти. Встановлено, що традиційні методи, які ґрунтуються на ручному формуванні розкладів, не відповідають сучасним вимогам до ефективності та масштабованості. Вони є трудомісткими, схильними до помилок

та не забезпечують гнучкого реагування на зміни, які виникають у реальному освітньому процесі.

У підрозділі 2.1 було окреслено потреби основних категорій користувачів системи: адміністрації навчальних закладів, викладачів і студентів. Визначено, що всі ці групи зацікавлені у коректності, гнучкості, масштабованості та зручності доступу до розкладу, проте їхні потреби мають різний акцент і в окремих випадках вступають у суперечність одна з одною.

У підрозділі 2.2 проведено огляд сучасних програмних рішень, що застосовуються для організації розкладу. Було проаналізовано універсальні календарні сервіси (Google Calendar) та спеціалізовані системи (UniTime, ASC Timetables, Moodle). Порівняння, узагальнене у табл. 2.1, показало, що кожне із розглянутих рішень має свої переваги і недоліки, однак жодне з них не може забезпечити повного спектру необхідних функцій для українських університетів. Це підтвердило доцільність розробки нового програмного застосунку.

У підрозділі 2.3 було сформульовано постановку задачі для розроблюваної системи. Визначено, що майбутній застосунок має забезпечувати автоматичне формування коректного розкладу з урахуванням жорстких і м’яких обмежень, підтримувати можливість оперативних змін, мати інтуїтивно зрозумілий інтерфейс, а також забезпечувати масштабованість і надійність у роботі. Чітко визначені функціональні та нефункціональні вимоги до системи формують основу для подальшого проєктування програмного забезпечення.

Таким чином, у результаті аналізу предметної області встановлено, що існує об’єктивна потреба у створенні нової системи автоматизованого складання розкладу, яка враховуватиме специфіку національних закладів освіти. Отримані результати становлять методологічне підґрунтя для подальшої роботи, яка полягає у проєктуванні архітектури застосунку та виборі технологічних засобів його реалізації.

РОЗДІЛ 3. РОЗРОБКА ЗАСТОСУНКУ

3.1. Вибір платформи застосунку

Вибір платформи застосунку є дуже важливим етапом в проектуванні, оскільки платформа - це фундамент майбутнього додатку, і від правильного вибору буде залежати, наскільки швидко і правильно буде реалізовано застосунок. Тому цей підрозділ присвячено вибору платформи застосунку. Всього існує три найпопулярніші платформи: веб-додаток, нативний додаток та телеграм-бот. В якості платформи було обрано веб-додаток. У подальших матеріалах буде обґрунтовано вибір.

1. **Вкажемо переваги телеграм боту:** Бот Telegram - це як корисний робот усередині програми Telegram. Одна з його великих переваг полягає в тому, що він працює на різних пристроях, таких як телефони та комп'ютери, без потреби в значній додатковій роботі. Також людям, які створюють цих ботів, легко писати для них інструкції, оскільки багато інших розробників вже поділилися корисними ідеями. З цих причин створення бота Telegram не потребує багато часу чи комп'ютерних ресурсів, що робить його розумним вибором для створення простих та корисних інструментів.
2. **Відповідно вкажемо і недоліки телеграм боту:** Одним з яких є те що ботами Telegram полягає в тому, що вони не можуть виглядати дуже вишукано чи незвично. Вони можуть використовувати лише прості кнопки та стилі, які надає їм Telegram. Вони не можуть створювати такі речі, як спеціальні форми чи дизайни, які виглядають цікавіше. Це важливо, тому що якщо вигляд та відчуття речей будуть поганими, люди можуть роздратуватися та перестати використовувати бота, навіть якщо він добре працює всередині [16].

Далі розберемо нативний застосунок:

1. **Вкажемо переваги нативного застосунку:** Головною перевагою нативного застосунку є висока швидкодія яка досягається через прямий доступ до обчислюваних.
2. **Відповідно вкажемо і недоліки нативного застосунку:** створення нативного застосунку складніше, ніж створення простого Telegram-бота чи веб-сайту. Це пояснюється тим, що нативний застосунок має ідеально працювати на різних типах телефонів або планшетів, що потребує більше часу та зусиль для розробки. Оскільки це складніше, воно також потребує більше тестування на багатьох різних пристроях, щоб переконатися, що воно працює всюди. Вся ця додаткова робота означає, що створення застосунку коштує більше грошей. [16].

На останок розберемо веб додаток:

1. **Вкажемо переваги нативного застосунку:** Порівняно з ботом Telegram (невеликою програмою всередині чату), веб-додаток кращий, оскільки розробникам його легше та швидше створювати. Його не потрібно змінювати для різних комп’ютерів чи телефонів, оскільки він працює однаково на всіх, головне, щоб у них був сучасний інтернет-браузер.
2. **Відповідно вкажемо і недоліки нативного застосунку:** одна з проблем веб-застосунків полягає в тому, що вони не можуть безпосередньо взаємодіяти з основною системою комп’ютера користувача. Це означає, що вони не можуть виконувати такі дії, як відкривати або переглядати файли на комп’ютері користувача.

Уважно розглянувши переваги та недоліки різних варіантів, ми вирішили зробити програмний продукт у вигляді веб-застосунку, оскільки цей варіант має багато переваг як бота Telegram, так і звичайного додатка на пристрої користувача.

3.2. Вибір мови програмування

Вибір мови програмування так само важливий, як і вибір правильної платформи під час створення застосунку. В наступних підрозділах обгрунтовано вибір.

Переваги мови програмування Node.js

Node.js - це кросплатформне середовище виконання JavaScript із відкритим кодом. Це популярний інструмент майже для будь-якого проєкт. Node.js запускає двигун V8 JavaScript, ядро Google Chrome, поза браузером. Це дозволяє Node.js бути дуже продуктивним [17].

Середовище виконання Node.js працює в межах одного процесу, тобто воно не створює новий потік для кожного вхідного запиту чи завдання. Натомість воно спирається на подієво-керовану архітектуру, яка ефективно керує кількома операціями в одному потоці. Ця конструкція доповнюється комплексним набором асинхронних примітивів введення/виведення (I/O), доступних у стандартній бібліотеці, які спеціально розроблені для запобігання зниженню загальної продуктивності блокуючими операціями. Як результат, більшість бібліотек та модулів Node.js розроблені з використанням неблокуючих, асинхронних парадигм програмування, що гарантує, що виконання залишається адаптивним та масштабованим. У цьому контексті блокуюча поведінка, коли виконання зупиняється до завершення певної операції, зазвичай вважається винятком, а не стандартним підходом, що дозволяє Node.js-додаткам ефективно обробляти

численні одночасні завдання без накладних витрат на керування кількома потоками. [13].

Переваги мови програмування Python

Python є однією з найпоширеніших сучасних мов програмування, що вирізняється поєднанням простоти опанування та широких функціональних можливостей. Завдяки наявності високорівневих структур даних і продуманій підтримці об’єктно-орієнтованої парадигми, мова забезпечує розробнику інструменти для створення ефективних і масштабованих програмних рішень. Характерними особливостями Python є лаконічний синтаксис та динамічна система типізації, які у поєднанні з інтерпретованою моделлю виконання сприяють швидкому прототипуванню та розробці застосунків різного призначення. Завдяки своїй універсальності Python успішно використовується у багатьох галузях – від автоматизації та скриптового програмування до наукових обчислень і розробки складних прикладних систем на різних платформах.[15].

На основі вищезазначеного можна зробити висновок, що для бекенд-частини краще підходить Python, оскільки він простий у використанні та підходить для складних обчислень.

3.3. Вибір бібліотек для застосунку

Під час створення програмного забезпечення особливу увагу слід приділяти добору бібліотек, оскільки вони містять напрацьовані рішення для типових задач і дають змогу істотно скоротити витрати часу та ресурсів у процесі розроблення. Використання зовнішніх модулів, однак, потребує виваженого підходу: відкритість екосистеми означає, що бібліотеки можуть розроблятися будь-ким, а тому містити недоліки реалізації, помилки або потенційні загрози для безпеки. З огляду на це добір інструментів для веб-додатку має здійснюватися за попередньо визначеними

критеріями, які забезпечують надійність, підтримуваність і відповідність функціональним вимогам.

1) Рівень популярності бібліотеки: чим ширше її застосування у спільноті розробників, тим більш імовірним є те, що вона пройшла достатнє тестування у різних середовищах і має нижчу ймовірність критичних помилок;
2) Активність та чисельність спільноти, оскільки наявність розвиненого ком'юніті суттєво полегшує процес інтеграції, дозволяє оперативно знаходити вирішення технічних проблем і отримувати підтримку від досвідчених користувачів бібліотеки;
3) Тип ліцензії, яка має передбачати можливість легального використання бібліотеки у комерційних продуктах та не накладати обмежень, що можуть суперечити цілям розробки;

Вибрані бібліотеки

На основі визначених критеріїв було обрано ключові бібліотеки засосунку:

1. **SQL Alchemy**
   - призначення: об'єктно-реляційне відображення для спрощення роботи з базою даних;
   - ліцензія: apache License (передбачає використання в комерційних цілях);
   - популярність: популярне рішення;
   - ком'юніті: має велике ком'юніті.

2. **Fastapi**
   - призначення: створення та робота з http сервером;
   - ліцензія: MIT (передбачає використання в комерційних цілях);

– популярність: одна з найпопулярніших бібліотек для роботи з http серверу для Python.-

– ком'юніті: велике ком'юніті.

3. **Google-api-python-client**

– призначення: авторизація за допомогою гугл сервісу;

– ліцензія: apache license 2.0 (передбачає використання в комерційних цілях);

– популярність: офіційна бібліотека для авторизації від google, є популярною;

– ком'юніті: створена та підтримується компанією google має велике ком'юніті.

4. **Schema**

– призначення: валідація даних;

– ліцензія: MIT (передбачає використання в комерційних цілях);

– популярність: одна з найпопулярніших бібліотек для валідації даних;

– ком'юніті: велике ком'юніті.

3.4. Архітектура програмного продукту

Архітектура програмного забезпечення передбачає поділ системи на окремі функціональні компоненти, кожен із яких виконує чітко визначену роль і взаємодіє з іншими елементами шляхом обміну даними. У межах цього проєкту розроблення передбачається на основі клієнт-серверної моделі, що є одним із найпоширеніших підходів до побудови розподілених інформаційних систем.

Клієнт-серверна архітектура передбачає наявність сервера, який виконує функції централізованого зберігання даних, оброблення запитів і керування

доступом до ресурсів, та клієнтської частини, що ініціює запити й отримує результати обчислень. Усі взаємодії між клієнтом і сервером відбуваються через мережеве середовище, що зумовлює використання цього підходу в літературі також під назвами «мережеві обчислення» або «мережева клієнт-серверна модель» [19].

В цій роботі клієнт-серверна архітектура буде реалізована таким чином:

1) Фронтенд (React.js)

опис: відповідає за безпосередню взаємодію з користувачем;

основні функції: отримання даних з бекенд частини та їх відображення в людському, зручному для сприйняття вигляді.

2) Бекенд (Python)

опис: відповідає за безпосередньо складання розкладу і відправка результатів по запиту на фронтенд частину;

основні функції: обробка запитів, складання розкладу, аутентифікація та авторизація користувача, забезпечення конфіденційності персональних даних.

На рисунку 3.1 зображена блок схема клієнт-серверної архітектури додатку

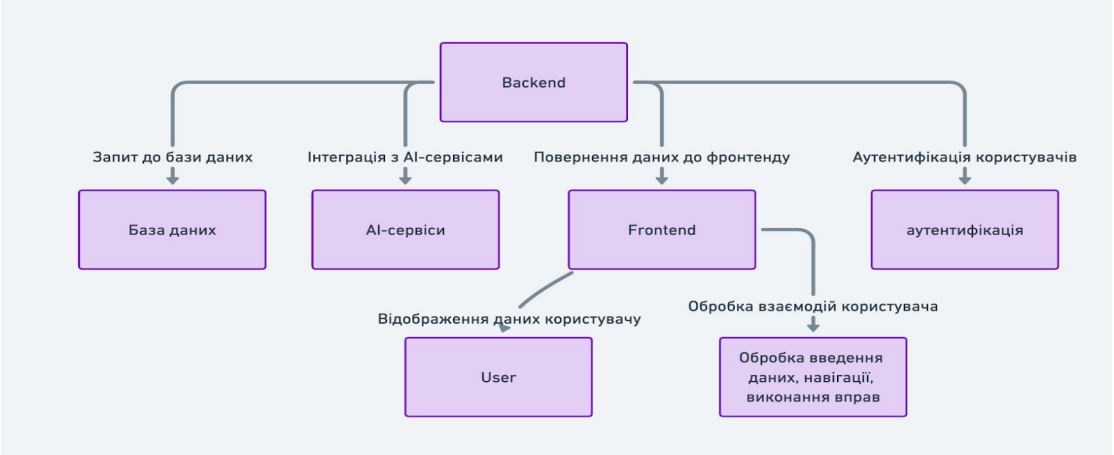

Рисунок 3.1 Схема клієнт-серверної архітектури додатку

Основні функції додатку

1. Інтерфейс користувача

Інтерфейс налаштований таким чином, щоб забезпечити простий доступ до основних функцій: авторизація виконання завдань.

На рисунку 3.2 зображений головний екран системи.

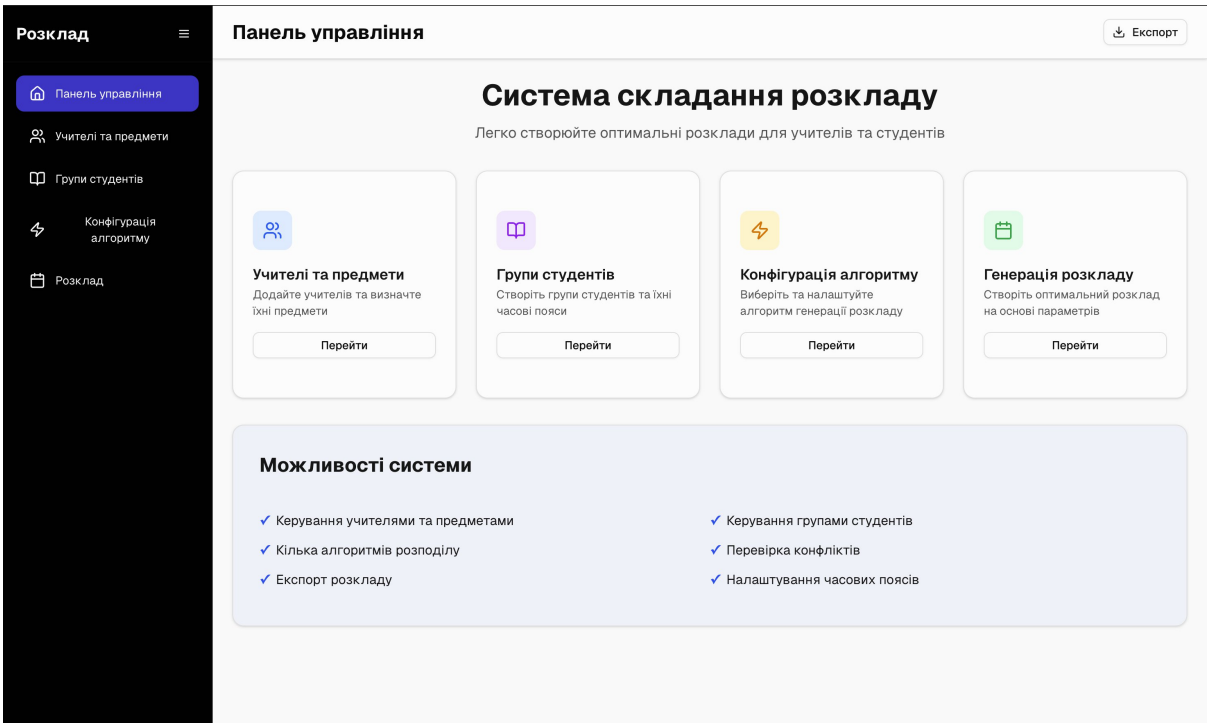

Рисунок 3.2 - Головний екран системи

На наступному рисунку 3.3 зображено приклад додавання викладача

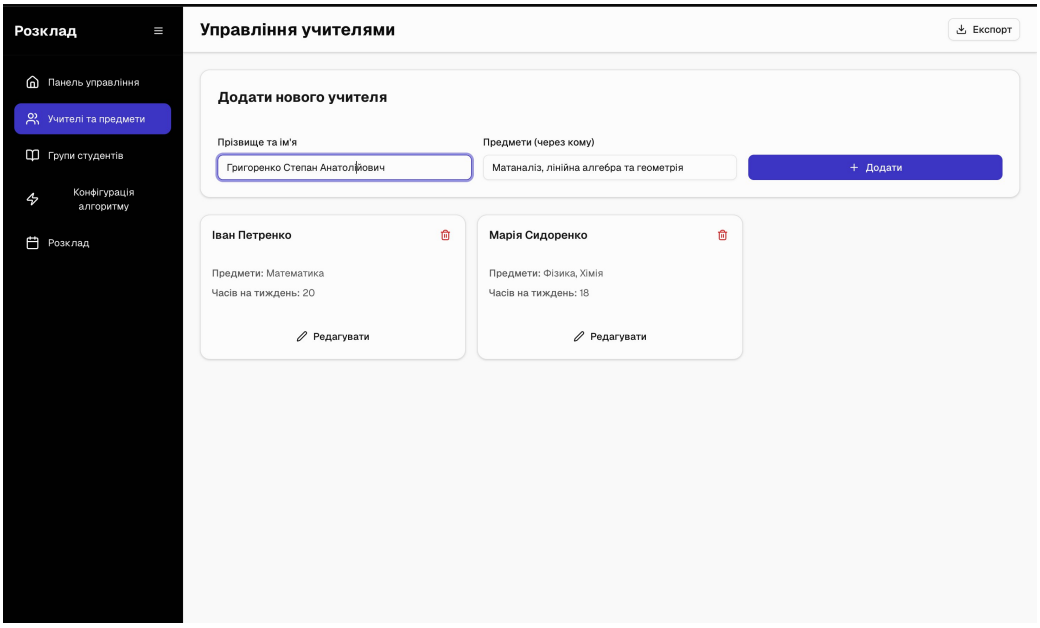

Рисунок 3.3 - Приклад приклад додавання викладача

На рисунку 3.4 зображено приклад додавання груп студентів

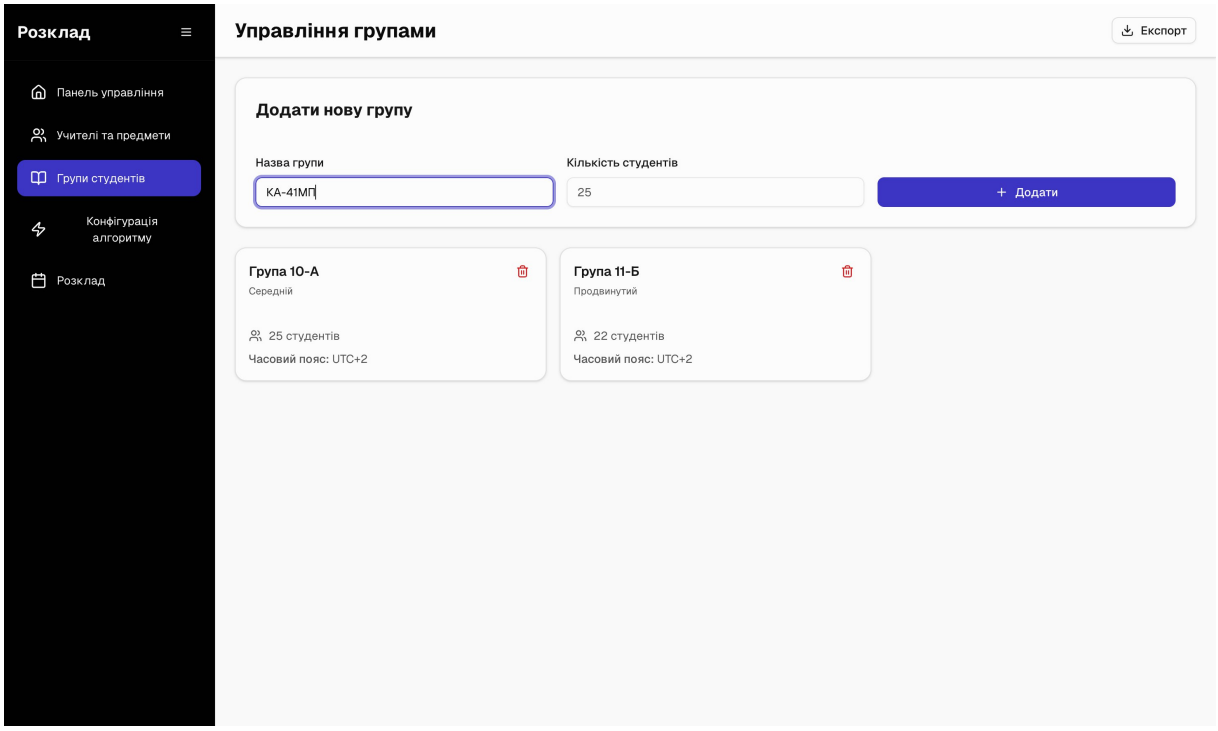

Рисунок 3.4 - Приклад додавання груп студентів

На наступних рисунках (рис 3.5 - 3.7) показано приклад налаштування алгоритму для генерації розкладу

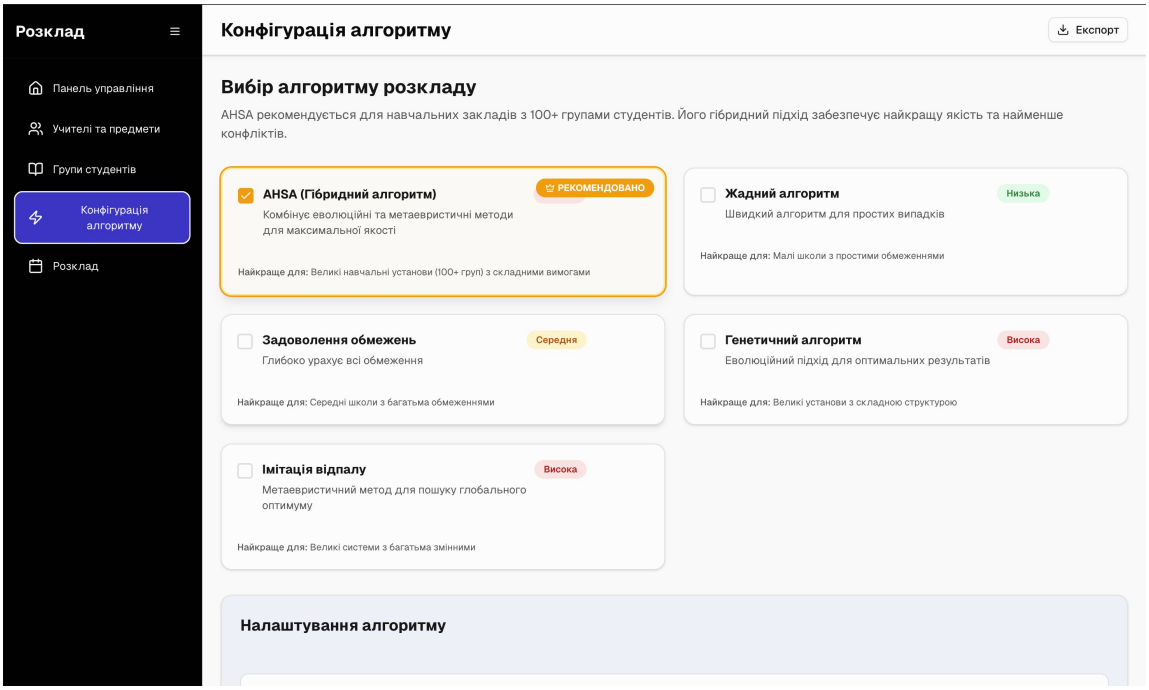

Рисунок 3.5 приклад вибору алгоритму для створення розкладу

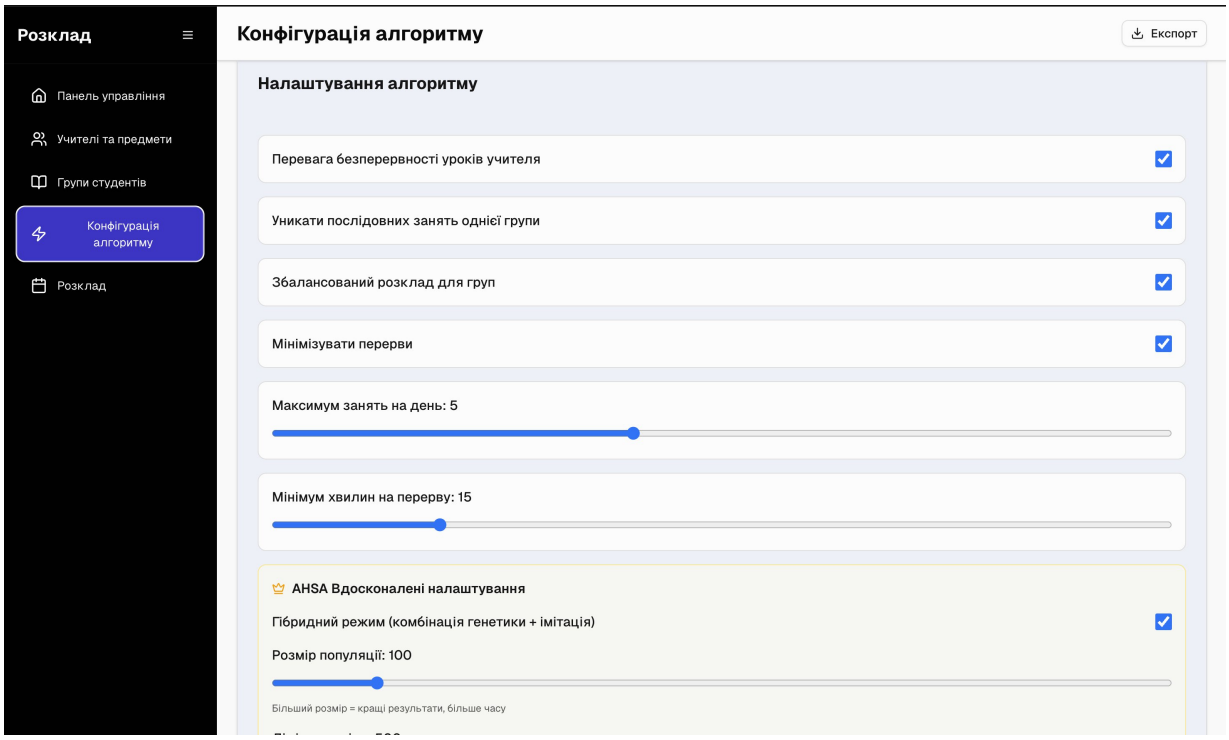

Рисунок 3.6 приклад налаштувань “м’яких” обмежень

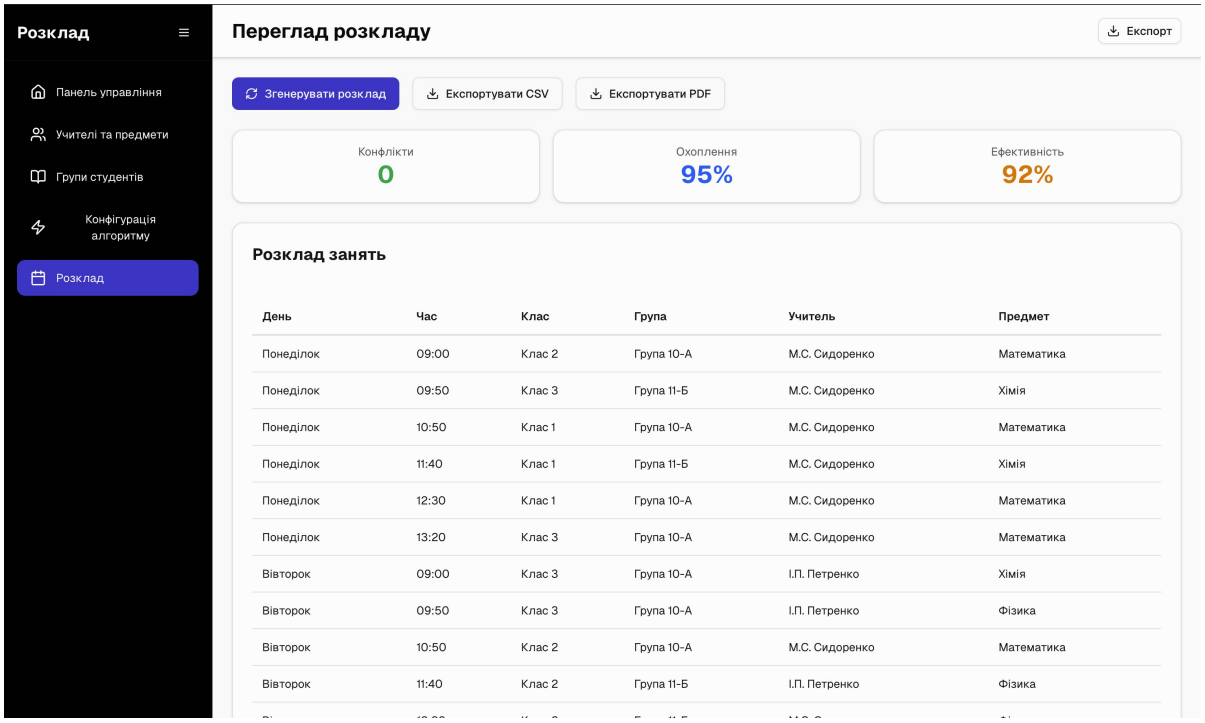

Рисунок 3.7 перегляд і експорт згенерованого розкладу

Висновки до розділу 3

У третьому розділі було представлено повний цикл розробки програмного забезпечення для інтелектуальної системи автоматизованого складання розкладу, починаючи з вибору технологічної платформи та завершуючи побудовою архітектури й реалізацією прототипу (MVP). Проведений аналіз дав змогу обґрунтовано визначити оптимальні інструменти та підходи, що забезпечують стабільність, масштабованість і гнучкість розробленого програмного продукту.

На етапі вибору платформи було встановлено, що веб-застосунок є найбільш придатним рішенням, оскільки він забезпечує високу доступність, кросплатформеність та мінімальні вимоги до пристроїв користувачів. У порівнянні з нативними мобільними застосунками та телеграм-ботами веб-платформа демонструє кращий баланс між швидкістю розробки, функціональністю та зручністю використання.

Було обґрунтовано вибір мови програмування Python для серверної частини, що пояснюється її простотою, широкою підтримкою алгоритмічних і наукових бібліотек, а також зручністю реалізації складних обчислювальних процедур, пов’язаних з роботою оптимізаційного алгоритму AHSA. Node.js, у свою чергу, був розглянутий як альтернативне рішення, проте через характер задач і необхідність виконання інтенсивних обчислень перевага була надана саме Python.

Проведений огляд бібліотек дав змогу відібрати інструменти, що відповідають критеріям надійності, популярності, безпеки та відповідності ліцензійним вимогам. До ключових компонентів було включено Prisma для ORM-взаємодії, FastAPI для побудови серверної логіки, Google-API-клієнт для авторизації та Schema для валідації даних. Такий вибір дає змогу забезпечити стійкість системи, підтримку з боку спільноти та можливість масштабування проєкту.

Архітектура застосунку побудована за принципами клієнт–серверної моделі, що дозволило розділити відповідальність між фронтендом та бекендом. Фронтенд, реалізований засобами React.js, забезпечує динамічний, інтерактивний та зручний для користувачів інтерфейс. Механізми маршрутизації, авторизації та взаємодії з API були спроєктовані таким чином, щоб забезпечити швидку й коректну роботу навіть за умови значного навантаження. Бекенд-частина побудована на Python і відповідає за бізнес-логіку, роботу алгоритму AHSA, взаємодію з базою даних PostgreSQL та забезпечення безпеки даних користувачів.

У рамках реалізації MVP було створено повноцінний функціональний прототип системи, який дає змогу додавати викладачів, групи, дисципліни, налаштовувати параметри алгоритму, запускати генерацію розкладу та експортувати результати. Реалізовані інтерфейси продемонстрували практичну придатність створеної архітектури та підтвердили можливість подальшого масштабування.

Таким чином, результати третього розділу засвідчують, що розроблена система має продуману технологічну основу, реалізована відповідно до сучасних інженерних підходів та готова до інтеграції в навчальні процеси. Обрані програмні засоби та архітектурні рішення забезпечують гнучкість застосунку, а також створюють надійний фундамент для подальшого вдосконалення алгоритму AHSA та розширення функціональності продукту.

РОЗДІЛ 4 РОЗРОБКА СТАРТАП-ПРОЕКТУ ЗА ТЕМОЮ МД

У сучасних умовах цифровізації освітнього процесу зростає потреба у створенні інтелектуальних систем управління навчальними даними, здатних автоматизувати складні організаційні завдання. Однією з таких задач є автоматизоване складання розкладу занять, яке потребує врахування великої кількості параметрів — навчальних планів, викладачів, груп, аудиторій та часових слотів.

Незважаючи на наявність ряду інформаційних систем, більшість із них використовують класичні алгоритмічні підходи, що ґрунтуються на жорстко визначених правилах або простих евристиках. Такі методи не здатні забезпечити високу якість розкладу при великій кількості обмежень, особливо у масштабі великих університетів. Вони часто не враховують гнучкі критерії, такі як комфорт викладачів, рівномірність навантаження чи мінімізація «вікон» у студентів.

У результаті користувачі стикаються з труднощами, пов’язаними з ручним коригуванням розкладу, надмірними витратами часу та неможливістю швидко реагувати на зміни у навчальному процесі. Це створює потребу у нових програмних рішеннях, здатних забезпечити високу якість розкладу завдяки використанню адаптивних алгоритмів оптимізації.

Таким чином, проблема, яку вирішує дана робота, є актуальною як з наукової, так і з практичної точки зору. Розробка програмного застосунку для автоматизованого складання розкладу на основі сучасних методів математичного програмування та комбінаторної оптимізації може стати вагомим внеском у підвищення ефективності освітнього процесу й мати значний потенціал для впровадження у реальні системи управління навчальними закладами.

4.1 План розробки стартапу та масштабування його на ринок

Розробка програмного продукту та його успішне виведення на ринок передбачає послідовне виконання комплексу аналітичних, організаційних та економічних заходів. У межах даної роботи пропонується поетапний план реалізації стартап-проєкту, спрямованого на створення інтелектуальної системи автоматизованого складання розкладу.

Першим етапом є проведення маркетингового аналізу, метою якого є визначення актуальності продукту, потенційного ринку збуту та конкурентних переваг майбутньої системи. Даний етап включає:

1) конкурентний аналіз — дослідження існуючих аналогів і рішень, що вже використовуються на ринку, виявлення їхніх сильних і слабких сторін;
2) формування концепції проєкту — уточнення ідеї продукту, визначення його ключових характеристик і унікальних переваг;
3) визначення цільової аудиторії — окреслення основних груп користувачів (адміністрація навчальних закладів, викладачі, студенти);
4) розроблення стратегії виходу на ринок — побудова моделі позиціонування та плану просування продукту на основі аналізу ринкового середовища та поведінки споживачів.

Другим кроком є організація процесу створення стартапу, у межах якої визначаються технологічні, кадрові та часові параметри реалізації.

На цьому етапі необхідно:

1) сформувати детальний план робіт і таймлайн розробки з визначенням ключових контрольних точок (milestones);
2) оцінити необхідний обсяг ресурсів — людських, технічних і фінансових;
3) здійснити попередній розрахунок витрат на розробку, тестування, маркетинг і підтримку системи;

4) визначити терміни реалізації основних етапів, щоб забезпечити контроль за виконанням графіка проєкту.

Наступним кроком являється організація самого стартапу. На цьому етапі мають бути:

1) складений весь план та побудований таймлайн розробки та запуску продукту;
2) запланований обсяг виробництва та оцінений потенційний обсяг ресурсу, який буде потрібен для виконання плану;
3) розраховані витрати, необхідні для реалізації проекту, та витрати на запуск проекту.

Далі необхідно виконати фінансово-економічний аналіз та оцінити ризики стартап-проекту, в межах якого:

1) визначити обсяг інвестиційних втрат;
2) розрахувати основні фінансово-економічні показники проекту (собівартість, ціну продукту/послуги, податковий збір та чистий прибуток) та визначити показники інвестиційної привабливості проекту (рентабельність продажів, період окупності проекту);
3) визначити основні ризики проекту та способи для їх запобігання.

Фінальним кроком являється розробка заходів з комерціалізації продукту.

Для того, щоб залучити інвесторів та знайти різні способи фінансування проекту, необхідно:

1) провести дослідження на предмет інтересів потенційних інвесторів та бізнесів;
2) скласти інвестиційну пропозицію, яка включає в себе як опис самого продукту та його теперішні розміри, так і можливі шляхи розширення та розвитку;
3) обрати канали комунікації із потенційно зацікавленими персонами.

Далі наведемо результати виконання кожного з описаних кроків.

4.2 Опис ідеї стартап-проекту

Суть продукту полягає у створенні інтелектуальної системи, яка на основі вхідних даних про курси, викладачів, аудиторії та часові слоти автоматично формує оптимальний навчальний розклад.

Система аналізує задані ресурси, часові обмеження, вимоги до навантаження та інші параметри, після чого генерує розклад, що відповідає як жорстким, так і м’яким критеріям оптимальності.

У результаті користувач отримує повністю узгоджений розклад із мінімальними витратами часу на ручне коригування та без порушень обмежень, пов’язаних із доступністю викладачів, аудиторій і навчальних груп.

Застосунок забезпечує прозорий механізм конфігурації параметрів алгоритму, підтримує адаптивне налаштування метаевристичних компонентів та дозволяє повторно генерувати розклад за змінених вхідних умов.

У таблиці 4.1 наведена інформаційна карта стартапу.

Таблиця 4.1 – Інформаційна карта стартап-проекту

Назва проекту	Розклад+
Автори проекту	Гнідобор Сергій Михайлович
Термін реалізації проекту	12 місяців

Продовження таблиці 4.1

Необхідні ресурси	Приміщення, обладнане комп'ютерами з доступом до мережі Інтернет та електроживленням. Програмні засоби для розробки. Фінансові ресурси для виплати заробітної плати виконавцям протягом 12 місяців, а також для покриття витрат на оренду приміщення, комунальні послуги.
Опис проблеми, яку вирішує проект	Продукт вирішує задачу автоматизованого складання навчальних розкладів для освітніх закладів.
Головні цілі та завдання проекту	Метою проєкту є створення системи, яка буде автоматично обробляти вхідні дані про курси, викладачів та аудиторії й формувати оптимізований розклад занять.
Очікувані результати	Залучення навчальних установ та ІТ-компаній до використання стартапу, автономна система для генерації та оптимізації розкладів.

4.3 Технологічний аудит ідеї проекту

Тепер можна розібрати ідею стартапу та провести конкурентний аналіз. У таблиці 4.2 наведений опис ідеї стартапу.

Таблиця 4.2 – Опис ідеї стартапу

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Основна ідея полягає у створенні комплексної системи, яка автоматично формуватиме оптимальний навчальний розклад	Автоматизація процесу складання розкладів у закладах освіти	Користувач зможе швидко отримувати безконфліктний і збалансований розклад занять
	Постійне оновлення та адаптація системи до специфіки навчальних процесів	Якість розкладу буде поступово покращуватись, забезпечуючи персоналізований підхід до користувачів

Далі аналізуємо реальність технічно здійснити ідею проекту (таблиця 4.3).

Таблиця 4.3 – Технологічна здійсненність продукту

№ п/п	Ідея проекту	Технології і реалізації	Наявність технологій	Доступність технологій
1	Створення комплексної системи, яка буде приймати на вхід дані про курси, викладачів і аудиторії та формуватиме оптимальний розклад	Використання мови програмування Python з бібліотеками для оптимізації (NumPy, SciPy, DEAP)	Наявні	Доступні
2		Реалізація алгоритмів оптимізації у вигляді мікросервісів	Використання Node.js для API-компонентів	Доступні
3		Використання TypeScript / React для клієнтської частини	Наявні, необхідні допрацювання	Доступні
Обрана технологія реалізації ідеї проекту: Python				

4.4 Аналіз ринкових можливостей запуску стартап-проекту

Далі проведемо попередній аналіз ринку для запуску стартап-проекту (таблиця 4.4).

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники ринку (найменування)	Характеристика
1	Кількість головних гравців, од	4
2	Загальний обсяг продаж, грн/ум.од	9000
3	Динаміка ринку (якісна оцінка)	Позитивна, зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Відсутні
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні, програмні рішення відкритого типу
6	Середня норма рентабельності в галузі (або по ринку), %	9%

Тепер проведемо характеристику потенційних клієнтів, які можуть бути зацікавлені в проекті (таблиця 4.5).

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреби, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Автоматизація складання навчальних розкладів	Заклади вищої освіти, факультети, деканати	Потребують точності, стабільності та можливості швидких змін у розкладі	Простота використання, надійність
2	Оптимізація навантаження викладачів та аудиторій	Адміністрація навчальних закладів	Орієнтовані на ефективність використання ресурсів	Висока якість оптимізації, зручна аналітика
3	Планування навчального процесу в режимі реального часу	Викладачі, студенти	Прагнуть мати швидкий доступ до актуального розкладу	Можливість онлайн-доступу та автоматичних оновлень

Обрахуємо фактори загроз (таблиця 4.6) та можливостей (таблиця 4.7). Проаналізуємо загрози, щоб зрозуміти можливі перешкоди при запуску продукт на ринок. Фактори можливостей же треба обрахувати, щоб знати усі сприятливі умови та по можливості ними скористатися.

Таблиця 4.6 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренція	Хоча ринок залишається відкритим і недостатньо освоєним, на ньому вже присутні кілька великих компаній, які мають сформовану клієнтську базу та власні сегменти користувачів.	Знайти точки додаткової цінності для користувача
2	Ціна збуту	Конкурентні продукти можуть пропонуватись за нижчою ціною, що	Сфокусуватися на якості роботи застосунку та продумати маркетингову стратегію
3	Якість аналізу	Через комплексність задачі, можливі помилки в роботі алгоритму формування розкладу	Мати достатній штаб і ресурси, для тестування алгоритму на всіх можливих варіантах використання включаючи але не обмежуючись нецільовим використанням системи

Таблиця 4.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Доступність	Продукт доступний онлайн і не вимагає встановлення як в більшості конкурентів	Зробити акцент при маркетингу, продовжувати розвиток як окремого продукту
2	Простота у використанні	Від користувача треба лише завантажити данні	Реалізувати зручний інтерфейс для завантаження
3	Якість та гарантії	Надавати найбільш якісний розклад який задовольняє всіх учасників навчального процесу	також надавати усю необхідну технічну підтримку
4	Безкоштовний сервіс при MVP	Максимально швидко набрати базу своїх клієнтів та заявити про себе на ринку	Розгорнути маркетинг

Далі розглянемо питання конкуренції, а саме визначимо її тип та рівень (таблиця 4.8).

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	У чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції: недосконала конкуренція	Представлено мало продуктів	Зробити максимальним збут застосунку
2. За рівнем конкурентної боротьби: міжнародний	Наявні проекти, розроблені та можуть бути доступні у всьому світі	Додати інтерфейс з іноземними мовами
3. За галузевою ознакою: внутрішньогалузева	Можуть працювати з різними галузями	Покращити персоналізацію
4. Конкуренція	Конкуренція з існуючими рішеннями	Підтримувати та покращувати якість існуючих результату
5. За характером конкурентних переваг: нецінова	Різні компанії пропонують різну якість	Постійно покращувати алгоритм і підлаштовувати під реальні потреби
6. За інтенсивністю: марочна	Вже представлені компанії із сильним	Враховувати побажання користувачів

	брендом	

Далі необхідно виконаємо аналіз конкуренції за моделлю 5 сил конкуренції Майкла Портера (таблиця 4.9).

Таблиця 4.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти у галузі	Потенційні конкуренти	Постачальники	Клієнти	Товаровиробники
	Інші існуючі системи та продукти	Якість, ціни, кількість капіталовкладення	Фактори сили постачальників	Контроль якості, порівняння цін	Сила бренду, якість, ціна, масштаби
Висновки	Конкуренція з невеликою інтенсивністю, а також підігрітий ринок	Можливості входження на ринок, нові потенційні конкуренти	Постачальники відсутні	Клієнти не диктують умови роботи на ринку	Товаровиробники відсутні

Далі було сформульовано та обґрунтовано перелік факторів конкурентоспроможності (таблиця 4.10).

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Універсальність	Продукт доступний через мережу інтернет
2	Простота у використанні	Від користувача треба лише завантажити вхідні данні
3	Якість та гарантії	Система гарантує створення оптимального розкладу який максимально враховує побажання всіх учасників навчального процесу
4	Безкоштовний сервіс при MVP	Максимально швидко набрати базу своїх клієнтів

Тепер можна провести аналіз сильних та слабких сторін продукту (таблиця 4.11).

Таблиця 4.11 – Порівняльний аналіз сильних та слабких сторін системи

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг конкурентів						
			-3	-2	-1	0	1	2	3
1	Універсальність	20	+						
2	Простота у використанні	16		+					
3	Якість та гарантії	10		+					
4	Безкоштовний сервіс при MVP	17			+				

Далі проведемо SWOT-аналіз продукту (таблиця 4.12).

Таблиця 4.12 – SWOT-аналіз стартап-проекту

Сильні сторони Універсальність Якість Безкоштовний сервіс при MVP	Слабкі сторони Відсутність сильного бренду Не сформована база клієнтів Маркетинг
Можливості Покращення системи Персоналізація Інтеграція з існуючими системами завдяки гнучкому експорту розкладу	Нові системи Збут

Завдяки проведенню SWOT-аналізу, ми змогли визначити сильні та слабкі сторони, можливості та загрози, пов'язані з конкуренцією та плануванням стартап-

проекту. Далі спроектуємо альтернативну ринкову поведінку для інтеграції стартап-проекту на ринок та приблизний час реалізації системного комплексу, з урахуванням потенційних проектів, що можуть бути виведені на ринок та наведемо результати у таблиці 4.13.

Таблиця 4.13 – Альтернативи ринкового впровадження стартап проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Вихід на ринок з нижче якістю	70%	4 місяці
2	Пропонувати одразу платне використання	50%	6 місяців
3	Представлення користувачам системи без інтерфейсу	60%	5 місяці

У даному пункті був проведений детальний аналіз ринку та продукту. Також відповідно до результатів проведеного конкурентного аналізу, визначених факторів ринку та його сприятливість, описання ідеї та характеристик стартап-проекту, робимо висновок висновок, що існують дуже сприятливі умови для виходу продукту на ринок.

4.5 Розроблення ринкової стратегії стартап-проекту

Для розробки ринкової стратегії продукту, у першу чергу, необхідно проаналізувати цільову аудиторію проекту (таблиця 4.14).

Таблиця 4.14 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит у межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Персональні користувачі	Висока	26%	Висока	Середня
2	Вищі навчальні заклади	Висока	21%	Середня	Середня
Які цільові групи обрано: 1, 2					

Маючи аналіз цільових груп, далі визначимо базову стратегію розвитку продукту (таблиця 4.15).

Таблиця 4.15 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
1	1 та 3	Диференційованого маркетингу	Масштабування та максимізація	Оптимальних витрат

Для роботи в обраних сегментах ринку сформовано базову стратегію розвитку (таблиці 4.16, 4.17).

Таблиця 4.16 – Визначення базової стратегії конкурентної поведінки

Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
Ні	Так	Ні	Домінація на ринку завдяки інноваційним рішенням

Таблиця 4.17 – Визначення стратегії позиціонування

Вимоги до продукту цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
Простота у використанні Якість результатів	Оптимальних витрат	Універсальність Простота у використанні Якість Безкоштовне використання при MVP	Система, яка надає найкращий розклад і якою може користуватись навіть не підготовлена людина

4.6 Розроблення маркетингової програми стартап-проекту

Після проведеного комплексного аналізу, можемо повноцінно описати ключові переваги концепції потенційного товару (таблиця 4.18) та побудувати концепцію маркетингових комунікацій (таблиця 4.19).

Таблиця 4.18 – Ключові переваги концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	Якісна генерація розкладу	Розклад який влаштовує всіх учасників навчального процесу	Постійне покращення та підлаштування під специфіку навчального процесу
2	Універсальність	Система не залежить від апаратної .	Такого виду систему може використовувати навіть непідготовлений користувач
3	Простий інтерфейс	Система дуже проста у використанні	Система із інтуїтивно зрозумілим інтерфейсом, який вимагає всього лише вхідних данних

Таблиця 4.19 – Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламно о повідомлення	Концепція рекламного звернення
1	Пошук спеціалізованих систем	b2b продажі Зв’язок через теплі контакти	Точність Якість	Поєднати повідомлення про те, що ця система створює розклад краще ніж іниші	Переговори з представниками ВНЗ

Висновки до розділу 4

Даний розділ був присвячений дослідженню стартап-проєкту SmartSched, спрямованого на автоматизацію процесу складання навчальних розкладів у закладах освіти.

У межах розділу було розглянуто стратегії виходу продукту на ринок та побудову ефективної маркетингової стратегії. Проведений аналіз показав, що ринок подібних систем є перспективним і ще недостатньо насиченим конкурентами. Існуючі рішення забезпечують лише часткову автоматизацію процесу, тоді як запропонована система є універсальною, адаптивною та зручною у використанні, що створює умови для швидкого зайняття лідируючих позицій на ринку.

Крім того, у розділі були розглянуті сильні та слабкі сторони проєкту, проведено SWOT-аналіз, а також аналіз конкурентного середовища та цільових аудиторій. На основі отриманих результатів було сформовано концепцію маркетингової стратегії, орієнтовану на навчальні заклади, адміністрації факультетів та IT-компанії, зацікавлені в інтеграції системи SmartSched у власну інфраструктуру.

Таким чином, проведені дослідження підтвердили, що стартап має високий потенціал розвитку і здатен зайняти стабільну нішу на ринку освітніх програмних рішень.

ВИСНОВКИ

У ході виконання магістерської дисертації було комплексно досліджено проблему автоматизованого складання навчальних розкладів у закладах вищої освіти, проаналізовано сучасні алгоритмічні підходи до її розв’язання та розроблено інтелектуальну систему на основі адаптивного гібридного метаевристичного алгоритму AHSA (Adaptive Hybrid Scheduling Algorithm). Результати роботи підтверджують актуальність обраної тематики та демонструють ефективність запропонованого методу для практичного застосування в освітніх інформаційних системах.

У першому розділі було здійснено огляд існуючих алгоритмів оптимізації — точних (ILP, CP), евристичних (greedy, rule-based) та метаевристичних (генетичний алгоритм, імітація відпалу, табу-пошук). Проведений аналіз показав, що жоден з класичних підходів не може ефективно забезпечити одночасно масштабованість, високу якість розкладу та стійкість до зміни вхідних даних. На основі порівняння встановлено, що метаевристичні алгоритми є найбільш перспективними, однак потребують підвищення адаптивності та комбінування сильних сторін різних методів. Це обґрунтувало необхідність розроблення нового гібридного підходу — AHSA.

У другому розділі сформульовано математичну постановку задачі складання розкладу та наведено моделі, що описують структуру розкладу та обмежень. Визначено множини ресурсів (курси, групи, викладачі, аудиторії, часові слоти), введено бінарні змінні та побудовано систему жорстких і м’яких обмежень. Запропоновано узагальнену цільову функцію, яка враховує сумарні штрафи за порушення різних типів обмежень. Розглянуто ILP-модель, CP-підхід і графову модель розкладу, що дало змогу формально оцінити межі застосовності класичних оптимізаційних методів та підтвердити доцільність використання метаевристик.

У третьому розділі представлено розроблений алгоритм AHSA та описано архітектуру програмної системи. AHSA поєднує глобальний еволюційний пошук (генетичний алгоритм), локальну оптимізацію (імітацію відпалу) та адаптивний модуль регулювання параметрів. Наведено структуру клієнт–серверного застосунку, механізми авторизації, інтеграції з базою даних і відображення результатів. Було реалізовано MVP-версію системи, що забезпечує введення даних, налаштування алгоритму, генерацію розкладу та експорт результатів. Експериментальні тести на демонстраційному датасеті показали, що AHSA забезпечує вищу якість розкладу та менший час обчислень порівняно з класичним GA, greedy-методами та ILP-моделюванням.

У четвертому розділі розроблено стартап-проєкт, спрямований на комерціалізацію інтелектуальної системи формування розкладів. Проведено аналіз ринку, визначено цільові сегменти користувачів та конкурентне середовище. Побудовано економічну модель, оцінено витрати на реалізацію продукту, можливі ризики та стратегію виходу на ринок. Запропоновані маркетингові та організаційні рішення демонструють потенціал системи AHSA для впровадження в університетах, коледжах, приватних освітніх закладах і EdTech-платформах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Pinedo M. L. Scheduling: Theory, Algorithms, and Systems. 5th ed. New York: Springer, 2016. 670 p.
2. Nemhauser G. L., Wolsey L. A. Integer and Combinatorial Optimization. New York: Wiley, 1999. 763 p.
3. Hillier F. S., Lieberman G. J. Introduction to Operations Research. 10th ed. New York: McGraw-Hill Education, 2015. 1220 p.
4. Gendreau M., Potvin J.-Y. (eds.) Handbook of Metaheuristics. 3rd ed. Cham: Springer, 2019. 668 p. (International Series in Operations Research & Management Science).
5. Talbi E.-G. Metaheuristics: From Design to Implementation. Hoboken: John Wiley & Sons, 2009. 624 p.
6. Glover F., Laguna M. Tabu Search. Boston: Springer, 1997. 376 p. (International Series in Operations Research & Management Science).
7. Goldberg D. E. Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley, 1989. 432 p.
8. Kirkpatrick S., Gelatt C. D., Vecchi M. P. Optimization by Simulated Annealing // Science. 1983. Vol. 220, No. 4598. P. 671–680.
9. Rossi F., Van Beek P., Walsh T. (eds.) Handbook of Constraint Programming. Amsterdam: Elsevier, 2006. 950 p. (Foundations of Artificial Intelligence; Vol. 2).
10. Jensen T. R., Toft B. Graph Coloring Problems. New York: Wiley-Interscience, 1995. 295 p.
11. Brucker P. Scheduling Algorithms. 5th ed. Berlin: Springer-Verlag, 2007. 367 p.
12. Leung J. Y.-T. (ed.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis. Boca Raton: Chapman & Hall/CRC Press, 2004. 1224 p.

13. Burke E. K., Kendall G. (eds.) Search Methodologies: Introductory Tutorials in Optimization, Decision Support and Data Mining. - 2nd ed. - New York: Springer, 2014. - 779 p.
14. Burke E. K., Petrovic S. Recent Research Directions in Automated Timetabling // In: Burke E. K., De Causmaecker P. (eds.) Practice and Theory of Automated Timetabling IV (PATAT 2002). Berlin: Springer, 2002. P. 1–21.
15. Ahuja R. K., Magnanti T. L., Orlin J. B. Network Flows: Theory, Algorithms, and Applications. Upper Saddle River: Prentice Hall, 1993. 846 p.
16. Norman D. The Design of Everyday Things. New York: Basic Books, 2013. 368 p. ISBN 978-0-465-06710-7.
17. Node.js — JavaScript runtime. URL: https://nodejs.org/
18. Welcome to Python.org. Python.org. URL: https://www.python.org/ (date of access: 08.12.2025).
19. Clements P., Bass L., Kazman R. Software Architecture in Practice. Pearson Education, Limited, 2021.
20. React Router Official Documentation. React Router Official Documentation. URL: https://reactrouter.com/ (date of access: 08.12.2025).
21. Бідюк П.І., Романенко В.Д., Тимощук О.Л. Аналіз часових рядів: навч. посіб. Київ: НТУУ «КПІ», 2013. 600 с.
22. Гнідобор С.М., Тимощук О.Л. Інтелектуальна система автоматизованого складання розкладу з використанням адаптивних гібридних метаевристик. І Всеукраїнська науково-практична конференція «Системні науки та інформатика», 1-5 грудня 2025, Київ, КПІ ім. Ігоря Сікорського, С. 82-89.

ДОДАТОК А. Лістинг програмного продукту

```
// Dynamic API client that automatically uses localhost or production URL
export function getApiBaseUrl(): string {
  // For client-side code
  if (typeof window !== "undefined") {
    return typeof process.env.NEXT_PUBLIC_API_URL !== "undefined"
      ? process.env.NEXT_PUBLIC_API_URL
      : window.location.origin
  }
  // For server-side code
  return process.env.API_URL || "http://localhost:3000"
}

export async function apiCall<T>(endpoint: string, options?: RequestInit):
      Promise<T> {
  const baseUrl = getApiBaseUrl()
  const url = `${baseUrl}/api${endpoint}`

  const response = await fetch(url, {
    headers: {
      "Content-Type": "application/json",
      ...options?.headers,
    },
    ...options,
  })

  if (!response.ok) {
```

```
    throw new Error(`API Error: ${response.statusText}`)
  }

  return response.json()
}

export async function apiGet<T>(endpoint: string): Promise<T> {
  return apiCall<T>(endpoint, { method: "GET" })
}

export async function apiPost<T>(endpoint: string, data: unknown): Promise<T> {
  return apiCall<T>(endpoint, {
    method: "POST",
    body: JSON.stringify(data),
  })
}

export async function apiPut<T>(endpoint: string, data: unknown): Promise<T> {
  return apiCall<T>(endpoint, {
    method: "PUT",
    body: JSON.stringify(data),
  })
}

export async function apiDelete<T>(endpoint: string): Promise<T> {
  return apiCall<T>(endpoint, { method: "DELETE" })
}
"use client"
```

```
import { useState } from "react"
import { Button } from "@/components/ui/button"
import { Card } from "@/components/ui/card"
import { Download, RefreshCw, AlertCircle } from "lucide-react"
import { apiPost } from "@/lib/api-client"

export default function ScheduleGenerator({ scheduleData }: any) {
  const [schedule, setSchedule] = useState<any[]>([])
  const [isGenerating, setIsGenerating] = useState(false)
  const [stats, setStats] = useState({ conflicts: 0, coverage: 0, efficiency: 0 })
  const [error, setError] = useState<string | null>(null)

  const daysOfWeek = ["Понеділок", "Вівторок", "Середа", "Четвер", "П'ятниця"]
  const timeSlots = ["09:00", "09:50", "10:50", "11:40", "12:30", "13:20", "14:10"]

  const generateSchedule = async () => {
    setIsGenerating(true)
    setError(null)
    try {
      const response = await apiPost<any>("/schedule", {
        teachers: scheduleData?.teachers || [],
        groups: scheduleData?.groups || [],
        algorithm: scheduleData?.algorithm || "ahsa",
      })

      if (response.schedule && response.schedule.length > 0) {
        setSchedule(response.schedule)
```

```
      setStats(response.stats || { conflicts: 0, coverage: 95, efficiency: 92 })
    } else {
      setError("Не вдалося згенерувати розклад. Переконайтесь, що додані
    учителі та групи.")
    }
  } catch (err) {
    setError(err instanceof Error ? err.message : "Помилка генерування розкладу")
  } finally {
    setIsGenerating(false)
  }
}

return (
  <div className="space-y-6">
    <div className="flex gap-4">
      <Button onClick={generateSchedule} disabled={isGenerating} size="lg"
    className="gap-2">
        <RefreshCw className={`w-5 h-5 ${isGenerating ? "animate-spin" : ""}`} />
        {isGenerating ? "Генерування..." : "Згенерувати розклад"}
      </Button>
      <Button variant="outline" size="lg" className="gap-2 bg-transparent">
        <Download className="w-5 h-5" />
        Експортувати CSV
      </Button>
      <Button variant="outline" size="lg" className="gap-2 bg-transparent">
        <Download className="w-5 h-5" />
        Експортувати PDF
      </Button>
```

```
</div>

{error && (
  <Card className="p-4 bg-destructive/10 border-destructive/20">
    <p className="text-destructive text-sm">{error}</p>
  </Card>
)}

{schedule.length > 0 && (
  <>
    <div className="grid grid-cols-3 gap-4">
      <Card className="p-4">
        <div className="text-center">
          <p className="text-sm text-muted-foreground">Конфлікти</p>
          <p className={`text-3xl font-bold ${stats.conflicts === 0 ? "text-green-
600" : "text-destructive"}`}>
            {stats.conflicts}
          </p>
        </div>
      </Card>
      <Card className="p-4">
        <div className="text-center">
          <p className="text-sm text-muted-foreground">Охоплення</p>
          <p className="text-3xl font-bold text-blue-600">{stats.coverage}%</p>
        </div>
      </Card>
      <Card className="p-4">
        <div className="text-center">
```

```
        <p className="text-sm text-muted-foreground">Ефективність</p>
        <p className="text-3xl font-bold text-amber-
600">{stats.efficiency}%</p>
      </div>
    </Card>
  </div>

  <Card className="p-6 overflow-x-auto">
    <h2 className="text-xl font-bold mb-4 text-foreground">Розклад
занять</h2>
    <div className="min-w-max">
     <table className="w-full text-sm border-collapse">
       <thead>
         <tr className="border-b-2 border-border">
           <th className="p-3 text-left font-semibold text-foreground">День</th>
           <th className="p-3 text-left font-semibold text-foreground">Час</th>
           <th className="p-3 text-left font-semibold text-foreground">Клас</th>
           <th className="p-3 text-left font-semibold text-
foreground">Група</th>
           <th className="p-3 text-left font-semibold text-
foreground">Учитель</th>
           <th className="p-3 text-left font-semibold text-
foreground">Предмет</th>
         </tr>
       </thead>
       <tbody>
         {schedule.map((slot, idx) => (
```

```
                <tr key={idx} className="border-b border-border hover:bg-accent/5
    transition-colors">
                  <td className="p-3 text-foreground">{slot.day}</td>
                  <td className="p-3 text-foreground">{slot.time}</td>
                  <td className="p-3 text-foreground">{slot.room}</td>
                  <td className="p-3 text-foreground">{slot.group}</td>
                  <td className="p-3 text-foreground">{slot.teacher}</td>
                  <td className="p-3 text-foreground">{slot.subject}</td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      </Card>
    </>
  )}

  {schedule.length === 0 && (
    <Card className="p-8 text-center">
      <AlertCircle className="w-12 h-12 text-muted-foreground mx-auto mb-3" />
      <p className="text-muted-foreground">
        Розклад ще не згенерований. Натисніть кнопку "Згенерувати розклад"
    щоб почати.
      </p>
    </Card>
  )}
 </div>
)
```

```
}
"use client"

import { Button } from "@/components/ui/button"
import { Card } from "@/components/ui/card"
import { Users, BookOpen, Zap, Calendar } from "lucide-react"

interface DashboardProps {
  onNavigate: (page: string) => void
}

export default function Dashboard({ onNavigate }: DashboardProps) {
  const steps = [
    {
      id: "teachers",
      title: "Учителі та предмети",
      description: "Додайте учителів та визначте їхні предмети",
      icon: Users,
      color: "bg-blue-100 text-blue-600",
    },
    {
      id: "groups",
      title: "Групи студентів",
      description: "Створіть групи студентів та їхні часові пояси",
      icon: BookOpen,
      color: "bg-purple-100 text-purple-600",
    },
    {
```

```
    id: "algorithm",
    title: "Конфігурація алгоритму",
    description: "Виберіть та налаштуйте алгоритм генерації розкладу",
    icon: Zap,
    color: "bg-amber-100 text-amber-600",
  },
  {
    id: "schedule",
    title: "Генерація розкладу",
    description: "Створіть оптимальний розклад на основі параметрів",
    icon: Calendar,
    color: "bg-green-100 text-green-600",
  },
]

return (
  <div className="space-y-8">
    <div className="text-center">
      <h1 className="text-4xl font-bold text-foreground mb-3">Система складання
    розкладу</h1>
      <p className="text-muted-foreground text-lg">Легко створюйте оптимальні
    розклади для учителів та студентів</p>
    </div>

    <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">
      {steps.map((step) => {
        const Icon = step.icon
        return (
```

```
      <Card
        key={step.id}
        className="hover:shadow-lg transition-shadow duration-300 cursor-
pointer"
        onClick={() => onNavigate(step.id)}
      >
        <div className="p-6 space-y-4">
          <div className={`w-12 h-12 rounded-lg flex items-center justify-center
${step.color}`}>
            <Icon className="w-6 h-6" />
          </div>
          <div>
            <h3 className="font-semibold text-lg text-foreground">{step.title}</h3>
            <p className="text-sm text-muted-foreground">{step.description}</p>
          </div>
          <Button variant="outline" size="sm" className="w-full bg-transparent">
            Перейти
          </Button>
        </div>
      </Card>
    )
  })}
</div>

<Card className="bg-accent/5 p-8">
  <h2 className="text-2xl font-bold text-foreground mb-4">Можливості
системи</h2>
  <ul className="space-y-3 text-foreground grid grid-cols-1 md:grid-cols-2">
```

```
        <li className="flex gap-2">
          <span className="text-accent font-bold">✓</span> Керування учителями
      та предметами
        </li>
        <li className="flex gap-2">
          <span className="text-accent font-bold">✓</span> Керування групами
      студентів
        </li>
        <li className="flex gap-2">
          <span className="text-accent font-bold">✓</span> Кілька алгоритмів
      розподілу
        </li>
        <li className="flex gap-2">
          <span className="text-accent font-bold">✓</span> Перевірка конфліктів
        </li>
        <li className="flex gap-2">
          <span className="text-accent font-bold">✓</span> Експорт розкладу
        </li>
        <li className="flex gap-2">
          <span className="text-accent font-bold">✓</span> Налаштування часових
      поясів
        </li>
      </ul>
    </Card>
  </div>
 )
}"use client"
```

```
import { useState } from "react"
import { Button } from "@/components/ui/button"
import { Card } from "@/components/ui/card"
import { Zap, Check, Crown } from "lucide-react"

type Algorithm = "greedy" | "genetic" | "simulated-annealing" | "constraint-
    satisfaction" | "ahsa"

interface AlgorithmOption {
  id: Algorithm
  name: string
  description: string
  complexity: "low" | "medium" | "high"
  bestFor: string
  isRecommended?: boolean
  badge?: string
}

export default function AlgorithmConfig() {
  const [selectedAlgorithm, setSelectedAlgorithm] = useState<Algorithm>("ahsa")
  const [config, setConfig] = useState({
    preferTeacherContinuity: true,
    avoidConsecutiveClasses: true,
    balanceGroupSchedule: true,
    minimizeBreaks: true,
    maxClassesPerDay: 5,
    minBreakMinutes: 15,
    enableHybridMode: true,
```

```
  populationSize: 100,
  generationLimit: 500,
  adaptiveWeighting: true,
})

const algorithms: AlgorithmOption[] = [
  {
    id: "ahsa",
    name: "AHSA (Гібридний алгоритм)",
    description: "Комбінує еволюційні та метаевристичні методи для
    максимальної якості",
    complexity: "high",
    bestFor: "Великі навчальні установи (100+ груп) з складними вимогами",
    isRecommended: true,
    badge: "РЕКОМЕНДОВАНО",
  },
  {
    id: "greedy",
    name: "Жадний алгоритм",
    description: "Швидкий алгоритм для простих випадків",
    complexity: "low",
    bestFor: "Малі школи з простими обмеженнями",
  },
  {
    id: "constraint-satisfaction",
    name: "Задоволення обмежень",
    description: "Глибоко урахує всі обмеження",
    complexity: "medium",
```

```
    bestFor: "Середні школи з багатьма обмеженнями",
  },
  {
    id: "genetic",
    name: "Генетичний алгоритм",
    description: "Еволюційний підхід для оптимальних результатів",
    complexity: "high",
    bestFor: "Великі установи з складною структурою",
  },
  {
    id: "simulated-annealing",
    name: "Імітація відпалу",
    description: "Метаевристичний метод для пошуку глобального оптимуму",
    complexity: "high",
    bestFor: "Великі системи з багатьма змінними",
  },
]

const complexityColor = {
  low: "bg-green-100 text-green-700",
  medium: "bg-amber-100 text-amber-700",
  high: "bg-red-100 text-red-700",
}

const complexityLabel = {
  low: "Низька",
  medium: "Середня",
  high: "Висока",
```

```
}

return (
  <div className="space-y-8">
    <div>
      <h1 className="text-2xl font-bold text-foreground mb-2">Вибір алгоритму
    розкладу</h1>
      <p className="text-muted-foreground">
        AHSA рекомендується для навчальних закладів з 100+ групами студентів.
    Його гібридний підхід забезпечує
        найкращу якість та найменше конфліктів.
      </p>
    </div>

    <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
      {algorithms.map((algo) => (
        <Card
          key={algo.id}
          className={`p-5 cursor-pointer transition-all duration-300 relative overflow-
    hidden ${
            selectedAlgorithm === algo.id
              ? algo.isRecommended
                ? "ring-2 ring-amber-500 bg-amber-50/30 shadow-lg"
                : "ring-2 ring-primary bg-primary/5"
              : "hover:shadow-md"
          } ${algo.isRecommended ? "border-2 border-amber-200" : ""}`}
          onClick={() => setSelectedAlgorithm(algo.id)}
        >
```

```
    {algo.isRecommended && (
     <div className="absolute top-3 right-3 flex items-center gap-1 bg-amber-
500 text-white px-3 py-1 rounded-full text-xs font-bold">
      <Crown className="w-3 h-3" />
      {algo.badge}
     </div>
    )}
    <div className="flex justify-between items-start mb-3 pr-20">
     <div className="flex items-start gap-3">
      <div className="mt-1">
       <div
        className={`w-5 h-5 rounded border-2 flex items-center justify-center
${
         selectedAlgorithm === algo.id
          ? algo.isRecommended
           ? "border-amber-500 bg-amber-500"
           : "border-primary bg-primary"
          : "border-border"
        }`}
       >
        {selectedAlgorithm === algo.id && <Check className="w-4 h-4 text-
white" />}
       </div>
      </div>
      <div>
       <h3 className="font-bold text-foreground">{algo.name}</h3>
       <p className="text-sm text-muted-foreground mt-
1">{algo.description}</p>
```

```
          </div>
        </div>
        <span className={`text-xs font-semibold px-3 py-1 rounded-full
${complexityColor[algo.complexity]}`}>
          {complexityLabel[algo.complexity]}
        </span>
      </div>
      <p className="text-xs text-muted-foreground">
        <span className="font-medium">Найкраще для:</span> {algo.bestFor}
      </p>
    </Card>
  ))}
</div>

{selectedAlgorithm && (
  <Card className="p-6 bg-accent/5">
    <h2 className="text-xl font-bold mb-6 text-foreground">Налаштування
алгоритму</h2>

    <div className="space-y-4">
      <div className="flex items-center justify-between p-4 bg-card border border-
border rounded-lg">
        <label className="font-medium text-foreground">Перевага
безперервності уроків учителя</label>
        <input
          type="checkbox"
          checked={config.preferTeacherContinuity}
```

```
        onChange={(e) => setConfig({ ...config, preferTeacherContinuity:
e.target.checked })}
        className="w-5 h-5 cursor-pointer"
      />
    </div>

    <div className="flex items-center justify-between p-4 bg-card border border-
border rounded-lg">
      <label className="font-medium text-foreground">Уникати послідовних
занять однієї групи</label>
      <input
        type="checkbox"
        checked={config.avoidConsecutiveClasses}
        onChange={(e) => setConfig({ ...config, avoidConsecutiveClasses:
e.target.checked })}
        className="w-5 h-5 cursor-pointer"
      />
    </div>

    <div className="flex items-center justify-between p-4 bg-card border border-
border rounded-lg">
      <label className="font-medium text-foreground">Збалансований розклад
для груп</label>
      <input
        type="checkbox"
        checked={config.balanceGroupSchedule}
        onChange={(e) => setConfig({ ...config, balanceGroupSchedule:
e.target.checked })}
```

```
          className="w-5 h-5 cursor-pointer"
        />
      </div>

      <div className="flex items-center justify-between p-4 bg-card border border-border rounded-lg">
        <label className="font-medium text-foreground">Мінімізувати перерви</label>
        <input
          type="checkbox"
          checked={config.minimizeBreaks}
          onChange={(e) => setConfig({ ...config, minimizeBreaks: e.target.checked })}
          className="w-5 h-5 cursor-pointer"
        />
      </div>

      <div className="p-4 bg-card border border-border rounded-lg">
        <label className="font-medium text-foreground block mb-3">
          Максимум занять на день: {config.maxClassesPerDay}
        </label>
        <input
          type="range"
          min="3"
          max="8"
          value={config.maxClassesPerDay}
          onChange={(e) => setConfig({ ...config, maxClassesPerDay: Number.parseInt(e.target.value) })}
```

```
        className="w-full"
      />
    </div>

    <div className="p-4 bg-card border border-border rounded-lg">
      <label className="font-medium text-foreground block mb-3">
        Мінімум хвилин на перерву: {config.minBreakMinutes}
      </label>
      <input
        type="range"
        min="5"
        max="60"
        value={config.minBreakMinutes}
        onChange={(e) => setConfig({ ...config, minBreakMinutes:
Number.parseInt(e.target.value) })}
        className="w-full"
      />
    </div>

    {selectedAlgorithm === "ahsa" && (
      <>
        <div className="p-4 bg-amber-50/50 border border-amber-200 rounded-
lg">
          <h3 className="font-semibold text-foreground mb-4 flex items-center
gap-2">
            <Crown className="w-4 h-4 text-amber-500" />
            AHSA Вдосконалені налаштування
          </h3>
```

```
          <div className="flex items-center justify-between mb-4">
            <label className="font-medium text-foreground">
              Гібридний режим (комбінація генетики + імітація)
            </label>
            <input
              type="checkbox"
              checked={config.enableHybridMode}
              onChange={(e) => setConfig({ ...config, enableHybridMode:
e.target.checked })}
              className="w-5 h-5 cursor-pointer"
            />
          </div>

          <div className="mb-4">
            <label className="font-medium text-foreground block mb-3">
              Розмір популяції: {config.populationSize}
            </label>
            <input
              type="range"
              min="50"
              max="500"
              step="50"
              value={config.populationSize}
              onChange={(e) => setConfig({ ...config, populationSize:
Number.parseInt(e.target.value) })}
              className="w-full"
            />
```

```
          <p className="text-xs text-muted-foreground mt-2">Більший розмір =
кращі результати, більше часу</p>
        </div>

        <div className="mb-4">
          <label className="font-medium text-foreground block mb-3">
            Ліміт поколінь: {config.generationLimit}
          </label>
          <input
            type="range"
            min="100"
            max="1000"
            step="100"
            value={config.generationLimit}
            onChange={(e) => setConfig({ ...config, generationLimit:
Number.parseInt(e.target.value) })}
            className="w-full"
          />
          <p className="text-xs text-muted-foreground mt-2">
            Більше поколінь = можливість більше оптимізації
          </p>
        </div>

        <div className="flex items-center justify-between">
          <label className="font-medium text-foreground">Адаптивне
зважування критеріїв</label>
          <input
            type="checkbox"
```

```
                checked={config.adaptiveWeighting}
                onChange={(e) => setConfig({ ...config, adaptiveWeighting:
    e.target.checked })}
                className="w-5 h-5 cursor-pointer"
              />
            </div>
          </div>
        </>
      )}
    </div>
  </Card>
)}

<div className="flex gap-4">
  <Button size="lg" className={`gap-2 ${selectedAlgorithm === "ahsa" ? "bg-
amber-600 hover:bg-amber-700" : ""}`}>
    <Zap className="w-5 h-5" />
    Генерувати розклад
  </Button>
  <Button variant="outline" size="lg">
    Скинути налаштування
  </Button>
</div>
</div>
)
}
from __future__ import annotations
from fastapi import FastAPI, HTTPException
```

```
from pydantic import BaseModel, Field
from typing import List, Dict, Optional, Literal, Tuple
from uuid import uuid4
import math, random, asyncio

AlgorithmKind = Literal['AHSA', 'GREEDY', 'GA', 'SA']

class Teacher(BaseModel):
    id: int
    name: str
    canTeachCourseIds: List[int] = []
    busySlots: List[int] = []

class Group(BaseModel):
    id: int
    size: int
    name: str

class Room(BaseModel):
    id: int
    capacity: int
    name: str
    features: List[str] = []

class Slot(BaseModel):
    id: int
    day: int
    time: int
```

```
class Course(BaseModel):
    id: int
    groupId: int
    teacherId: int
    hours: int
    roomReq: Dict = {}

class Dataset(BaseModel):
    groups: List[Group]
    teachers: List[Teacher]
    rooms: List[Room]
    slots: List[Slot]
    courses: List[Course]

class HardWeights(BaseModel):
    groupConflict: int = 1000
    teacherConflict: int = 1000
    roomConflict: int = 1000
    planSatisfaction: int = 500
    capacity: int = 200

class SoftWeights(BaseModel):
    windows: int = 10
    teacherLoadBalance: int = 5
    spread: int = 5
    preferences: int = 3
```

```
class ObjectiveWeights(BaseModel):
    hard: HardWeights = HardWeights()
    soft: SoftWeights = SoftWeights()

class TimetableGene(BaseModel):
    courseId: int
    roomId: int
    slotId: int

class Timetable(BaseModel):
    genes: List[TimetableGene]
    score: float
    meta: Dict = {}

class AlgoConfig(BaseModel):
    weights: ObjectiveWeights = ObjectiveWeights()

    populationSize: int = 60
    tournamentSize: int = 3
    crossoverRate: float = 0.9
    mutationRate: float = 0.05
    elitism: int = 1

    saInitialTemp: float = 5000.0
    saCooling: float = 0.96
    saMovesPerTemp: int = 150

    stagnationIters: int = 2
```

```
    adapt: Dict[str, float] = Field(default_factory=lambda: {
        "mutationBump": 0.05,
        "coolingRelax": 0.02,
        "softFocusFactor": 0.25
    })

class SolveRequest(BaseModel):
    dataset: Dataset
    algorithm: AlgorithmKind
    config: Optional[AlgoConfig] = None

class SolveStatus(BaseModel):
    id: str
    state: Literal['queued','running','succeeded','failed'] = 'queued'
    progress: int = 0
    bestScore: Optional[float] = None
    message: Optional[str] = None

def evaluate(dataset: Dataset, tt: Timetable, w: ObjectiveWeights) -> float:
    hard = 0
    soft = 0

    by_slot_room: Dict[Tuple[int,int], int] = {}
    by_slot_group: Dict[Tuple[int,int], int] = {}
    by_slot_teacher: Dict[Tuple[int,int], int] = {}
    placed_hours: Dict[int, int] = {}

    groups = {g.id: g for g in dataset.groups}
```

```
rooms = {r.id: r for r in dataset.rooms}
courses = {c.id: c for c in dataset.courses}

for g in tt.genes:
    c = courses[g.courseId]
    key_sr = (g.slotId, g.roomId)
    key_sg = (g.slotId, c.groupId)
    key_st = (g.slotId, c.teacherId)
    by_slot_room[key_sr] = by_slot_room.get(key_sr, 0) + 1
    by_slot_group[key_sg] = by_slot_group.get(key_sg, 0) + 1
    by_slot_teacher[key_st] = by_slot_teacher.get(key_st, 0) + 1
    placed_hours[c.id] = placed_hours.get(c.id, 0) + 1

    room = rooms[g.roomId]
    grp = groups[c.groupId]
    if room.capacity < grp.size:
        hard += w.hard.capacity

for v in by_slot_room.values():
    if v > 1: hard += (v-1) * w.hard.roomConflict
for v in by_slot_group.values():
    if v > 1: hard += (v-1) * w.hard.groupConflict
for v in by_slot_teacher.values():
    if v > 1: hard += (v-1) * w.hard.teacherConflict

for c in dataset.courses:
    done = placed_hours.get(c.id, 0)
    if done != c.hours:
```

```
            hard += abs(done - c.hours) * w.hard.planSatisfaction

    return 1_000_000 * hard + soft

def clone_tt(t: Timetable) -> Timetable:
    return Timetable(genes=[TimetableGene(**g.dict()) for g in t.genes], score=t.score,
      meta=dict(t.meta))

def random_timetable(dataset: Dataset) -> Timetable:
    genes = []
    for c in dataset.courses:
        for _ in range(c.hours):
            r = random.choice(dataset.rooms).id
            s = random.choice(dataset.slots).id
            genes.append(TimetableGene(courseId=c.id, roomId=r, slotId=s))
    return Timetable(genes=genes, score=math.inf)

async def solve_greedy(dataset: Dataset, cfg: AlgoConfig, on_progress):
    tt = Timetable(genes=[], score=math.inf)
    total = sum(c.hours for c in dataset.courses)
    done = 0
    for c in sorted(dataset.courses, key=lambda x: -x.hours):
        for _ in range(c.hours):
            placed = False
            for s in dataset.slots:
                for r in dataset.rooms:
                    tt.genes.append(TimetableGene(courseId=c.id, roomId=r.id, slotId=s.id))
                    score = evaluate(dataset, tt, cfg.weights)
```

```
                if score < 1e12:
                    placed = True; break
                tt.genes.pop()
            if placed: break
        if not placed:  # якщо не знайшли без-жорстких — все одно ставимо
    перший
            tt.genes.append(TimetableGene(courseId=c.id, roomId=dataset.rooms[0].id,
    slotId=dataset.slots[0].id))
        done += 1
        await on_progress(int(done/total*100), tt)
    tt.score = evaluate(dataset, tt, cfg.weights)
    await on_progress(100, tt)
    return tt

def ga_tournament(pop: List[Timetable], k: int) -> Timetable:
    best = random.choice(pop)
    for _ in range(k-1):
        cand = random.choice(pop)
        if cand.score < best.score:
            best = cand
    return best

def ga_crossover(a: Timetable, b: Timetable) -> Timetable:
    cut = random.randrange(len(a.genes))
    genes = a.genes[:cut] + b.genes[cut:]
    return Timetable(genes=[TimetableGene(**g.dict()) for g in genes],
      score=math.inf)
```

```
def ga_mutate(t: Timetable, dataset: Dataset, rate: float):
    for i in range(len(t.genes)):
        if random.random() < rate:
            if random.random() < 0.5:
                t.genes[i].roomId = random.choice(dataset.rooms).id
            else:
                t.genes[i].slotId = random.choice(dataset.slots).id

async def solve_ga(dataset: Dataset, cfg: AlgoConfig, on_progress):
    pop = [random_timetable(dataset) for _ in range(cfg.populationSize)]
    for t in pop: t.score = evaluate(dataset, t, cfg.weights)
    pop.sort(key=lambda x: x.score)
    best = clone_tt(pop[0])

    gens = 200
    for g in range(gens):
        next_pop: List[Timetable] = []

        for _ in range(cfg.elitism):
            next_pop.append(clone_tt(pop[_]))
        while len(next_pop) < cfg.populationSize:
            p1 = ga_tournament(pop, cfg.tournamentSize)
            p2 = ga_tournament(pop, cfg.tournamentSize)
            child = ga_crossover(p1, p2) if random.random() < cfg.crossoverRate else
      clone_tt(p1)
            ga_mutate(child, dataset, cfg.mutationRate)
            next_pop.append(child)
        for t in next_pop: t.score = evaluate(dataset, t, cfg.weights)
```

```
        next_pop.sort(key=lambda x: x.score)
        pop = next_pop
        if pop[0].score < best.score: best = clone_tt(pop[0])
        await on_progress(min(99, int((g/gens)*100)), best)
    await on_progress(100, best)
    return best

def sa_neighbor(dataset: Dataset, t: Timetable) -> Timetable:
    nxt = clone_tt(t)
    i = random.randrange(len(nxt.genes))
    if random.random() < 0.5:
        nxt.genes[i].slotId = random.choice(dataset.slots).id
    else:
        nxt.genes[i].roomId = random.choice(dataset.rooms).id
    return nxt

async def solve_sa(dataset: Dataset, cfg: AlgoConfig, on_progress):
    cur = random_timetable(dataset)
    cur.score = evaluate(dataset, cur, cfg.weights)
    best = clone_tt(cur)
    T = cfg.saInitialTemp
    steps = 200
    for step in range(steps):
        for _ in range(cfg.saMovesPerTemp):
            nxt = sa_neighbor(dataset, cur)
            nxt.score = evaluate(dataset, nxt, cfg.weights)
            dF = nxt.score - cur.score
            if dF < 0 or random.random() < math.exp(-dF / max(T, 1e-9)):
```

```
            cur = nxt
            if cur.score < best.score:
                best = clone_tt(cur)
        T *= cfg.saCooling
        await on_progress(min(99, int((step/steps)*100)), best)
    await on_progress(100, best)
    return best

def adapt_up(cfg: AlgoConfig) -> AlgoConfig:
    c = cfg.copy(deep=True)
    bump = c.adapt.get("mutationBump", 0.05)
    relax = c.adapt.get("coolingRelax", 0.02)
    focus = c.adapt.get("softFocusFactor", 0.25)

    c.mutationRate = min(0.5, c.mutationRate + bump)
    c.saCooling = min(0.999, c.saCooling + relax)
    sw = c.weights.soft
    c.weights.soft = SoftWeights(
        windows=int(sw.windows * (1 + focus)),
        teacherLoadBalance=int(sw.teacherLoadBalance * (1 + focus)),
        spread=int(sw.spread * (1 + focus)),
        preferences=int(sw.preferences * (1 + focus)),
    )
    return c

def adapt_down(base: AlgoConfig, cur: AlgoConfig) -> AlgoConfig:
    c = cur.copy(deep=True)
    c.mutationRate = c.mutationRate - 0.5 * (c.mutationRate - base.mutationRate)
```

```
    c.saCooling = c.saCooling - 0.5 * (c.saCooling - base.saCooling)
    c.weights = base.weights
    return c

async def solve_ahsa(dataset: Dataset, base_cfg: AlgoConfig, on_progress):
    cfg = base_cfg.copy(deep=True)

    best_ga = await solve_ga(dataset, cfg, lambda p,b: on_progress(int(p*0.6), b))
    best = await solve_sa(dataset, cfg, lambda p,b: on_progress(60 + int(p*0.3), b))

    stagn = 0
    last = best.score
    pulses = 10
    for _ in range(pulses):
        if best.score >= last - 1e-9: stagn += 1
        else: stagn = 0
        last = best.score

        cfg = adapt_up(cfg) if stagn >= cfg.stagnationIters else adapt_down(base_cfg,
      cfg)

        micro = await solve_ga(dataset, cfg.copy(update={"populationSize": max(10,
      cfg.populationSize//3)}),
                               lambda p,b: on_progress(90 + int(p*0.08), b))
        if micro.score < best.score: best = micro

        local = await solve_sa(dataset, cfg.copy(update={"saInitialTemp":
      cfg.saInitialTemp*0.5, "saMovesPerTemp": max(50, cfg.saMovesPerTemp//2)}),
```

```
                        lambda p,b: on_progress(98, b))
        if local.score < best.score: best = local
        await on_progress(100, best)
    return best

class JobEntry:
    def __init__(self):
        self.status = SolveStatus(id=str(uuid4()), state='queued', progress=0)
        self.result: Optional[Timetable] = None

JOBS: Dict[str, JobEntry] = {}

async def run_job(entry: JobEntry, payload: SolveRequest):
    entry.status.state = 'running'
    cfg = payload.config or AlgoConfig()

    async def on_progress(p: int, best: Timetable):
        entry.status.progress = p
        entry.status.bestScore = best.score

    try:
        if payload.algorithm == 'GREEDY':
            res = await solve_greedy(payload.dataset, cfg, on_progress)
        elif payload.algorithm == 'GA':
            res = await solve_ga(payload.dataset, cfg, on_progress)
        elif payload.algorithm == 'SA':
            res = await solve_sa(payload.dataset, cfg, on_progress)
        elif payload.algorithm == 'AHSA':
```

```
            res = await solve_ahsa(payload.dataset, cfg, on_progress)
        else:
            raise ValueError("Unknown algorithm")
        entry.result = res
        entry.status.state = 'succeeded'
        entry.status.progress = 100
        entry.status.bestScore = res.score
    except Exception as e:
        entry.status.state = 'failed'
        entry.status.message = str(e)

app = FastAPI(title="Timetabling API (AHSA/GA/SA/Greedy)")

@app.post("/solve")
async def solve(req: SolveRequest):
    entry = JobEntry()
    JOBS[entry.status.id] = entry

    asyncio.create_task(run_job(entry, req))
    return {"id": entry.status.id}

@app.get("/jobs/{job_id}")
async def job_status(job_id: str):
    entry = JOBS.get(job_id)
    if not entry: raise HTTPException(404, "Not found")
    return entry.status

@app.get("/jobs/{job_id}/result")
```

```
async def job_result(job_id: str):
    entry = JOBS.get(job_id)
    if not entry: raise HTTPException(404, "Not found")
    if entry.result is None: raise HTTPException(404, "Not ready or failed")
    return {"timetable": entry.result}
```