НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

«На правах рукопису»
УДК 004.421.2
004.4:519.8:378

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
«___»_____2025 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»
зі спеціальності 121 «Інженерія програмного забезпечення»
на тему: «Метод та програмне забезпечення розподілу навчального навантаження учня з урахуванням індивідуального графіку»

Виконав:
студент ІІ курсу, групи ІП-43мп
Курильченко Кирило Олегович _____
Керівник:
доцент кафедри ІПІ, к.т.н., доцент,
Лісовиченко Олег Іванович _____
Рецензент:
професор каф. ІСТ, д.т.н., професор
Корнага Ярослав Ігорович _____

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

Київ – 2025 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення інформаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Едуард ЖАРІКОВ

«___»_____2025р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Курильченко Кирило Олегович

1. Тема дисертації «Метод та програмне забезпечення розподілу навчального навантаження учня з урахуванням індивідуального графіку», науковий керівник дисертації к.т.н., доцент, Лісовиченко Олег Іванович, затверджені наказом по університету від «06» листопада 2025 р. № 4841-с

2. Термін подання студентом дисертації «15» грудня 2025 р.

3. Об’єкт дослідження – процеси планування та організації індивідуальної навчальної діяльності студента

4. Предмет дослідження – методи розподілу навчального навантаження між днями та архітектура програмного рішення/засобу

5. Перелік завдань, які потрібно розробити – аналіз проблеми та існуючих рішень; формалізувати задачу розподілу навчального навантаження;

розробити метод автоматизованого планування розкладу; реалізувати програмний засіб; проаналізувати отримані результати

6. Орієнтовний перелік графічного (ілюстративного) матеріалу – 3 плакати

7. Орієнтовний перелік публікацій – одні тези

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9.Дата видачі завдання «1» вересня 2025 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання	Примітка
1	Дослідження літератури	05.09.2025	
2	Огляд існуючих рішень	20.09.2025	
3	Аналіз методів	24.09.2025	
4	Проектування системи	07.10.2025	
5	Програмна реалізація системи	28.10.2025	
6	Тестування та виправлення помилок	03.11.2025	
7	Виконання експериментальних досліджень	06.11.2025	
8	Оформлення пояснювальної записки	20.11.2025	
9	Подання дисертації на попередній захист	26.11.2025	
10	Подання дисертації на захист	15.12.2025	

Студент Кирило КУРИЛЬЧЕНКО

Науковий керівник Олег ЛІСОВИЧЕНКО

РЕФЕРАТ

Розмір пояснювальної записки – **114 аркушів**, містить **14 ілюстрацій, 29 таблиць, 3 додатки, 21 посилання на джерела**.

Актуальність теми. У роботі розглянуто проблему планування індивідуального навчального навантаження студента за наявності великої кількості завдань з різними дедлайнами, тривалістю та обмеженим доступним часом. Показано, що поширені інструменти (календарі, списки завдань, трекери завдань, LMS) здебільшого лише фіксують дедлайни й нагадують про них, але не забезпечують автоматизоване побудову збалансованого розкладу з урахуванням добового ліміту навантаження, ризику зриву дедлайнів та нерівномірності роботи протягом семестру. Виявлено потребу в розробленні методу та програмного забезпечення, яке формує індивідуальний розклад навчальних завдань студента як результат оптимізаційної задачі з ресурсними обмеженнями.

Мета: Покращення ефективності планування навчального процесу студента.

Об’єктом дослідження є процеси планування та організації індивідуальної навчальної діяльності студента.

Предметом дослідження є методи розподілу навчального навантаження між днями та архітектура програмного рішення/засобу.

Для реалізації поставленої мети сформульовані наступні завдання:

- Проаналізувати проблему нерівномірного навчального навантаження студента та обмеження існуючих інструментів планування.
- Формалізувати задачу розподілу навчального навантаження як задачу побудови індивідуального розкладу з урахуванням дедлайнів, тривалості завдань і доступних годин студента.
- Розробити метод автоматизованого планування розкладу, який: розбиває великі завдання на часові блоки, розподіляє ці блоки по днях до дедлайнів без перевищення ліміту навантаження на день.

- Реалізувати програмний засіб системи планування навчального навантаження з інтерфейсом користувача.
- Проаналізувати отримані результати.

Наукова новизна роботи полягає у розробці методу розподілу навчального навантаження, який поєднує модифікований алгоритм EDD з введенням ефективного дедлайну (дедлайн мінус буфер), добовими обмеженнями на кількість годин та дробленням завдань на блоки, а також у використанні спеціалізованої локальної оптимізації для згладжування добового навантаження. Додатково запропоновано систему метрик оцінки якості індивідуального розкладу (максимальне навантаження, кількість перевантажених і «пікових» днів, дисперсія навантаження, завдання в зоні ризику), орієнтовану на комфортність та стійкість навчального процесу.

Практичне значення отриманих результатів полягає в тому, що:

- розроблено вебзастосунок – програмний засіб планування навчального навантаження студента, який реалізує запропонований метод, надає зручний інтерфейс для введення завдань, параметрів графіка доступності та візуалізації побудованого розкладу;
- реалізований програмний засіб може бути використаний окремим студентом для щоденного планування індивідуальної навчальної діяльності, а також потенційно інтегрований як модуль до існуючих систем підтримки навчального процесу (LMS, календарні сервіси, EdTech-платформи);
- застосунок може бути використаний у навчальному процесі як демонстраційний приклад застосування методів теорії розкладів, евристичних алгоритмів та веб-технологій у прикладній задачі планування навчальної діяльності.

Зв’язок з науковими програмами, планами, темами. Робота виконувалась на кафедрі інформатики та програмної інженерії Національного технічного

університету України "Київський політехнічний інститут імені Ігоря Сікорського".

Апробація. Наукові положення дисертації пройшли апробацію на IX Міжнародній науково-практичній конференції молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології» (SoftTech-2025).

Ключові слова: ПЛАНУВАННЯ НАВЧАЛЬНОГО НАВАНТАЖЕННЯ, ІНДИВІДУАЛЬНИЙ РОЗКЛАД, DEADLINE, EDD, ЛОКАЛЬНА ОПТИМІЗАЦІЯ, HILL CLIMBING, ВЕБЗАСТОСУНОК, EDTECH.

ABSTRACT

Explanatory note size – 114 pages, contains 14 figures, 29 tables, 3 appendices, 21 references.

Topicality. The thesis addresses the problem of planning an individual student's study load in the presence of a large number of tasks with different deadlines, durations and limited available time. It is shown that widely used tools (calendars, to-do lists, task trackers, LMS) mostly just record deadlines and send reminders, but do not provide automated construction of a balanced schedule that takes into account the daily workload limit, the risk of missing deadlines and the uneven distribution of work throughout the semester. This reveals the need for a method and software system that generates an individual schedule of a student's study tasks as the solution of a resource-constrained optimisation problem.

Aim. To improve the efficiency of planning a student's learning process.

Object of research. The processes of planning and organising an individual student's learning activities.

Subject of research. Methods of distributing the student's study load between days and the architecture of the software solution/tool that implements such methods.

To achieve this aim, the following tasks were formulated:

- to analyse the problem of uneven student study load and the limitations of existing planning tools;
- to formalise the study-load distribution problem as the problem of constructing an individual schedule that takes into account task deadlines, durations and the student's available hours;
- to develop a method for automated schedule construction that splits large tasks into time blocks and allocates these blocks to days before their deadlines without exceeding the daily workload limit;
- to implement a software tool for planning the student's study load with a user interface;
- to analyse the obtained results.

Scientific novelty. The scientific novelty of the thesis lies in the development of a study-load distribution method that combines a modified EDD algorithm with the introduction of an effective deadline (deadline minus buffer), daily constraints on the number of hours and splitting tasks into blocks, as well as in the use of a specialised local optimisation procedure to smooth the daily workload. In addition, a system of metrics for evaluating the quality of an individual schedule is proposed (maximum workload, number of overloaded and "peak" days, workload variance, tasks in the risk zone), focused on the comfort and robustness of the learning process.

Practical significance. The practical value of the obtained results is as follows:

- a web application – a software tool for planning a student's study load – has been developed; it implements the proposed method and provides a convenient interface for entering tasks, availability parameters and visualising the resulting schedule;
- the implemented software tool can be used by an individual student for daily planning of personal learning activities, and can potentially be integrated as a module into existing educational support systems (LMS, calendar services, EdTech platforms);
- the application can be used in the educational process as a demonstrational example of applying scheduling theory, heuristic algorithms and web technologies to a practical study-planning task.

Relationship with scientific programs, plans and topics. The work was carried out at the Department of Computer Science and Software Engineering of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute".

Approbation. The scientific results of the thesis were approbated at the IX International Scientific and Practical Conference of Young Scientists and Students "Software Engineering and Advanced Information Technologies" (SoftTech-2025).

Keywords: STUDY LOAD PLANNING, INDIVIDUAL SCHEDULE, DEADLINE, EDD, LOCAL OPTIMISATION, HILL CLIMBING, WEB APPLICATION, EDTECH.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення;

БД – база даних;

LMS – система управління навчанням (Learning Management System) для організації дистанційного навчання та видачі завдань;

EdTech – освітні технології (Education Technology), сукупність цифрових сервісів і платформ для підтримки навчання;

AI – штучний інтелект (Artificial Intelligence), напрям інформатики, що розробляє методи й системи для виконання інтелектуальних задач;

EDD – Earliest Due Date, евристика розкладання, у якій завдання впорядковують за зростанням дедлайнів;

EDF – Earliest Deadline First, алгоритм планування, що в кожен момент обирає до виконання завдання з найранішим дедлайном;

SPT – Shortest Processing Time first, евристика розкладання, у якій завдання впорядковують за зростанням тривалості виконання;

WSPT – Weighted Shortest Processing Time first, евристика розкладання з урахуванням ваг, що мінімізує зважену суму запізнень;

LPT – Largest Processing Time first, евристика розкладання, у якій завдання впорядковують за спаданням тривалості виконання;

NP – клас задач, розв’язання яких можна перевірити за поліноміальний час (Non-deterministic Polynomial time), до якого належать багато задач розкладання;

SPA – односторінковий вебзастосунок (Single Page Application), у якому основне завантаження інтерфейсу відбувається один раз у браузері;

PWA – прогресивний вебзастосунок (Progressive Web App), вебдодаток, що підтримує офлайн-режим, установку на пристрій та інші можливості, подібні до нативних застосунків;

UI – користувацький інтерфейс (User Interface), сукупність екранів, форм і елементів взаємодії користувача із системою;

API – прикладний програмний інтерфейс (Application Programming Interface), узгоджений набір методів для взаємодії компонентів програмної системи або зовнішніх сервісів;
HTTP – протокол передавання гіпертексту (HyperText Transfer Protocol), базовий протокол взаємодії клієнта й вебсервера;
JSON – JavaScript Object Notation, текстовий формат обміну структурованими даними «ключ–значення»;
ICS – текстовий формат файлів електронного календаря iCalendar (.ics) для обміну подіями та розкладами між застосунками;
ISO – International Organization for Standardization; у роботі використовується формат дат за стандартом ISO (наприклад, ISO 8601, «YYYY-MM-DD»);
CRUD – набір базових операцій над даними: Create (створення), Read (читання), Update (оновлення), Delete (видалення);
CSS – каскадні таблиці стилів (Cascading Style Sheets), мова опису зовнішнього вигляду вебсторінок;
B2C – модель взаємодії «business-to-consumer», коли продукт орієнтований на індивідуальних кінцевих користувачів;
B2B – модель взаємодії «business-to-business», коли клієнтами є організації (університети, освітні платформи тощо);
SWOT – метод стратегічного аналізу сильних (Strengths) і слабких (Weaknesses) сторін, можливостей (Opportunities) і загроз (Threats) проєкту або продукту;
CFA – міжнародна сертифікаційна програма Chartered Financial Analyst; у роботі використовується як приклад складного сертифікаційного іспиту при аналізі «розумних» планерів підготовки.

ВСТУП

Навчальний процес у закладах вищої освіти супроводжується значною кількістю паралельних завдань: лабораторні та практичні роботи, індивідуальні завдання, модульні контролі, курсові проєкти, пояснювальні записки тощо. Кожне з них має власний дедлайн і трудомісткість, тоді як час студента обмежений розкладом занять, додатковою роботою та особистими справами. За відсутності системного планування це призводить до нерівномірного навантаження: накопичення завдань наприкінці семестру, «пікових» днів із надмірним обсягом роботи, зривів дедлайнів і підвищеного стресу.

Поширені засоби організації навчальної діяльності (календарі, списки справ, таск-трекери) здебільшого лише фіксують дедлайни й нагадують про них. Вони майже не враховують доступний час студента у кожен окремий день і не контролюють, чи не формують введені завдання нереалістичний графік. У наявних рішеннях зазвичай відсутній явний добовий ліміт навантаження, буфер до дедлайнів та оцінка рівномірності розкладу за спеціальними метриками. Це зумовлює актуальність задачі розробки методу та програмного інструмента, які не просто зберігають список завдань, а й автоматично розподіляють навчальне навантаження в часі з урахуванням індивідуального графіку студента.

Мета: Покращення ефективності планування навчального процесу студента

Об’єктом дослідження є процеси планування та організації індивідуальної навчальної діяльності студента

Предметом дослідження є методи розподілу навчального навантаження між днями та архітектура програмного рішення/засобу

Для досягнення поставленої мети необхідно розв’язати такі **основні завдання**:

- Проаналізувати проблему нерівномірного навчального навантаження студента та обмеження існуючих інструментів планування
- Формалізувати задачу розподілу навчального навантаження як задачу побудови індивідуального розкладу з урахуванням дедлайнів, тривалості завдань і доступних годин студента.

- Розробити метод автоматизованого планування розкладу, який: розбиває великі завдання на часові блоки, розподіляє ці блоки по днях до дедлайнів без перевищення ліміту навантаження на день.
- Реалізувати програмний засіб системи планування навчального навантаження з інтерфейсом користувача.
- Проаналізувати отримані результати

Наукова новизна роботи полягає у розробці методу розподілу навчального навантаження, який поєднує модифікований алгоритм EDD з введенням ефективного дедлайну (дедлайн мінус буфер), добовими обмеженнями на кількість годин та дробленням завдань на блоки, а також у використанні спеціалізованої локальної оптимізації для згладжування добового навантаження. Додатково запропоновано систему метрик оцінки якості індивідуального розкладу (максимальне навантаження, кількість перевантажених і «пікових» днів, дисперсія навантаження, завдання в зоні ризику), орієнтовану на комфортність та стійкість навчального процесу.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1. Контекст проблеми та актуальність

Сучасний студент одночасно виконує велику кількість різнорідних навчальних завдань: лабораторні роботи, індивідуальні та курсові проєкти, модульні й семестрові контролі, підготовку до заліків й іспитів, виконання тестів у системах дистанційного навчання. Ці активності часто накладаються між собою в межах одного тижня або навіть кількох днів, мають різні дедлайни та трудомісткість. На практиці це призводить до типового сценарію «усе в останній день», коли значна частина роботи виконується напередодні строку здачі, а навантаження в окремі дні різко зростає.

Поширені інструменти організації роботи — календарі, електронні щоденники, таск-трекери та системи LMS — переважно фіксують факт наявності завдання і його кінцевий строк. Вони надсилають нагадування про дедлайн, але не відповідають на базове для студента питання: *скільки годин потрібно попрацювати сьогодні, скільки — завтра, і в які саме дні реально встигнути виконати усі поточні задачі*. Відсутність такого поденного планування означає, що студент змушений інтуїтивно «розкидати» час між предметами, спираючись на власне відчуття терміновості, яке часто виявляється неточним [18].

Наслідком є нерівномірне навчальне навантаження: частина днів залишається відносно «порожньою», тоді як напередодні дедлайнів виникають пікові перевантаження. У такі періоди студент намагається одночасно закінчити кілька великих робіт, працює понад комфортний добовий ліміт часу, що призводить до втоми, зниження якості виконання та зриву строків. У довгостроковій перспективі це створює ефект «снігової кулі», коли відкладені завдання накопичуються, а режим постійних авралів підвищує ризик емоційного вигорання.

Паралельно в університетському управлінні вже давно відбувається активна автоматизація процесів планування. Для формування розкладів занять, розподілу аудиторного фонду та навантаження викладачів застосовуються формальні моделі й алгоритми оптимізації. Задачі розкладання (timetabling, scheduling) розглядаються як повноцінні обчислювальні задачі з обмеженими ресурсами, для яких розробляються спеціалізовані методи та евристики. Тобто на рівні закладу освіти ідея «оптимізувати розклад» уже давно сприймається як природна.

Водночас індивідуальний навчальний графік студента зазвичай залишається сферою ручного планування. Навіть за наявності цифрових інструментів основна робота — оцінити обсяг завдань, розкласти їх по днях з урахуванням доступного часу, збалансувати навантаження та передбачити ризики зриву дедлайнів — покладається на самого студента. Ринок EdTech поступово пропонує рішення у вигляді «розумних» планерів та AI-асистентів, однак більшість з них або зосереджені на підготовці до окремого іспиту, або працюють як вдосконалений календар нагадувань, не виконуючи повноцінного оптимізаційного перерозподілу задач [19; 20].

Отже, тема **автоматизованого розподілу навчального навантаження студента з урахуванням індивідуального графіку доступності** є актуальною з кількох причин. По-перше, вона відповідає реальній потребі студентів у інструментах, які допомагають не лише «пам’ятати про дедлайни», а й щоденно планувати обсяг роботи без перевантажень. По-друге, вона продовжує загальну тенденцію застосування методів теорії розкладів та оптимізації в освітніх процесах, але переносить їх на рівень особистого навчального графіка. По-третє, подібні рішення мають помітний стартап-потенціал на ринку освітніх технологій, де зростає попит на персоналізовані сервіси підтримки навчання та управління часом. Тому розроблення методу і програмного забезпечення для побудови збалансованого індивідуального розкладу навчальної роботи є як науково, так і практично значущою задачею.

1.2. Проблема нерівномірного навчального навантаження та зриву дедлайнів

Однією з ключових причин зриву дедлайнів у студентів є не стільки сумарний обсяг навчальних завдань, скільки те, **як саме цей обсяг розподілений у часі**. Типова ситуація виглядає так: великі завдання (розгорнуті лабораторні, підготовка до модульного контролю, написання пояснювальної записки чи курсового проєкту) відкладаються «на потім» і фактично виконуються в один-два останні дні перед терміном здачі. У результаті окремі дні тижня виявляються критично перевантаженими: після пар студент змушений працювати ще 5–6 годин поспіль, що неминуче веде до перевтоми, падіння якості виконання роботи та прострочення дедлайнів.

У термінах теорії розкладів така ситуація проявляється як накопичення **запізнення** та **прострочення**: коли кілька завдань одночасно «виїжджають» за межі своїх дедлайнів через пікові навантаження наприкінці тижня чи навчального циклу. Чим вищі ці піки, тим більша сумарна кількість завдань, що завершуються із запізненням, навіть якщо загалом студент витрачає на навчання «достатньо годин». Проблема полягає не в небажанні працювати, а у відсутності збалансованого графіка, який би заздалегідь розкладав великі завдання на менші блоки і розподіляв їх у відносно вільні дні.

Другий аспект проблеми — **брак прогнозування**. Більшість студентів не бачить заздалегідь, що, наприклад, «день Х» через два тижні вже зараз перевантажений сукупністю дедлайнів із кількох дисциплін. Немає механізму раннього попередження на кшталт: «якщо не почати працювати вже цього тижня, у вказану дату необхідний обсяг годин буде недосяжним». Без такого прогнозу навантаження накопичується «невидимо», а момент кризи настає раптово — у вигляді кількох паралельних дедлайнів в один день.

Це породжує **ефект “снігової кулі”**: якщо у вівторок студент не встиг завершити частину запланованої роботи, ця частина переноситься на середу, де

вже заплановані інші завдання. Далі відставання наростає лавиноподібно — друга половина тижня перетворюється на безперервний аврал, а частина завдань узагалі перестає потрапляти до реалізовуваного розкладу. У проєктному менеджменті давно показано, що відсутність контролю піків завантаження й обмежень за ресурсами неминуче веде до режиму постійних авралів; у навчальному контексті ситуація є аналогічною.

Існуючі інструменти, якими користується студент щодня (календарі, to-do-списки, таск-трекери), по суті лише **фіксують дедлайни й перелік завдань**, але не дають відповіді на питання «скільки годин і в який день потрібно попрацювати, щоб реально встигнути». Вони не враховують добовий ліміт доступного часу, не забезпечують автоматичного дроблення великих завдань на послідовні блоки, не намагаються згладжувати піки навантаження й не допомагають відновити план після зриву окремих днів. Таким чином, **нерівномірне навчальне навантаження** є не лише наслідком «самоорганізації» студента, а передусім результатом відсутності інструментів, які б будували збалансований, ресурсно-обмежений розклад на основі формальної оптимізації, а не ручних інтуїтивних рішень [20; 21].

1.3. Класичні підходи та евристики (EDD, EDF, WSPT/LPT)

У класичній теорії розкладів послідовність робіт описують через кілька базових параметрів: тривалість виконання p_j (processing time), крайній термін / дедлайн d_j (due date), інколи – вагу або важливість w_j. Якість розкладу зазвичай оцінюють за певною цільовою функцією: мінімізація запізнення або прострочення (lateness / tardiness), мінімізація максимальної затримки чи сумарного часу завершення робіт, обмеження пікового навантаження на ресурс тощо [1; 5].

Для побудови «початкового» розкладу широко застосовують прості жадібні правила впорядкування завдань. Одне з найвідоміших – **правило**

Earliest Due Date (EDD): завдання виконуються у порядку зростання дедлайнів, тобто першими обираються задачі з найранішими строками. Це правило оптимальне для ряду класичних постановок (наприклад, мінімізація максимального запізнення на одній машині) і часто використовується як базовий орієнтир у прикладних задачах планування. У контексті навчального навантаження інтерпретація EDD інтуїтивна: «те, що горить раніше, має опинитися в черзі першим».

Близьким за ідеєю до EDD є підхід **Earliest Deadline First (EDF)**, який активно застосовується в теорії планування реального часу. У класичній формі EDF припускає, що задачі можна переривати, а система завжди обирає до виконання ту задачу, у якої найменший поточний дедлайн. У схемі одно-ресурсного розкладу це знову ж таки приводить до впорядкування за датами завершення, але вже з акцентом на динамічність: по мірі надходження нових задач розклад може перебудовуватись, а пріоритет переназначається в реальному часі.

Окремий клас евристик базується не на дедлайнах, а на **тривалості робіт**. Найпростіше правило — SPT (Shortest Processing Time first), коли першими виконуються найкоротші завдання. Його ваговою модифікацією є **WSPT (Weighted Shortest Processing Time)**: задачі впорядковуються за зростанням відношення $\frac{p_j}{w_j}$, тобто пріоритет надається роботам із високою «корисністю на одиницю часу». У ситуації студента це означало б віддавати перевагу завданням, які суттєво впливають на підсумкову оцінку, але відносно швидко виконуються. З іншого боку, правило **LPT (Longest Processing Time first)** пропонує спочатку брати найдовші задачі; воно використовується, коли важливо якнайшвидше «розвантажити» ресурс від великих робіт і не залишати їх на кінець періоду.

Загальна ідея таких евристик полягає в тому, щоб за простим локальним критерієм (найближчий дедлайн, найкоротша/найдовша тривалість, найкраще

співвідношення «вага/час») побудувати початковий, обчислювально дешевий розклад, який у низці випадків є оптимальним або близьким до оптимального. Для складніших задач, де потрібно враховувати багато обмежень, цей первинний план потім доопрацьовують методами локальної оптимізації (hill climbing, simulated annealing тощо).

Однак усі перелічені правила у «чистому» вигляді мають важливі обмеження для нашого сценарію. По-перше, вони, як правило, **не враховують обмеження на добовий ресурс**: машина в класичній постановці може працювати скільки завгодно, тоді як студент має фіксований ліміт годин на день. По-друге, в базових моделях завдання часто розглядаються як неподільні, тоді як великі навчальні роботи (курсові, проєкти) потрібно **розбивати на декілька блоків** і планувати їх у різні дні. По-третє, стандартні правила не оперують поняттями «комфортного навантаження» чи «пікових перевантажень» – їхні цілі зазвичай зводяться до мінімізації запізнень, а не до вирівнювання навантаження в часі. Саме ці обмеження класичних евристик мотивують подальшу модифікацію правила EDD та поєднання його з локальною оптимізацією, що буде детально розглянуто в наступних розділах.

1.4. Автоматизація навчального розкладу в закладах освіти

У попередніх підрозділах увага була зосереджена переважно на проблемах індивідуального планування навчальної роботи студента та інструментах особистої організації часу. У цьому підрозділі фокус переноситься на протилежний “полюс” освітньої системи — автоматизацію розкладу на рівні закладу освіти (школи, коледжу, університету), де розв’язується задача побудови глобального розкладу занять для сотень груп, десятків викладачів та обмеженої кількості аудиторій.

Традиційно складання навчального розкладу виконувалося вручну: адміністратори або методисти формували таблиці, намагаючись «утиснути» усі дисципліни в наявні часові слоти, не допустити перетинів для груп та

викладачів, врахувати санітарні норми, поєднати лекції та практичні заняття тощо. Такий підхід є надзвичайно трудомістким і погано масштабується: будь-яка зміна (звільнення викладача, відкриття нової групи, ремонт аудиторії) вимагає фактично ручного перерахунку значної частини розкладу. Саме тому в літературі та практиці управління освітою сформувався окремий клас задач — **timetabling**, або задача складання розкладу занять, яку розглядають як повноцінну оптимізаційну проблему [7].

Суть задачі навчального розкладу на рівні закладу полягає в одночасному врахуванні великої кількості обмежень. До **жорстких** обмежень належать: відсутність перетинів курсів для однієї групи, неможливість викладання одним викладачем у двох аудиторіях одночасно, обмежена місткість приміщень, обов’язкові часові вікна для певних дисциплін (наприклад, лабораторії, прив’язані до роботи спеціалізованих лабораторій чи обладнання). **М’які** обмеження пов’язані зі зручністю та якістю навчального процесу: уникнення «вікон» у розкладі, нерівномірного навантаження по днях, небажаних ранніх або пізніх занять, чергування складних та простіших дисциплін упродовж дня.

З формальної точки зору така постановка швидко призводить до комбінаторного вибуху: кількість можливих варіантів розкладу зростає експоненційно зі збільшенням кількості груп, предметів і часових слотів. Тому сучасні дослідження прямо відносять задачу навчального розкладу до класу NP-складних: знайти математично оптимальний розклад для реального університету практично неможливо, натомість шукають **достатньо хороше** рішення за прийнятний час. Це обґрунтовує застосування евристик та метаевристик — жадібних стратегій, локального пошуку, генетичних алгоритмів, імітації відпалу, табу-пошуку тощо, які дозволяють поступово покращувати якість розкладу, рухаючись від початкового наближення до кращих конфігурацій [3; 7; 8].

Університетські й шкільні системи розкладання, описані в сучасній літературі, демонструють перехід від простих «генераторів сітки» до складних

оптимізаційних модулів. У перших реалізаціях автоматизація часто обмежувалася допоміжними інструментами: система перевіряла розклад, складений людиною, на наявність конфліктів і підсвічувала проблемні місця. Натомість нові системи самі **генерують** варіанти розкладу, оцінюють їх за кількома критеріями (кількість конфліктів, кількість порушених побажань, рівномірність завантаження аудиторій, збалансованість навантаження викладачів) і обирають найкращий з точки зору заданої функції якості [12].

Окремим напрямом розвитку стали рішення, орієнтовані на **персоналізацію навчальних траєкторій**. Такі системи не лише будують глобальний розклад занять для закладу, а й намагаються сформувати для кожного студента індивідуальний набір курсів та зручні часові слоти. У наукових працях описуються моделі, де студент подає перелік бажаних дисциплін, а система за допомогою еволюційних алгоритмів або інших метаевристик підбирає комбінацію груп, так щоб:

- не було перетинів у часі;
- було дотримано обмеження щодо обов’язкових та вибіркових компонент;
- максимізувалася «зручність» розкладу (мінімум «вікон», небажаних слотів, тривалих переїздів між кампусами тощо).

Сучасні платформені рішення для університетів (у тому числі з використанням штучного інтелекту) додають до цього ще й **динамічну адаптацію**: можливість автоматично перерахувати розклад при зміні вхідних даних (відміна пари, форс-мажор у викладача, переведення курсу в онлайн-формат). У деяких системах планувальник може враховувати побажання студентів щодо часу занять (наприклад, «не раніше 9:00» або «не більше трьох пар підряд»), що наближає інституційний розклад до більш гнучкої й «людиноцентричної» моделі.

Попри значний прогрес у цій сфері, більшість розглянутих рішень залишаються сфокусованими на **організаційних ресурсах закладу**: аудиторіях, викладачах, групах, курсах. Їхній головний критерій — забезпечити коректне

використання матеріальної та кадрової бази, уникнути конфліктів та «дір» у розкладі, підвищити ефективність планування з точки зору адміністрації. Індивідуальна ситуація окремого студента враховується переважно через призму вибору курсів і часових слотів, але не через призму дрібніших щоденних активностей (підготовка до семінару, виконання домашніх завдань, робота над проєктом тощо).

Таким чином, автоматизація навчального розкладу на рівні закладу освіти підтверджує два ключові моменти, важливі для магістерської роботи. По-перше, **побудова "якісного" розкладу** — це визнана у науковій та практичній площині оптимізаційна задача, для якої розробляються спеціалізовані алгоритми і метаевристики, а не «ручне мистецтво» окремих методистів. По-друге, навіть найбільш просунуті інституційні системи працюють переважно з **глобальним розкладом** (пари, курси, аудиторії), тоді як **персональний розклад дрібних навчальних задач студента** (що саме і коли робити між парами, у вечірній час, у вихідні) залишається поза сферою автоматизованого планування. Саме цю «незаповнену нішу» покликана закрити система, що пропонується в даній магістерській роботі [6; 7; 9].

1.5. Персоналізовані планери навчання та динамічне розкладання (EdTech)

Другий важливий напрям розвитку рішень у сфері планування навчання — це **персоналізовані планери навчання** та **динамічні планувальники ("dynamic study schedulers")**, орієнтовані вже не на глобальний розклад університету, а на **індивідуальний графік конкретного студента**. Такі сервіси позиціонують себе як «personal study planner», «AI study plan generator» тощо і обіцяють побудувати персональний план підготовки до іспиту або опанування курсу на основі даних, які вводить користувач. Зазвичай від студента вимагається вказати перелік предметів або тем, дедлайни іспитів чи тестів, загальний обсяг матеріалу та кількість годин, які він реально може приділяти

навчанню щотижня. Після цього система автоматично розбиває матеріал на менші щоденні блоки й формує покроковий розклад занять [14].

Типовий опис функціональності таких планерів включає можливість:

- прийняти від користувача **список предметів/тем, дедлайни, обсяг матеріалу**;
- врахувати **доступні години** (наприклад, «тільки вечори по буднях» або «2 години у вихідні»);
- побудувати **помодульний або поденний план** занять;
- **автоматично підлаштовувати план**, якщо студент пропустив заплановану сесію або змінив свій графік.

Існують окремі приклади **динамічних планувальників** для підготовки до складних сертифікаційних іспитів (на кшталт CFA). У таких системах користувач задає стартову дату, кінцеву дату і кількість доступних годин на тиждень, після чого сервіс:

- автоматично **розкладає матеріал по днях**,
- задає проміжні контрольні точки прогресу,
- **перебудовує план**, якщо реальний темп відстає від ідеального або якщо студент пропустив кілька сесій.

По суті, це вже не статичний календар, а **адаптивний графік**, що сам перерозподіляє навантаження при зміні графіку («життя втрутилось») і зменшує ризик накопичення завдань у кінці підготовчого періоду. Важливо, що такі системи намагаються планувати не лише «до певної дати», а й **у конкретні часові вікна**, коли студент дійсно може вчитись.

Окрему підкатегорію складають **адаптивні або «розумні» планувальники навчання**, які прямо декларують використання підходів штучного інтелекту. Вони не тільки будують первинний план, а й **оновлюють його у реальному часі**, аналізуючи прогрес студента, складність матеріалу та зміни у розкладі. Якщо певна тема виявляється складнішою і потребує більше годин, система збільшує для неї обсяг часу та переносить інші блоки на пізніші

дні; якщо ж студент систематично не встигає виконувати заплановані сесії, план перебудовується так, щоб уникнути подальших перевантажень. У літературі та маркетингових матеріалах таких продуктів це подають як реалізацію концепції **dynamic scheduling** — переходу від статичного плану до постійно адаптивного графіка [9].

Для EdTech-ринку ці рішення важливі тим, що вони визнають **необхідність персоналізації навчального графіку**: врахування індивідуальної пропускної здатності студента (доступних годин), уникнення пікових навантажень та підтримки мотивації. Частина таких систем прямо декларує мету **попередити вигорання**, розкладаючи матеріал так, щоб не було надмірних «піків» в окремі дні, а навантаження було більш рівномірним упродовж усього періоду підготовки.

Водночас у наявних персоналізованих планерах можна виділити кілька принципових обмежень, важливих для даної роботи. По-перше, більшість таких сервісів орієнтовані на **одну велику ціль** (підготовка до конкретного іспиту, завершення одного курсу), тоді як реальний студент паралельно виконує **десятки різнорідних завдань** з різних дисциплін. По-друге, хоча у рекламних описах часто згадується «AI» та «динамічний графік», внутрішні механізми планування не завжди явно формулюються як **оптимізаційна задача розкладу з ресурсними обмеженнями**: добовий ліміт годин, буфер до дедлайнів, формальні метрики якості розкладу зазвичай не задаються явним чином. По-третє, такі продукти рідко надають прозорі метрики на кшталт «кількість перевантажених днів», «дисперсія добового навантаження», «кількість задач із ризиком прострочки», які дозволили б об’єктивно оцінювати та порівнювати якість побудованого розкладу [14-16].

Таким чином, персоналізовані планери навчання та динамічні планувальники вже демонструють, що **персоналізація та адаптивність** є стійким трендом в EdTech. Вони підтверджують попит на інструменти, які враховують індивідуальний графік і щоденну пропускну здатність студента.

Однак наявні рішення або працюють у вузькому сценарії (один курс/іспит), або не розглядають задачу як повноцінну **формальну задачу побудови розкладу** з чітко заданими ресурсними обмеженнями та цільовими метриками. Це створює нішу для підходу, запропонованого в даній роботі, де метод розподілу навчального навантаження формулюється і досліджується саме як задача оптимізації індивідуального розкладу.

1.6. Обмеження наявних підходів та вимоги до програмного забезпечення

Попередній аналіз показав, що більшість інструментів, які сьогодні використовує студент для організації навчання (календарі, таск-трекери, списки справ, окремі модулі LMS), фокусуються переважно на фіксації дедлайнів і переліку завдань, але не розглядають планування навчальної роботи як задачу оптимізації з обмеженим ресурсом часу студента [18].

У класичній теорії розкладів завдання формулюється як розподіл множини робіт із відомою тривалістю і дедлайнами по часовій шкалі за умови, що є обмежений ресурс (машина, виконавець), який не може виконувати більше однієї роботи одночасно; метою є мінімізація певного критерію якості (запізнення, максимального навантаження тощо). У контексті навчального процесу студентської роботи цю постановку природно інтерпретувати так: навчальне завдання відповідає окремій «роботі», її тривалість — очікуваному часу виконання, дедлайн — крайній термін здачі/перевірки, а ресурсом виступає час конкретного студента, який він реально може витратити на навчання протягом кожного дня.

Однак жоден із популярних інструментів планування не моделює доступний час студента як жорстке обмеження ресурсу. На практиці це означає, що користувач може безконтрольно додавати нові задачі в список або нові «блоки часу» в календар, і система не попереджає, що внаслідок цього окремі дні стають нереалістично перевантаженими (наприклад, коли після декількох пар поспіль студенту потрібно ще 5–6 годин безперервної роботи).

Окрема проблема стосується роботи з великими завданнями — курсовими, проєктами, об’ємними лабораторними роботами. Типові календарі й трекери оперують таким завданням як єдиним блоком, прив’язаним до однієї дати дедлайну. Відсутній вбудований механізм розбиття таких завдань на дрібні послідовні підзадачі, які можна рівномірно розкласти по днях до дедлайну з урахуванням добового ліміту часу. Це суперечить рекомендаціям як класичної теорії планування (розбиття робіт для вирівнювання навантаження), так і сучасних EdTech-підходів, де дроблення матеріалу на щоденні блоки є стандартною практикою.

Таким чином, наявні рішення не виконують одночасно три ключові умови, які є критичними саме для індивідуального навчального графіка студента [13]:

- **Урахування реальних часових обмежень студента по кожному дню.** Система має працювати не з абстрактним «тижневим навантаженням», а з конкретними годинами, які студент може виділити в понеділок, вівторок тощо, з урахуванням пар, роботи, дороги, відпочинку.
- **Балансування навантаження між днями, а не лише «встигнути будь-якою ціною».** План має мінімізувати пікові навантаження, кількість перевантажених днів і розкид навантаження по календарю, а не просто гарантувати факт виконання завдань до дедлайну.

Ці дві умови, виділені в ході аналізу, фактично задають «розрив» між теоретичною постановкою задачі розкладання та поведінкою реальних інструментів, і водночас формують ядро вимог до нового методу та програмного забезпечення.

На підставі виявлених обмежень до програмного забезпечення для підтримки індивідуального графіку навчання висуваються такі основні вимоги.

- **Розширена модель даних про завдання.**
  Система повинна зберігати для кожного навчального завдання принаймні: назву, дедлайн (дата/час), орієнтовну тривалість у годинах, а

також підтримувати розбиття на підзадачі/часові блоки. Така структура безпосередньо відповідає формальній постановці задачі розкладу в теорії планування робіт і дозволяє застосовувати алгоритми на кшталт EDD, локального пошуку тощо.

- **Модель доступності часу студента.**
  На відміну від класичних LMS, програмний засіб має працювати із реальним календарем доступності: години, коли студент завідомо не може навчатися (заняття, робота, дорога), та інтервали потенційної навчальної активності. У моделі розкладу це виступає обмеженням ресурсу — неможливістю запланувати більше годин, ніж студент фізично може виконати в конкретний день.
- **Алгоритмічне ядро для побудови збалансованого розкладу.**
  ПЗ повинно реалізовувати формальний метод планування, який: враховує дедлайни, тривалості та добовий ліміт часу; розбиває великі завдання на блоки й розподіляє їх по днях з буфером k днів до дедлайну; мінімізує критерії пікового та нерівномірного навантаження (кількість перевантажених днів, максимальне навантаження, дисперсія).
  Конкретна реалізація (модифікований EDD у поєднанні з локальною оптимізацією) детально розглядається в наступному розділі, але на рівні вимог важливо закласти сам факт алгоритмічної, а не суто ручної побудови розкладу.
- **Механізми візуалізації та інтерпретації навантаження.**
  Програмний засіб має не лише формувати розклад, але й наочно відображати його властивості: добове навантаження в годинах, позначення перевантажених днів, виділення завдань із ризиком прострочки, базові метрики (максимальне навантаження, кількість перевантажених днів, дисперсія, кількість задач, що не потрапили в план). Це перетворює систему з «чорної скриньки» на інструмент підтримки прийняття рішень для самого студента.

- **Вимоги до зручності використання.**

  Інтерфейс має бути інтуїтивно зрозумілим, простим у використанні й налагодженні: введення завдань і доступності не повинно вимагати від студента значних зусиль; операції генерації та оновлення розкладу мають виконуватися в кілька зрозумілих кроків. Це узгоджується з загальними вимогами до програмних продуктів магістерського рівня, наведеними в шаблоні пояснювальної записки.

У сукупності наведені обмеження та вимоги окреслюють розрив між поточним станом інструментів планування навчального навантаження та цільовою функціональністю розроблюваного програмного засобу. У подальших розділах ці вимоги конкретизуються через формальну постановку задачі, вибір методу (модифікований EDD + локальна оптимізація) і опис архітектури системи, що їх реалізує.

1.7. Постановка задачі

Метою роботи є покращення ефективності планування навчального процесу студента

Об’єктом дослідження є процеси планування та організації індивідуальної навчальної діяльності студента

Предметом дослідження є методи розподілу навчального навантаження між днями та архітектура програмного рішення/засобу

Для досягнення мети необхідно вирішити наступні задачі:

- Проаналізувати проблему нерівномірного навчального навантаження студента та обмеження існуючих інструментів планування
- Формалізувати задачу розподілу навчального навантаження як задачу побудови індивідуального розкладу з урахуванням дедлайнів, тривалості завдань і доступних годин студента.

- Розробити метод автоматизованого планування розкладу, який: розбиває великі завдання на часові блоки, розподіляє ці блоки по днях до дедлайнів без перевищення ліміту навантаження на день.
- Реалізувати програмний засіб системи планування навчального навантаження з інтерфейсом користувача.
- Аналіз отриманих результатів

Створений програмний продукт повинен відповідати таким вимогам:

- забезпечувати введення навчальних завдань із зазначенням назви, опису, орієнтовної тривалості та дедлайну;
- дозволяти задавати індивідуальний графік доступності студента по днях (доступні години) та його бажаний добовий ліміт навчального навантаження;
- автоматично будувати розклад виконання завдань у вигляді розподілу часових блоків по днях з урахуванням доступного часу, ліміту навантаження та буфера до дедлайнів;
- мати простий та зрозумілий інтерфейс для роботи зі списком завдань і отриманим розкладом, щоб студент міг користуватися системою без зайвих налаштувань.

1.8. Висновки до 1 розділу

У цьому розділі було показано, що сучасний студент одночасно працює з великою кількістю взаємопов’язаних навчальних завдань, які мають різну трудомісткість та дедлайни. За відсутності інструменту, який би допомагав саме у щоденному розподілі роботи, це призводить до типового ефекту «авралів»: значна частина завдань виконується в останні дні перед строком здачі, окремі дні виявляються суттєво перевантаженими, а частина робіт — прострочується.

Проблема полягає не стільки у загальному обсязі навантаження, скільки у його нерівномірному розподілі в часі.

Розгляд класичних задач теорії розкладів показав, що існує розвинений апарат для планування робіт із тривалостями та дедлайнами: жадібні правила на кшталт EDD/EDF, SPT/WSPT/LPT дозволяють будувати початкові розклади і в ряді випадків є оптимальними або близькими до оптимальних. Водночас ці підходи в базовому вигляді зазвичай не враховують обмеження на добову «пропускну здатність» виконавця, не працюють із дробленням великих завдань на менші блоки та не орієнтовані безпосередньо на вирівнювання навантаження між днями.

Аналіз автоматизації навчального розкладу на рівні закладів освіти показав, що задача побудови розкладу вже давно розглядається як NP-складна оптимізаційна проблема, для якої застосовують спеціалізовані алгоритми та метаевристики. Однак такі системи фокусуються на ресурсах університету (аудиторії, викладачі, групи) і глобальному розкладі занять. Індивідуальний рівень — щоденний розклад саме навчальної роботи студента між парами та в позааудиторний час — залишається поза їх прямою увагою.

Персоналізовані планери навчання та динамічні EdTech-планувальники демонструють наявний попит на інструменти, що формують індивідуальний навчальний графік і вміють адаптувати його до змін. Вони дозволяють будувати поденні плани, розкладати матеріал на блоки та частково реагувати на відставання. Водночас більшість таких рішень або орієнтовані на один курс/іспит, або не формулюють задачу явно як розклад з добовими ресурсними обмеженнями і не гарантують збалансованого розподілу навантаження між днями.

У підсумку було виявлено ключові обмеження наявних підходів і сформульовано базові вимоги до нового програмного засобу. Такий засіб має працювати з моделлю завдань, де для кожного вказано тривалість та дедлайн, враховувати доступний час студента по днях та заданий ним добовий ліміт,

підтримувати розбиття завдань на часові блоки й розподіляти їх по календарю з певним буфером до дедлайнів, а також забезпечувати можливість перепланування при зміні вхідних даних. Ці вимоги визначають постановку задачі, яка надалі формалізується та розв’язується за допомогою методу, що базується на модифікації правила EDD у поєднанні з локальною оптимізацією.

2. МЕТОДИ РОЗПОДІЛУ НАВЧАЛЬНОГО НАВАНТАЖЕННЯ

2.1. Формалізація вхідних даних, обмежень і припущень

Задачу розподілу навчального навантаження розглянемо як одноресурсну задачу планування (scheduling), де ресурсом виступає час конкретного студента, дискретизований по днях у межах фіксованого планувального горизонту (наприклад, модуль, сесія або семестр) [1; 5].

Вхідні дані:

- **Планувальний горизонт** у вигляді послідовності календарних днів $j = 1, 2, ..., H,$,
  де H — останній день періоду, для якого будується розклад.
- **Множину навчальних завдань**
  $T = \{t_1, t_2 ..., t_n\}$
  Для кожного завдання t_i задаються: **назва** (текстова ідентифікація завдання в інтерфейсі системи); **оцінка тривалості** p_i — кількість годин, необхідних для виконання завдання (трудомісткість); **дедлайн** $d_i \in \{1, ..., H\}$ — номер дня, до якого завдання бажано завершити.
- **Календар доступного часу студента**
  множина днів $\{day_1, ..., day_H\}$, для кожного з яких задано: **доступний час на навчання** h_j — максимальна кількість годин, яку студент реально може витратити на навчання в день day_j з урахуванням пар, роботи, дороги, відпочинку тощо.
- **Параметр буфера безпеки** $k \geq 0$ (у днях) — бажана відстань між фактичним завершенням завдання і його формальним дедлайном.
- **Технічні параметри дискретизації**: крок часу Δ (у годині; у реалізації використовується «годинний» крок); на основі p_i та Δ кожне завдання

може бути розбите на кілька однакових **часових блоків** (синонім «слотів»), що будуть рознесені по різних днях до дедлайну.

У поточній реалізації всі завдання вважаються **однаково важливими**: окремі вагові коефіцієнти важливості або явні пріоритети в модель не вводяться й у програмному засобі не використовуються

Обмеження:

Модель передбачає виконання двох основних груп обмежень.

– **Добовий ліміт годин (обмеження ресурсу)**

Для кожного дня day_j сумарна кількість годин, запланованих на всі завдання, не може перевищувати доступний час h_j:

$$\sum_{i=1}^{n} x_{ij} \le h_j, \quad j = 1, \ldots, H, \tag{2.1}$$

де x_{ji} — кількість годин (або блоків), відведених на завдання t_i у день day_j. Це класичне обмеження потужності ресурсу (capacity constraint).

– **Завершення завдань із буфером до дедлайну**

Для кожного завдання сумарний обсяг запланованих годин до «ефективного дедлайну» повинен бути не меншим за трудомісткість завдання:

$$\sum_{j=1}^{d_i-k} x_{ij} \ge p_i, \quad i = 1, \ldots, n, \tag{2.2}$$

де $d_i - k$ — ефективний дедлайн, що враховує буфер безпеки. Ідея буфера: знизити ризик зриву дедлайнів за рахунок завершення роботи трохи раніше формального терміну.

Якщо для деяких завдань через обмеженість h_j виконати цю умову не вдається, алгоритм фіксує залишок як потенційне запізнення і надалі враховує його у значеннях цільової функції.

Окремо на рівні реалізації також враховуються:

- **виключені дні** (повністю недоступні для навчання — відпустка, поїздка, робоча зміна тощо), які просто мають $h_j = 0$;
- **режим планування** (лише будні або будні + вихідні), що впливає на множину $\{day_j\}$.

Припущення моделі:

Для спрощення постановки задачі вводяться такі припущення:

- **Дискретний час**

  Час у моделі дискретизовано по днях, а всередині дня — на блоки фіксованої тривалості Δ (наприклад, 1 година). Це відповідає тому, як користувач задає доступні години на день у системі.
- **Адитивність трудомісткості**

  Загальна тривалість завдання p_i вважається сумою тривалостей окремих блоків; якісно немає різниці, у які саме години дня вони виконуються — важлива лише кількість годин у кожен день.
- **Незалежність завдань**

  Між завданнями немає жорстких прецедентних залежностей (типу «завдання В можна почати тільки після завершення А»). Усі задачі можуть виконуватися в будь-які дні до свого дедлайну, якщо це дозволяють h_j.
- **Фіксовані дедлайни та оцінки тривалості**

  Дедлайни d_i і оцінки тривалості p_i вважаються заданими коректно й не змінюються в межах одного запуску алгоритму. Їх корекція (наприклад, якщо студент недооцінив трудомісткість) розглядається як окремий сценарій адаптивного перепланування.

- **Однакова важливість завдань**

  У поточній версії моделі всі завдання мають однаковий пріоритет; критерії якості розкладу ґрунтуються на структурі навантаження (максимальне навантаження, кількість перевантажених днів, дисперсія, ризикові завдання), а не на індивідуальних вагових коефіцієнтах завдань.

2.2. Обґрунтування вибору підходу (модифікований EDD + локальна оптимізація)

Як було показано в розділі 1, задача побудови індивідуального розкладу навчального навантаження зі множиною завдань і обмеженим добовим ресурсом часу є варіантом складної задачі розкладання. Для таких задач у практиці зазвичай застосовують **дворівневу схему**: спочатку будують початковий (жадібний або евристичний) розклад, а потім покращують його за допомогою локальної оптимізації [5].

У цій роботі обрано саме такий підхід:

- на першому етапі використовується **модифікований алгоритм Earliest Due Date (EDD)** як базовий механізм розподілу завдань до їх дедлайнів;
- на другому етапі застосовується **локальна оптимізація типу hill climbing**, яка покращує початковий розклад, зменшуючи пікові навантаження та кількість перевантажених днів.

Вибір алгоритму EDD як бази:

Класичний EDD впорядковує завдання за зростанням дедлайнів: першими плануються задачі з найближчими строками виконання. Для навчального контексту таке правило є природним: завдання, дедлайн яких ближче, повинні потрапляти в графік раніше, ніж віддалені за часом.

Чистий EDD, однак, **ігнорує ресурсні обмеження студента**: алгоритм лише визначає порядок задач, але не контролює, скільки годин реально доступно в конкретний день. У результаті, якщо просто розкладати роботи в порядку

дедлайнів, перші дні можуть виявитися дуже щільними, а наступні — майже вільними, що є причиною пікових перевантажень.

Тому у роботі використовується **модифікований EDD**, який зберігає переваги базового алгоритму (простота, прив'язка до дедлайнів), але доповнюється механізмами:

- **урахування добового ліміту годин**: планувальник не розміщує на день більше годин, ніж доступно студенту;
- **дроблення великих завдань на часові блоки** і рознесення їх по різних днях до дедлайну;
- **буфер безпеки**: намагання завершити завдання на кілька днів раніше формального дедлайну для зниження ризику зриву.

У такій конфігурації EDD виступає **швидким конструктором первинного розкладу**, який уже враховує ключові обмеження саме задачі розподілу навчального навантаження, а не лише правильний порядок задач за дедлайнами.

Необхідність локальної оптимізації

Навіть із урахуванням добового ліміту, дроблення завдань і буфера, первинний розклад, побудований за модифікованим EDD, може залишатися **некомфортним**: окремі дні опиняються на межі ліміту, тоді як інші мають помітний резерв; блоки одного й того ж завдання можуть стояти надто щільно, а локальні «піки» навантаження — не відповідати реальним побажанням студента.

Щоб усунути ці недоліки, застосовується **локальна оптимізація розкладу** за принципом hill climbing. Ідея полягає в тому, що:

- відштовхуючись від вже побудованого розкладу, алгоритм виконує **малі локальні зміни** (перенесення окремих блоків завдань між днями, обміни блоками тощо);
- кожна зміна приймається, лише якщо вона **покращує певну цільову функцію**, яка відображає «якість» розкладу — зокрема, зменшує

максимальне добове навантаження, кількість перевантажених днів або дисперсію навантаження по днях;

– процес триває доти, поки вдається знайти локальні покращення.

Переваги такого підходу для задачі магістерської роботи:

– він добре поєднується з жадібним конструктором: спочатку швидко отримуємо реалістичний план, потім поступово його покращує, не перебудовуючи все з нуля;
– локальні зміни легко **пояснити з точки зору користувача** («перенесли частину цієї задачі на середу, щоб четвер не був перевантажений»);
– алгоритм природно працює в режимі регулярного перерахунку — при зміні стану (нові завдання, невиконані блоки, зміна доступних годин) можна повторно запустити побудову й оптимізацію, що важливо для сценарію адаптивного перепланування.

Недоліком локальної оптимізації є те, що отриманий результат **не гарантує глобального оптимуму**; це компроміс між якістю та обчислювальною складністю. Проте для практичного застосування в особистому планувальнику студента важливішою є можливість швидко отримати хороший, збалансований розклад, ніж витрачати значні ресурси на пошук теоретично найкращого.

Підсумкове обґрунтування

Отже, комбінація **модифікованого EDD** та **локальної оптимізації hill climbing** обрана з таких причин:

– відповідає типовій для складних задач розкладу двоетапній схемі «жадібна побудова + локальне покращення»;
– враховує специфічні для студентського сценарію обмеження: добовий ліміт часу, дроблення великих завдань, буфер до дедлайнів;

- дозволяє зменшити пікові навантаження та кількість перевантажених днів, що підтверджують результати експериментів, наведені в презентації (зниження кількості днів із максимальним навантаженням, дисперсії тощо);
- залишається обчислювально прийнятним і достатньо простим для реалізації в рамках веб-застосунку магістерського рівня.

Подальші підрозділи розділу 2 детально описують, як саме побудовано модифікований алгоритм EDD для даної задачі та як реалізовано локальну оптимізацію розкладу.

2.3. Базовий метод: модифікований алгоритм EDD

Базовим кроком запропонованого підходу є побудова початкового розкладу навчальних завдань на основі модифікованого правила **Earliest Due Date (EDD)** з урахуванням ресурсних обмежень студента. На цьому етапі система отримує на вхід:

- множину завдань з оціненою тривалістю виконання p_i та дедлайнами d_i;
- календар доступності, де для кожного дня d_j задано максимально допустиме навчальне навантаження h_j (у годинах);
- параметри методу: довжину елементарного блоку роботи Δ (год) та величину буфера безпеки k (днів).

Результатом роботи базового методу є поденний розклад, у якому кожне завдання розбите на окремі блоки й розподілене по календарних днях так, щоб:

- не перевищувати добовий ліміт h_j для жодного дня;
- завершувати завдання **раніше від формального дедлайну**;

Відмінності від класичного правила EDD

У класичній теорії розкладів правило EDD формулюється так: усі завдання впорядковуються за зростанням дедлайну d_i, і виконуються в цьому ж порядку. Такий підхід добре працює для мінімізації максимального запізнення в одноресурсних системах без явних обмежень на «місткість» кожного проміжку часу. Однак для задачі планування навчального навантаження студента така постановка є недостатньою:

- **ігноруються добові обмеження**: класичний EDD може фактично запланувати 10–12 годин роботи на один день, якщо так задовільно з точки зору дедлайнів;
- **ігнорується потреба розтягнути великі завдання**: курсова або проєкт можуть з’явитися у плані лише за декілька днів до дедлайну, хоча реалістично вимагають тижня поступової роботи.

Тому в розробленому підході правило EDD **модифікується** таким чином:

- вводиться **ефективний дедлайн** із буфером безпеки;
- жорстко враховується **добовий ліміт годин**;
- довгі завдання **автоматично дробляться на блоки** фіксованої тривалості;
- блоки завдань розміщуються в календарі, починаючи з днів, що передують дедлайну, зрухом назад та, за потреби, з винесенням частини роботи після дедлайну.

Ефективний дедлайн і дроблення завдань на блоки

Щоб знизити ризик роботи «в останню ніч», використовується поняття **ефективного дедлайну**:

$$d_i^{eff} = d_i - k, \tag{2.3}$$

де k — параметр буфера безпеки (у днях). Інтуїтивно це означає, що алгоритм планує завершити завдання на k днів раніше формального дедлайну d_i. Якщо для деяких завдань крайній термін потрапляє у минуле відносно початку

періоду планування, такий випадок фіксується як потенційний ризик прострочки й обробляється окремо на етапі аналізу якості розкладу.

Оскільки реальні навчальні завдання часто мають тривалість у кілька годин, а працювати 4–5 годин підряд над одним завданням не завжди доцільно, у методі використовується **дроблення завдань на блоки**:

- задається максимальна тривалість елементарного блоку Δ (наприклад, 1 година);
- для завдання з тривалістю p_i обчислюється кількість блоків $m_i = \left\lceil \frac{p_i}{\Delta} \right\rceil$
- створюється набір блоків кожен з яких має тривалість не більше Δ годин і наслідує ефективний дедлайн завдання.

Таким чином, далі алгоритм працює не з «суцільними» завданнями, а з набором дрібніших **робочих блоків**, які можна гнучко розкласти по календарю.

Покроковий опис алгоритму побудови розкладу

Базовий алгоритм побудови початкового розкладу можна описати наступним чином (Рисунок 2.1):

Крок 1. Підготовка даних

- для кожного завдання обчислюється ефективний дедлайн
- завдання, дедлайн яких уже минув, позначаються як **ризикові**; їхній коректний розподіл у межах буфера може бути неможливим, але алгоритм усе одно намагається виділити для них час.
- Кожне завдання розбивається на блоки тривалістю до Δ, як описано вище. На цьому етапі отримуємо множину блоків $\{b_k\}$ із відомою тривалістю та ефективним дедлайном.

Крок 2. Сортування блоків за ефективним дедлайном

- Усі блоки впорядковуються за зростанням ефективного дедлайну.

- За однакових значень ефективного дедлайну додаткове впорядкування може виконуватися за початковим дедлайном або за індексом завдання, але без введення окремих ваг чи пріоритетів — базовий метод оперує лише дедлайнами та тривалістю.

Таким чином, алгоритм спочатку розглядає блоки тих завдань, у яких «часу до дедлайну» найменше.

Крок 3. Розміщення блоків у календарі

- беремо черговий блок зі списку, який уже відсортовано за ефективним дедлайном.
- позначимо день, що безпосередньо передує ефективному дедлайну (або дорівнює йому, залежно від прийнятої нумерації).
- алгоритм намагається розмістити блок у доступні часові слоти, рухаючись назад у часі
- перевіряємо, чи не перевищить додавання блоку добовий ліміт
- якщо умова виконується, блок призначається на день обраний день, навантаження оновлюється, переходимо до наступного блоку
- якщо день перевантажений, переходимо до попереднього дня і повторюємо перевірку.
- якщо після проходу всіх доступних днів до початку періоду планування знайти місце для блоку не вдається, залишок роботи виноситься за ефективний дедлайн

Крок 4. Формування підсумкового розкладу

На виході базового алгоритму отримуємо:

- поденний розклад, у якому для кожного дня задано список блоків із прив’язкою до конкретних завдань;
- фактичне добове навантаження по кожному дню;
- список завдань (або їх частин), які не вдалося повністю розмістити до дедлайнів (потенційно прострочені або ризикові).

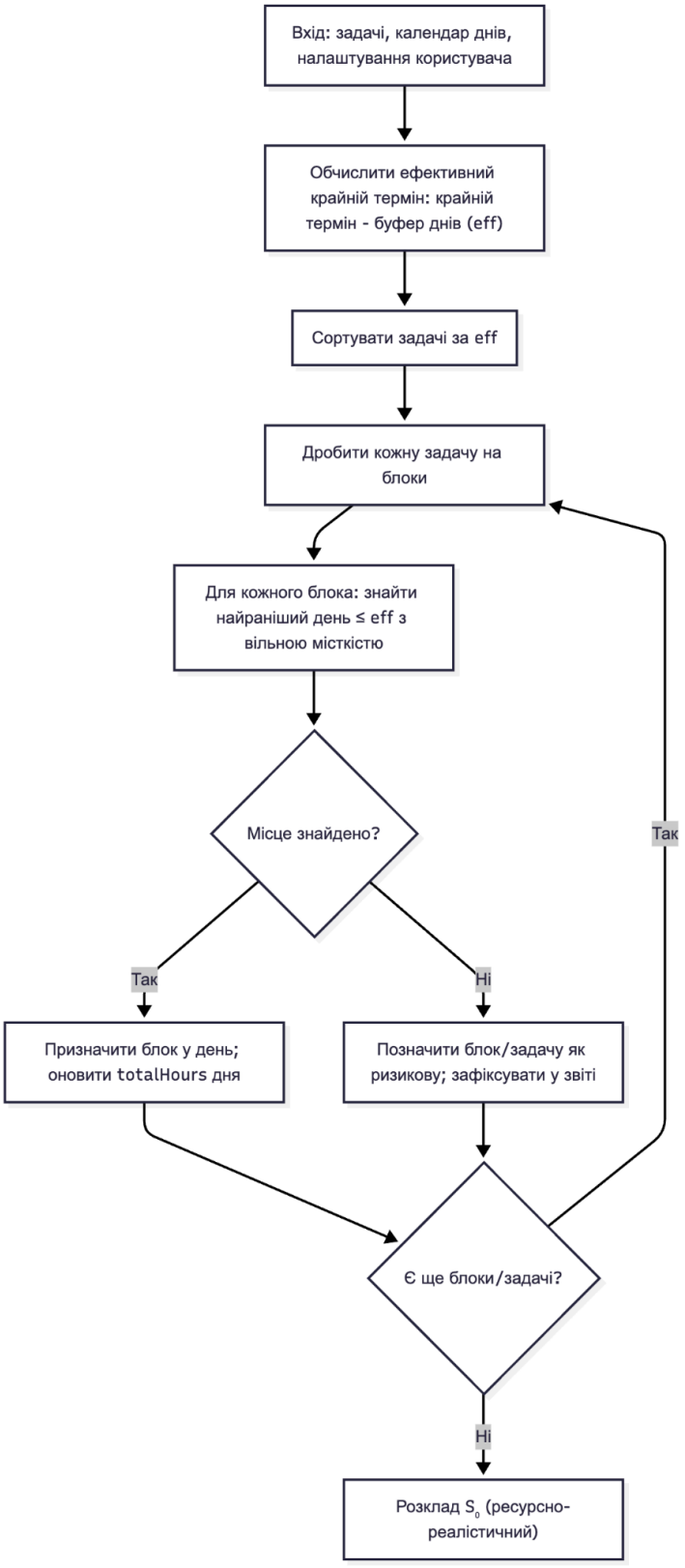


Рисунок 2.1 - модифікований алгоритм EDD

2.4. Локальна оптимізація розкладу (hill climbing)

Після застосування модифікованого алгоритму EDD отриманий розклад уже враховує дедлайни, добові ліміти та дроблення великих завдань. Проте навіть за цих умов окремі дні можуть залишатись надто щільними, тоді як сусідні — майже порожні. Тобто план є коректним з точки зору дедлайнів, але не обов'язково комфортним і рівномірним. Саме для «дошліфовування» такого базового розв'язку вводиться етап локальної оптимізації на основі підходу hill climbing [2].

Мета та загальна ідея локальної оптимізації

Локальна оптимізація розглядає розклад, побудований модифікованим EDD, як вихідну точку пошуку кращих рішень за тією ж функцією якості. Замість перебудови всього плану з нуля алгоритм виконує лише невеликі локальні зміни (перенесення або розбиття блоків роботи) і приймає тільки ті зміни, які покращують значення цільової функції [10; 11].

Підхід hill climbing можна інтерпретувати як стратегію «покращуй, поки можеш»:

- від поточного розкладу генерується множина «сусідніх» варіантів;
- для кожного сусіднього розкладу обчислюється значення функції якості;
- якщо знайдено варіант із кращим значенням, система переходить до цього варіанта;
- процес повторюється, доки не залишиться жодної локальної зміни, яка б давала покращення.

Таким чином, локальна оптимізація в запропонованому рішенні:

- згладжує пікові дні,
- зменшує кількість перевантажених днів,
- не порушує обмежень на добовий ліміт та дедлайни завдань.

Функція оцінки якості розкладу

Нехай S — фіксований розклад, тобто матриця значень $p_{i,j}$, де $p_{i,j}$ — кількість годин, відведених на завдання t_i у день d_j.

Для оцінки якості розкладу використовуються такі компоненти:

- **Запізнення завдань.**

  Для кожного завдання t_i визначається день фактичного завершення $C_i(S)$ як останній день, у який для цього завдання відводяться години. Тоді запізнення:

  $$L_i(S) = max\{0,\ C_i(S) - d_i\}, \tag{2.4}$$

  де d_i — дедлайн завдання. Якщо роботу завершено вчасно або з буфером, $L_i(S) = 0$.

- **Нерівномірність навантаження по днях.**

  Для кожного дня d_j обчислюється сумарне навантаження:

  $$x_j(S) = \sum_i p_{i,j}. \tag{2.5}$$

  На основі вектора $\{x_j\}$ обчислюється дисперсія добового навантаження $V(S)$, яка відображає, наскільки «стрибає» кількість годин по днях (чим більша дисперсія — тим нерівномірніший графік)

- **Перевантажені дні та пікові значення.**

  Додатково виділяються похідні показники, що використовуються у функції якості та для аналізу в експериментальному розділі: кількість днів, у яких навантаження перевищує заданий поріг (наприклад, > 4 год/день); максимальне значення x_j серед усіх днів (пікове навантаження); кількість «повністю заповнених» днів, коли використано всі доступні години; кількість завдань, які опиняються в зоні ризику прострочки.

На практиці для локальної оптимізації використовується скалярна функція якості $F(S)$, яка поєднує ці компоненти (наприклад, через зважену суму):

- штраф за запізнення завдань ($\sum_i L_i(S)$);
- штраф за нерівномірність ($V(S)$);
- штраф за перевантажені дні та інші небажані ефекти.

Коефіцієнти ваг α, β, ... задають баланс між «жорстким» дотриманням дедлайнів та комфортністю/рівномірністю навчального процесу: зменшення запізнень вважається пріоритетним, а згладжування навантаження — бажаним, але другорядним критерієм.

При пошуку сусідніх розкладів алгоритм розглядає лише ті зміни, які **не створюють нових прострочень** і не порушують добових лімітів. Це відповідає загальній меті — згладити піки, не жертвуючи дедлайнами [10].

Операції локальних змін (сусідні стани)

Щоб визначити «сусідні» розклади для hill climbing, вводяться кілька типів елементарних операцій над матрицею $p_{i,j}$.

- **Переміщення частини завдання в інший день**

  Обирається завдання t_i та день d_j, де воно займає $p_{i,j} > 0$ годин. Частина цього блоку (наприклад, Δp) переноситься в інший день d_k, де є вільний часовий ресурс. У результаті:

  - у дні d_j зменшується навантаження, що потенційно знижує пік та дисперсію;
  - у дні d_k навантаження зростає, але не перевищує добовий ліміт;
  - задача залишається завершеною не пізніше за дедлайн (з урахуванням буфера).

- **Обмін часовими блоками між днями**

  Обираються два блоки роботи (можливо, різних завдань), розташовані у різних днях. Алгоритм пробує поміняти їх місцями, якщо це:

  - зменшує сумарне запізнення по завданнях;
  - знижує пікове навантаження або дисперсію по днях;
  - не призводить до порушення лімітів або дедлайнів.

- **Додаткове дроблення великого блоку**

  Якщо деяке завдання t_i займає у певний день великий безперервний блок (наприклад, 3–4 години), алгоритм може розділити його на кілька менших блоків та рознести частину із них на попередні дні з вільним ресурсом. Такі операції:

  - зменшують навантаження у «піковий» день;
  - знижують дисперсію $V(S)$ і кількість перевантажених днів;
  - зберігають сумарну тривалість робіт над завданням.

Кожна з цих операцій створює новий кандидатний розклад S'. Прийнятними вважаються лише ті кандидати, що не порушують жодного з обмежень, сформульованих у п. 2.1 (дедлайни, добові ліміти, цілісність сумарної тривалості завдань).

Алгоритм hill climbing

Процедура локальної оптимізації розкладу на основі hill climbing у роботі реалізована у такому загальному вигляді:

- **Ініціалізація**
  - Вхід: базовий розклад S_0, побудований модифікованим EDD (п. 2.3).
  - Обчислюється початкове значення функції якості $F(S_0)$.
  - Встановлюються параметри: максимальна кількість ітерацій, максимальна кількість невдалих спроб покращення тощо.
- **Генерація сусідніх рішень**

- На кожній ітерації з поточного розкладу S формується обмежена множина кандидатів $\{S'_1, S'_2, \ldots\}$ за допомогою операцій з п. 2.4.3.
- Генерація зосереджується насамперед на «проблемних» днях: днях із надмірним навантаженням, днях із максимальними значеннями x_j, завданнях, які виконуються занадто пізно тощо.

– **Оцінка кандидатів**

Для кожного допустимого кандидата S':

- перевіряються обмеження (дедлайни, добові ліміти, сумарний час по завданнях);
- обчислюється значення $F(S')$;
- запам'ятовується кандидат із найменшим значенням F серед сформованих сусідів.

– **Крок покращення**

- Якщо знайдено сусіда S^* з $F(S^*) < F(S)$, алгоритм переходить у новий стан: $S \leftarrow S^*,\ F(S) \leftarrow F(S^*)$;
- якщо жоден із кандидатів не дає покращення, алгоритм зупиняється — досягнуто локального мінімуму за обраною функцією якості.

– **Критерії зупинки**

Окрім відсутності покращень, можуть застосовуватись додаткові обмеження:

- досягнення максимальної кількості ітерацій;
- занадто мала зміна функції якості (ефект насичення);
- обмеження часу виконання алгоритму.

У підсумку отримується оптимізований розклад S_{opt}, у якому:

- збережено всі обмеження моделі (дедлайни, добові ліміти, завершення завдань у межах буфера);
- знижено пікові значення добового навантаження;

– зменшено кількість перевантажених днів;
– підвищено рівномірність розподілу навчальної роботи у часі.

Класичний недолік hill climbing полягає в ризику «застрягти» у локальному мінімумі, коли жодна з малих локальних змін не дає покращення, хоча глобально існує кращий розклад. У роботі це враховується як потенційний напрямок розвитку методу (розширення до стохастичних підходів типу simulated annealing), але в реалізованому прототипі використовується саме детермінований hill climbing з обмеженням на кількість ітерацій, що забезпечує простоту, пояснюваність і достатню швидкодію для реальних навчальних сценаріїв.

2.5. Адаптивне перепланування після зриву

У реальному навчальному процесі навіть найкращий початковий план рідко виконується повністю: студент може не встигнути частину запланованих блоків роботи, змінюється графік занять або з’являються нові завдання. Якщо система не підтримує адаптивне перепланування, план фактично «помирає» після першого ж зриву, і користувач повертається до ручного планування.

У розробленому програмному засобі передбачено окремий експлуатаційний сценарій роботи після зриву плану. Його мета – коректно перебудувати розклад, зберігаючи дедлайни та добові обмеження.

На рівні алгоритму перепланування використовується та сама зв’язка, що й при первинному плануванні: модифікований EDD як базовий генератор розкладу та локальна оптимізація hill climbing. Відмінність полягає тільки в тому, які дані подаються на вхід.

Основні кроки сценарію перепланування такі:

– **Фіксація нового «поточного дня»**

  При переході до нового календарного дня система розглядає всі заплановані на попередні дні блоки роботи. Ті блоки, які були виконані,

потрібно видалити з набору. Блоки, які студент не встиг виконати, потрапляють до набору.

- **Повернення невиконаних блоків у пул завдань.**
  Невиконані блоки приводяться до формату «решткі задач»: для кожного завдання оновлюється залишкова тривалість (скільки годин ще потрібно), а дедлайн залишається тим самим. Таким чином, формується новий набір завдань із оновленими трудомісткостями.
- **Оновлення календаря доступності.**
  Дні, які вже минули, не потрапляють до нового графіку.
- **Повторний запуск генератора розкладу.**
  Оновлений набір завдань (із залишковими тривалостями) та календар доступності для майбутніх днів знову подається на вхід базового методу (модифікований EDD). Алгоритм розкладає залишкові блоки по доступних днях із урахуванням добових лімітів та буфера до дедлайнів.
- **Локальна оптимізація оновленого плану.**
  До отриманого розкладу знову застосовується hill climbing для згладжування локальних піків навантаження та зменшення кількості перевантажених днів у новій конфігурації. У результаті студент отримує оновлений план.

Таким чином, алгоритмічні модулі планування і оптимізації використовуються не одноразово, а в циклі «план → виконання → зрив → перепланування». Це дозволяє за потреби підтримувати адаптивний графік навчальної роботи, який відповідає реальній динаміці життя студента, а не лише початковим ідеальним припущенням.

2.6. Обчислювальна складність і очікувані властивості методу

Запропонований метод складається з двох основних етапів:

– побудова початкового розкладу за допомогою модифікованого алгоритму EDD;
– локальна оптимізація розкладу за схемою hill climbing.

Проаналізуємо їхню обчислювальну складність та властивості результату.

Обчислювальна складність базового алгоритму

Нехай:

– n – кількість завдань;
– H – кількість днів у планувальному горизонті;
– M – загальна кількість часових блоків після дроблення завдань (сума блоків для всіх задач).

Тоді:

– **Підготовка даних.**
  Обчислення ефективних дедлайнів та розбиття завдань на блоки виконується за час
  $$O(n + M) \tag{2.6}$$
  оскільки для кожного завдання один раз визначається d_i^{eff} та створюється відповідна кількість блоків.
– **Сортування блоків за ефективним дедлайном.**
  Усі M блоків сортуються за d_i^{eff}, що потребує
  $$O(MlogM) \tag{2.7}$$
– **Розміщення блоків у календарі.**
  Для кожного блока алгоритм у найгіршому випадку може послідовно переглянути до H днів (рухаючись назад від ефективного дедлайну, а за потреби – вперед). Таким чином, верхня оцінка складності цього кроку:
  $$O(M \cdot H) \tag{2.8}$$

Отже, загальна складність етапу побудови початкового розкладу:

$$O(MlogM + M \cdot H) \quad (2.9)$$

де домінуючим доданком зазвичай є $M \cdot H$. У типових сценаріях використання (де n – десятки завдань, а H – від кількох тижнів до 1–2 місяців) така складність є прийнятною для інтерактивної роботи веб-застосунку.

Обчислювальна складність локальної оптимізації

Нехай:

- I – кількість ітерацій алгоритму hill climbing;
- на кожній ітерації розглядається не більше K сусідніх розкладів (кандидатів);
- оцінка одного кандидата потребує перерахунку добових навантажень та метрик якості, що займає час порядку $O(n + H)$.

Тоді верхня оцінка складності етапу локальної оптимізації:

$$O(I \cdot K \cdot (n + H)) \quad (2.10)$$

У реалізованому прототипі значення I та K обмежуються (наприклад, десятками ітерацій та фіксованою кількістю кандидатів на кожному кроці), що дозволяє зберегти прийнятний час роботи навіть при повторних переплануваннях [3].

Очікувані властивості побудованого розкладу

Поєднання модифікованого EDD та локальної оптимізації забезпечує такі властивості отриманого розкладу:

- **Дотримання ресурсних обмежень по днях**

  Для кожного дня навантаження не перевищує доступного часу h_j, заданого користувачем. Якщо обсяг завдань надто великий, це проявляється не як «перевантажені» дні, а як неможливість повністю розмістити всі задачі до їх дедлайнів.

– **Переважання завдань з ближчими дедлайнами**

  Завдяки EDD-подібному впорядкуванню блоків, завдання з найближчими дедлайнами мають вищий шанс бути спланованими раніше, що зменшує ризик накопичення запізнень.

– **Розтягування великих завдань у часі**

  Дроблення завдань на блоки забезпечує більш плавне включення великих робіт у календар: замість одного «великого стрибка» навантаження в день дедлайну завдання розподіляється на кілька відносно коротких сесій.

– **Зменшення пікових навантажень та нерівномірності після локальної оптимізації**

  Hill climbing намагається згладити піки за рахунок перенесення блоків у дні з вільним ресурсом. У результаті очікується зниження максимального добового навантаження та дисперсії навантаження між днями порівняно з чистим результатом модифікованого EDD.

– **Відсутність гарантії глобального оптимуму**

  Використаний підхід є евристичним: результат локальної оптимізації є локальним мінімумом вибраної функції якості, але не обов’язково глобально оптимальним розкладом. Це свідомий компроміс між якістю рішення та обчислювальною складністю, прийнятний для персонального планувальника навчальної роботи [6].

Таким чином, запропонований метод має поліноміальну обчислювальну складність і демонструє передбачувану поведінку при збільшенні кількості завдань та довжини планувального горизонту, що дозволяє застосовувати його в реальних умовах використання студентами.

2.7. Висновки до 2 розділу

У цьому розділі задача автоматизованого розподілу навчального навантаження студента була формалізована як одноресурсна задача розкладання з обмеженим добовим ресурсом часу. Введено поняття планувального горизонту, множини завдань із тривалістю та дедлайнами, календаря доступних годин по днях, буфера до дедлайну та дискретизації часу на блоки фіксованої тривалості. Сформульовано основні обмеження: неперевищення добового ліміту годин, завершення завдань до ефективних дедлайнів та збереження сумарної тривалості роботи над кожним завданням.

Як базовий підхід до побудови розкладу обрано модифікований алгоритм Earliest Due Date. Він зберігає інтуїтивну ідею планування завдань у порядку наближення дедлайнів, але доповнюється врахуванням добових обмежень, розбиттям довгих завдань на часові блоки та використанням буфера безпеки відносно формальної дати здачі. На основі цього алгоритму формується початковий поденний розклад, який уже є коректним щодо ресурсних обмежень, але може залишатися нерівномірним за навантаженням.

Для покращення отриманого розкладу застосовано локальну оптимізацію за схемою hill climbing. Вона виконує локальні переміщення та обміни блоків роботи між днями, приймаючи лише ті зміни, які зменшують обрані показники «якості» розкладу (передусім пікове навантаження та нерівномірність розподілу годин між днями) без порушення дедлайнів і добових лімітів. Перепланування в реалізованій системі здійснюється шляхом повторної побудови розкладу на оновленому списку завдань після видалення виконаних робіт, що дозволяє отримувати актуальний графік у ході навчального процесу.

Проведена оцінка обчислювальної складності показала, що обидва етапи методу мають поліноміальну складність і можуть виконуватися в прийнятний час для типових обсягів завдань і тривалості планувального горизонту. Запропонований підхід не гарантує глобально оптимального розкладу, однак забезпечує конструктивний, ресурсно-коректний та покращений за основними

метриками варіант розподілу навчального навантаження, придатний для використання у якості основи програмного засобу персонального планування.

3. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Архітектура програмного рішення

Призначення системи та загальна структура

Розроблене програмне забезпечення реалізує прототип персонального планувальника навчального навантаження студента. Основна ідея полягає в автоматизації того, що зазвичай виконується вручну та безсистемно: розподілу підготовки до лабораторних робіт, модульних контрольних, курсових та інших завдань у межах обмеженого особистого часу студента.

Система працює із двома основними групами вхідних даних:

- список навчальних завдань із зазначенням крайніми термінами та орієнтовної тривалості у годинах;
- календар особистої доступності студента, де для кожного дня задається максимальна кількість годин, яку він може витратити на навчання.

На основі цих даних застосунок формує індивідуальний розклад навчальної роботи — для кожного дня визначає, скільки годин слід присвятити конкретним завданням, з урахуванням дедлайнів, денного ліміту та буферу безпеки. Далі розклад може бути покращено за допомогою локальної оптимізації, щоб зменшити пікові навантаження та зробити розподіл годин більш рівномірним. У випадку зміни вхідних даних (додавання/видалення завдань, корекція календаря) розклад перегенеровується.

Загальна архітектура програмного рішення представлена у вигляді односторінкового веб-застосунку (SPA), який повністю виконується на боці клієнта та не потребує окремого серверного компонента. На структурній схемі (рис. 3.1) виділяються такі блоки: шар користувацького інтерфейсу (UI), шар управління станом, алгоритмічне ядро планування та оптимізації, а також

підсистема локального зберігання даних у браузері.

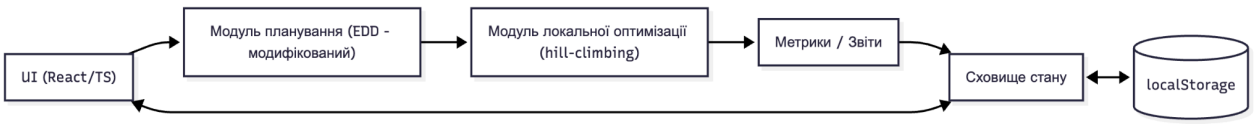


Рисунок 3.1 - Загальна архітектура процесів

Рівні та основні модулі системи

Архітектура рішення будується за принципом чіткого розділення відповідальностей між шарами:

- **Презентаційний шар (UI)**

  Реалізований як набір React-компонентів для основних екранів застосунку: «Мої завдання» — введення та редагування переліку завдань; «Налаштування» — конфігурація періоду планування та буферу безпеки; «Календар» — налаштування доступних годин по днях; «Розклад» — запуск генерації та перегляд отриманого розкладу з ключовими метриками; «Оптимізація» — запуск локального покращення розкладу.
- Деякі додаткові розділи (наприклад, окремий екран «Корекція / перепланування») закладені в архітектурі, однак у поточній версії застосунку використовуються обмежено й можуть розглядатися як резерв для подальшого розвитку.
- **Шар управління станом**

  Глобальний стан додатку зберігається у централізованому сховищі (Zustand-store). Тут підтримуються: дані предметної області (завдання, календар, налаштування, поточний розклад, метрики); службові прапорці (процес триває/завершився, помилки тощо), які використовуються для блокування кнопок та відображення повідомлень.
- Store інкапсулює всі дії з даними: додавання/редагування завдань, ініціалізація календаря, генерація розкладу, запуск оптимізації, скидання помилок та ін.

- **Алгоритмічне ядро (модулі EDD-core та оптимізації)**
  Окремі модулі інкапсулюють реалізацію модифікованого алгоритму EDD для побудови початкового розкладу та алгоритму локальної оптимізації Hill Climbing. Модуль планування отримує копії списку завдань та календаря і повертає набір блоків розкладу, які дотримуються обмежень денного ліміту та дедлайнів. Модуль оптимізації працює поверх уже згенерованого розкладу й намагається зменшити кількість перевантажених днів та дисперсію навантаження шляхом локальних переміщень блоків між днями.
- Для підтримки цих модулів виділено утилітний підмодуль (розрахунок метрик, групування блоків за днями, виявлення завдань із ризиком прострочення, генерація ідентифікаторів тощо).
- **Сервісний шар (API-абстракція)**
  Між UI/станом та алгоритмами розташований тонкий сервісний шар, який надає уніфіковані методи для запуску EDD-планування та оптимізації з імітацією мережевої затримки. З точки зору архітектури це дозволяє у майбутньому замінити локальні виклики на реальні HTTP-запити до backend-сервісу без суттєвої переробки клієнтського коду.
- **Шар зберігання даних (localStorage)**
  Дані користувача (список завдань, календар доступності, налаштування, кількість днів планування) зберігаються в локальному сховищі браузера через middleware персистентності у Zustand.

Такий поділ на шари забезпечує слабке зв’язування між інтерфейсом, станом та алгоритмами, полегшує тестування та подальший розвиток системи.

Управління даними та потоками взаємодії

Типи даних системи (завдання, день календаря, налаштування, блок розкладу, метрики розкладу) формально описані як TypeScript-структури. Це дозволяє

чітко задати інтерфейси між модулями й уникати неузгодженостей при зміні логіки.

Ключові сценарії роботи реалізовані як послідовності викликів дій сховища та алгоритмічних сервісів:

- **Формування початкового розкладу**: користувач задає період планування та буфер безпеки; налаштовує календар доступності; додає завдання; у розділі «Розклад» запускає генерацію, після чого store звертається до EDD-модуля й записує отримані блоки та метрики.
- **Оптимізація вже згенерованого розкладу:** на екрані «Розклад» користувач бачить початкові метрики; переходить до розділу «Оптимізація» та запускає покращення; сервіс оптимізації викликає алгоритм Hill Climbing, порівнює метрики до/після та оновлює стан розкладу в store.
- **Оновлення після змін у вхідних даних**

  Додавання нового завдання або зміна календаря приводять до повторного запуску процесу планування: користувач повертається до розділу «Розклад» і ініціює перегенерацію. Таким чином, поточний підхід до “перепланування” реалізовано як видалення застарілого розкладу та побудова нового на основі актуальних даних, а не як окремий, повністю відпрацьований сценарій розподілу пропущених блоків. Це відповідає фактичному стану реалізації та залишає можливість доопрацювати повноцінний модуль корекції в межах подальших досліджень.

Технологічний стек та обґрунтування вибору

Для реалізації прототипу обрано легкий веб-стек, орієнтований на швидку розробку клієнтського SPA-застосунку без окремого сервера:

- **TypeScript** — для статичної типізації основних сутностей (Task, CalendarDay, ScheduleBlock, ScheduleMetrics тощо) і зменшення помилок у складній логіці планування.

- **React** — як бібліотека побудови інтерфейсу з декларативним оновленням екранів при зміні стану [4].
- **Zustand** — компактне сховище глобального стану без надлишкового шаблонного коду, що добре підходить для прототипу.
- **Ant Design** — набір готових UI-компонентів (таблиці, форми, модальні діалоги), який дозволив сфокусуватися на логіці планування, а не на верстці.
- **Tailwind CSS** — утилітарний підхід до стилізації, зручний для побудови компактних індикаторів навантаження та станів без великого обсягу CSS.
- **Day.js** — для роботи з датами та розрахунку діапазону планування.
- **localStorage** — як механізм локального зберігання даних користувача без розгортання бази даних чи backend-компонента.

Вибір такого стеку узгоджується з вимогами до прототипу:

- можливість отримати працездатний продукт який легко запустити та підтримувати
- простота використання, достатньо звичайного браузера
- можливість у майбутньому винести алгоритмічне ядро на сервер без повної переробки клієнтської частини.

3.2. Модель даних та формати зберігання

Основні сутності предметної області

Модель даних побудована так, щоб напряму відповідати формалізації з розділу 2 (завдання, дні планувального горизонту, блоки розкладу, метрики якості) і водночас бути зручною для реалізації у веб-застосунку на TypeScript. Усі основні сутності описані у вигляді інтерфейсів, які задають структуру даних на рівні програми.

Завдання (Task)

Сутність *завдання* відповідає одному елементу множини задач T з розділу 2.1. Для кожного завдання зберігаються ідентифікатор, назва, (опційний) опис, оцінка тривалості в годинах, дедлайн у вигляді номера дня планувального горизонту та резервне поле пріоритету. Структуру сутності Task наведено в табл. 3.1.

Таблиця 3.1 – Атрибути сутності Task

Атрибут	Тип	Обов’язковий	Опис
id	string	так	Унікальний ідентифікатор завдання, використовується для зв’язку з блоками розкладу
title	string	так	Коротка назва завдання (відображається у списку завдань та в розкладі)
description	string | null	ні	Розширений текстовий опис завдання (за потреби)
totalHours	number	так	Орієнтовна тривалість виконання завдання в годинах
deadlineDay	number	так	Номер дня в межах планувального горизонту, до якого завдання бажано завершити
priority	"high" | "medium" | "low"	так	Резервне поле для підтримки пріоритетів; у поточній версії всі завдання мають значення medium, алгоритм планування це поле не враховує

У поточній реалізації всі завдання створюються з фіксованим значенням priority = "medium", тобто пріоритети **не впливають** на роботу алгоритмів і залишені як заділ для майбутнього розширення.

День календаря (CalendarDay)

Сутність *день календаря* описує один елемент планувального горизонту з точки зору доступності часу студента. Структуру сутності CalendarDay наведено в табл. 3.2.

Таблиця 3.2 – Атрибути сутності CalendarDay

Атрибут	Тип	Обов'язковий	Опис
dayIndex	number	так	Порядковий номер дня (1…N) у межах планувального горизонту
date	string	так	Календарна дата у форматі ISO (YYYY-MM-DD) для відображення в інтерфейсі
capacityHours	number	так	Кількість годин, яку студент може витратити на навчання в цей день (добовий ліміт)
isExcluded	boolean	так	Ознака повного виключення дня з планування (у цьому випадку capacityHours = 0)

Блок розкладу (ScheduleBlock)

Завдання можуть бути розбиті на кілька часових блоків, які розносяться по різних днях. Для представлення результатів планування використовується сутність *блок розкладу*. Її структура наведена в табл. 3.3.

Таблиця 3.3 – Атрибути сутності ScheduleBlock

Атрибут	Тип	Обов’язковий	Опис
id	string	так	Унікальний ідентифікатор блоку розкладу
dayIndex	number	так	Номер дня, до якого прив’язаний блок
taskId	string	так	Посилання на завдання (Task.id), якому належить блок
hours	number	так	Кількість годин роботи над завданням у цей день
isAfterDeadline	boolean	ні	Службова ознака: чи потрапляє блок після формального дедлайну завдання (для аналізу ризиків)

У термінах формальної моделі матриця $p_{i,j}$ реалізується як набір таких блоків, згрупованих за днями й завданнями.

Метрики розкладу (ScheduleMetrics)

Для оцінювання якості побудованого розкладу використовується набір агрегованих показників – сутність *метрик розкладу* (табл. 3.4).

Таблиця 3.4 – Атрибути сутності ScheduleMetrics

Атрибут	Тип	Обов’язковий	Опис
totalHours	number	так	Сумарна кількість годин, запланована на весь період
averageLoad	number	так	Середнє добове навантаження (годин на день)
peakLoad	number	так	Максимальне навантаження за один день
overloadedDays	number	так	Кількість днів, у яких навантаження перевищує заданий поріг (наприклад, > 4 год/день)
peakLoadDays	number	так	Кількість днів із піковим навантаженням (максимумом)
tasksAtRisk	number	так	Кількість завдань, для яких частина роботи потрапила за дедлайн або не вмістилася в розклад
loadVariance	number	так	Дисперсія добового навантаження, що характеризує нерівномірність розподілу годин

Ці метрики використовуються як для відображення користувачу, так і в алгоритмі локальної оптимізації для порівняння різних варіантів розкладу.

Налаштування користувача (UserSettings)

Параметри, що впливають на поведінку програми (кінець семестру, використання лише буднів, буфер до дедлайнів тощо), зберігаються в окремій сутності *налаштувань*. Структуру сутності UserSettings наведено в табл. 3.5.

Таблиця 3.5 – Атрибути сутності UserSettings

Атрибут	Тип	Обов’язковий	Опис
semesterEndDate	string	так	Кінцева дата планувального горизонту у форматі ISO (YYYY-MM-DD)
weekdaysOnly	boolean	так	Ознака використання лише робочих днів (понеділок–п’ятниця) при побудові календаря
excludedDates	string[]	так	Масив конкретних дат, які виключаються з планування незалежно від weekdaysOnly
bufferDays	number	так	Величина буфера k у днях, що віднімається від дедлайнів при обчисленні «ефективного дедлайну»

Глобальний стан застосунку (AppState)

Глобальний стан (store) об’єднує всі описані вище сутності та службову інформацію. Основні поля стану наведено в табл. 3.6.

Таблиця 3.6 – Основні поля глобального стану AppState

Атрибут	Тип	Обов’язковий	Опис
tasks	Task[]	так	Поточний список навчальних завдань
calendar	CalendarDay[]	так	Календар доступності студента по днях
schedule	ScheduleBlock[]	так	Останній згенерований розклад у вигляді набору блоків
metrics	ScheduleMetrics | null	ні	Метрики для поточного розкладу (якщо розклад ще не будувався – null)
numberOfDays	number	так	Довжина планувального горизонту в днях
userSettings	UserSettings	так	Налаштування користувача, що впливають на планування
isGeneratingSchedule	boolean	так	Ознака того, що зараз відбувається генерація розкладу (для блокування кнопок у UI)
isOptimizing	boolean	так	Ознака того, що виконується етап оптимізації розкладу
errorMessage	string | null	ні	Текст останньої помилки (якщо сталася)

Таким чином, модель даних безпосередньо відображає інформаційні об’єкти, введені у формалізації методу, і є базою для реалізації алгоритмів планування та оптимізації.

Зв’язки між сутностями

Між основними сутностями існують такі логічні зв’язки:

- **Task → ScheduleBlock**: одне завдання (Task) може мати нуль, один або кілька блоків розкладу (ScheduleBlock), пов’язаних через поле taskId.
- **CalendarDay → ScheduleBlock**: кожен день календаря (CalendarDay) може містити нуль, один або багато блоків розкладу; зв’язок реалізується через поле dayIndex.
- **ScheduleBlock → ScheduleMetrics**: метрики розкладу (ScheduleMetrics) обчислюються на основі всієї множини блоків (ScheduleBlock[]), зокрема сумарного навантаження по днях та наявності блоків після дедлайнів.
- **UserSettings → CalendarDay / алгоритми**: налаштування (UserSettings) впливають на формування масиву CalendarDay[] (кінець періоду, тільки будні, виключені дати) та на обчислення ефективних дедлайнів при плануванні (bufferDays).

Таке структурування забезпечує гнучкість: можна змінити календар або список завдань, не змінюючи формат блоків чи метрик, а просто переобчисливши розклад.

Формати зберігання даних

Оскільки застосунок працює повністю на боці клієнта, для зберігання даних використовується localStorage браузера. На рівні реалізації застосовується middleware персистентності бібліотеки Zustand, яке серіалізує частину глобального стану (tasks, calendar, numberOfDays, userSettings) у вигляді JSON-об’єкта під фіксованим ключем.

Схематично JSON-структуру збережених даних можна подати як:

```
{
 "tasks": [ … ],
 "calendar": [ … ],
 "numberOfDays": N,
 "userSettings": {
  "semesterEndDate": "YYYY-MM-DD",
  "weekdaysOnly": true/false,
  "excludedDates": [ "YYYY-MM-DD", … ],
  "bufferDays": k
 }
}
```

Усі дати зберігаються як рядки у форматі ISO, що спрощує їх подальшу обробку бібліотекою для роботи з датами.

Узгодженість моделі з математичною постановкою

Запропонована модель даних безпосередньо відображає математичні об’єкти з розділу 2:

- Task.totalHours ↔ тривалість завдання p_i;
- Task.deadlineDay ↔ дискретизований дедлайн d_i;
- CalendarDay.capacityHours ↔ добовий ресурс h_j;
- масив ScheduleBlock[] ↔ матриця розподілу $p_{i,j}$;
- UserSettings.bufferDays ↔ параметр буфера k;
- ScheduleMetrics ↔ набір метрик, що використовуються для аналізу та локальної оптимізації.

Завдяки цьому перехід від теоретичного опису до програмної реалізації є прозорим: усі величини, що фігурують у формальних обмеженнях і функціях якості, мають явні носії в моделі даних застосунку.

3.3. Реалізація модуля планування (EDD-core)

Модуль планування (умовно eddScheduler.ts) реалізує базовий етап методу, описаного в розділі 2: побудову початкового розкладу на основі модифікованого правила Earliest Due Date з урахуванням добових обмежень і буфера до дедлайнів.

На рівні коду це оформлено як функція на кшталт:

```
generateInitialSchedule(
  tasks: Task[],
  calendar: CalendarDay[],
  bufferDays: number
): { schedule: ScheduleBlock[]; metrics: ScheduleMetrics }
```

де:

- tasks – список завдань (структура наведена в табл. 3.1);
- calendar – масив днів планувального горизонту з місткістю по годинах (табл. 3.2);
- bufferDays – значення буфера (k), яке віднімається від дедлайнів для отримання «ефективних» дедлайнів;
- schedule – згенерований набір блоків розкладу (табл. 3.3);
- metrics – обчислені метрики якості цього розкладу (табл. 3.4).

Функція не змінює вхідні дані, а повертає новий масив блоків; це спрощує тестування й дозволяє незалежно порівнювати різні розклади.

Підготовка та впорядкування завдань

На першому етапі модуль виконує нормалізацію вхідних даних:

- **Фільтрація та валідація**

  - Із списку завдань відкидаються порожні або некоректні записи (наприклад, з нульовою або від’ємною тривалістю).
  - Дедлайни deadlineDay обмежуються діапазоном від 1 до довжини календаря numberOfDays (якщо користувач помилився з датою).
- **Обчислення ефективних дедлайнів**

  Для кожного завдання обчислюється як різниця між фактичним крайнім терміном та буфером безпеки.

  Якщо bufferDays занадто великий і «заводить» дедлайн у минуле, ефективний дедлайн обрізається до першого дня планувального горизонту; такі завдання вважаються потенційно ризиковими, якщо їх обсяг великий.
- **Впорядкування завдань**

  Відповідно до ідеї EDD задачі впорядковуються за зростанням дедлайну.

  У реалізації це робиться за кількома полями:
  - спочатку за deadlineDay (чим раніше дедлайн, тим раніше завдання в черзі),
  - далі – за ефективним дедлайном,
  - далі – за тривалістю totalHours (довші завдання потрапляють вище, щоб їх розкидати раніше).
- Поле priority присутнє в структурі Task, але в поточному варіанті всі завдання мають значення "medium", тож фактично пріоритет **не впливає** на впорядкування.

Результатом цього кроку є впорядкований список завдань, який подається на розподіл по днях.

Алгоритм розподілу годин по днях

На другому етапі виконується власне «пакування» годин завдань у вільну місткість днів календаря. На відміну від абстрактної схеми з розділу 2 (де зручно мислити «від дедлайну назад»), реалізація використовує **послідовний**

прохід по днях від початку періоду до кінця (рис 3.1). Логіка при цьому еквівалентна: завдання з ближчим дедлайном потрапляють до розкладу раніше, а буфер зсуває «цільове» завершення на кілька днів до формального дедлайну.

У спрощеному вигляді алгоритм можна подати так:

```
for each task in orderedTasks:
  remaining = task.totalHours

  // спробувати розмістити години до ефективного дедлайну
  for day = 1 .. dEff(task):
    free = calendar[day].capacityHours - loadOfDay[day]
    if free <= 0: continue

    assigned = min(remaining, free)
    addBlock(task, day, assigned)
    remaining -= assigned

    if remaining <= 0: break

  // якщо залишилось – використати буферні дні між dEff і deadlineDay
  for day = dEff(task)+1 .. task.deadlineDay:

  // якщо й цього недостатньо – виносимо залишок за дедлайн
  for day = task.deadlineDay+1 .. numberOfDays:
```

де:

- loadOfDay[day] – поточне сумарне навантаження цього дня (сума hours усіх блоків);
- addBlock(...) – функція, що створює ScheduleBlock із вказаним taskId, dayIndex та hours і додає його до масиву розкладу.

Ключові моменти реалізації:

- **Добові обмеження**

  Для кожного дня використовуються значення capacityHours з CalendarDay. Якщо добова місткість вичерпана (free <= 0), алгоритм просто переходить до наступного дня, не допускаючи перевищення ліміту.

- **Буфер безпеки**

  Практично буфер реалізується як пріоритетне використання днів до ефективного дедлайну; якщо години завдання не вміщуються до ефективного дедлайну, вони «розливаються» на проміжок між ним і deadlineDay, а потім — у майбутні дні.

- **Великі завдання**

  Якщо завдання має велику totalHours, воно автоматично розтягується на кілька днів — на кожному дні виділяється стільки годин, скільки дозволяє вільна місткість, аж поки remaining не стане нулем або не вичерпаються всі доступні дні.

Якщо після проходу всього календаря для деяких задач залишається remaining > 0, це означає, що за наявних обмежень доступного часу певну частину роботи **не вдалося розмістити**. Такі задачі потім потрапляють у показник tasksAtRisk при обчисленні метрик.

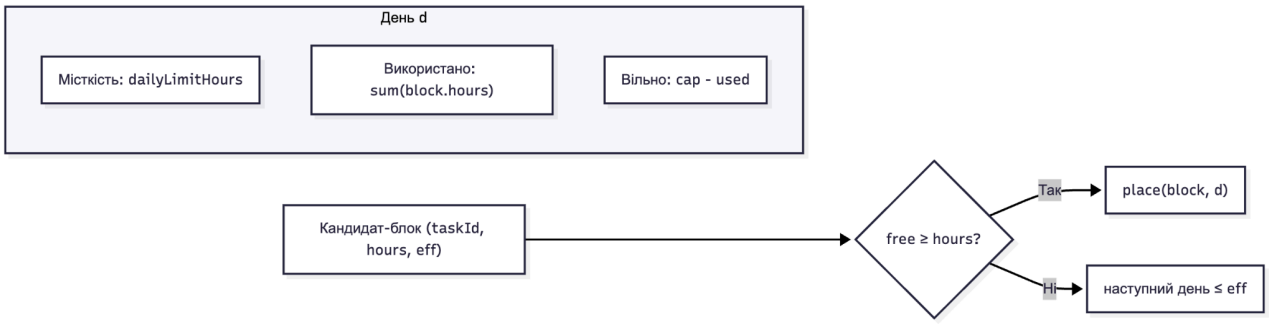


Рисунок 3.1 - схема вибору дня

Формування вихідних даних та метрик

По завершенні проходу:

- Модуль має масив ScheduleBlock[], де кожен блок містить: ідентифікатор завдання (taskId), день (dayIndex), тривалість (hours), ознаку того, чи виходить блок за формальний дедлайн (isAfterDeadline).
- На основі цього масиву обчислюються ScheduleMetrics: сумарні години та середнє навантаження; навантаження по кожному дню та пікове навантаження; кількість перевантажених днів (де навантаження перевищує заданий поріг); кількість завдань, що мають блоки після їх дедлайнів (tasksAtRisk); дисперсія навантаження (для подальшої локальної оптимізації).

Обчислення метрик винесено в окремі утилітні функції, що дозволяє повторно їх використовувати і для етапу оптимізації, і для порівняння різних версій розкладу.

Обробка крайових випадків

Модуль планування містить базові перевірки та «захисні» гілки для типових крайових ситуацій:

- **Порожній список завдань**
  Якщо масив tasks порожній, функція повертає порожній розклад і нульові метрики (усі дні мають навантаження 0).
- **Нульова доступність у календарі**
  Якщо для всіх днів capacityHours = 0, жоден блок не може бути розміщений; усі завдання в такому випадку потрапляють у зону ризику, а розклад залишається порожнім.
- **Некоректні параметри буфера**
  Якщо bufferDays від’ємний або надто великий, на етапі підготовки даних він нормалізується до допустимого діапазону (наприклад, не більше довжини планувального горизонту).

- **Занадто короткий планувальний горизонт**
  Якщо deadlineDay деяких задач виходить за межі numberOfDays, вони обрізаються до останнього дня. Це не ідеальна ситуація, але дозволяє алгоритму коректно відпрацювати й відобразити ризики у метриках.

Таким чином, модуль EDD-core реалізує описаний у розділі 2 базовий метод побудови розкладу: має чітко визначені вхідні дані (список задач і календар), послідовну процедуру розподілу годин з урахуванням дедлайнів, буфера та добових обмежень, а також віддає розклад у формі блоків з повним набором метрик якості. Цей модуль використовується як для початкового планування, так і для повторної побудови розкладу після змін у вхідних даних. Наступний підрозділ описує реалізацію модуля локальної оптимізації, який працює поверх результатів EDD-core.

3.4. Реалізація модуля локальної оптимізації та перепланування

Модуль локальної оптимізації реалізовано у файлі hillClimbing.ts.

Основна функція:

```
optimizeSchedule(
  schedule: ScheduleBlock[],
  tasks: Task[],
  calendar: CalendarDay[],
  bufferDays: number,
  maxIterations: number
): { schedule: ScheduleBlock[]; metrics: ScheduleMetrics; isOptimized: boolean }
```

На вхід вона отримує:

- поточний розклад у вигляді масиву ScheduleBlock[];
- список завдань Task[] і календар CalendarDay[];
- значення буфера bufferDays;

– максимальну кількість ітерацій maxIterations (у поточній реалізації – до 200).

На виході повертається оновлений розклад, перераховані метрики та прапорець isOptimized, що дозволяє інтерфейсу показувати, що до цього варіанта розкладу вже застосовували оптимізацію.

Функція оцінки якості розкладу

Відповідно до формалізації з розділу 2, якість розкладу оцінюється за кількома критеріями: кількість перевантажених днів, завдання «в зоні ризику», пікове навантаження, дисперсія добового навантаження тощо. У модулі оптимізації це зведено до єдиної скалярної функції оцінки:

score = overloadedDays * 1000 + tasksAtRisk * 500 + peakLoad * 50 + loadVariance * 20;

– overloadedDays – кількість днів, де навантаження перевищує порогове значення (наприклад, > 4 год/день);
– tasksAtRisk – кількість завдань, частина яких опинилась після дедлайну або не помістилася в розклад;
– peakLoad – максимальне добове навантаження;
– loadVariance – дисперсія навантаження по днях.

Таке вагове поєднання відображає пріоритети:

– у першу чергу – **усунення перевантажень** (великий коефіцієнт 1000);
– далі – **зменшення ризикових завдань**;
– потім – зниження пікового навантаження;
– і нарешті – більш рівномірний розподіл (дисперсія).

Усі ці величини розраховуються через утилітні функції модуля utils.ts, зокрема calculateScheduleMetrics, calculateDayLoads, findTasksAtRisk.

Режими роботи алгоритму

Реалізація Hill Climbing підтримує два режими, які автоматично перемикаються в залежності від стану розкладу:

- **Режим виправлення перевантажень**

  Використовується, якщо є дні з перевищенням порогового навантаження (overloadedDays > 0): джерелами (fromDay) обираються **перевантажені дні**; цільовими днями (toDay) – дні з вільною місткістю (capacityHours - load > 0); переміщення блоків суворо контролюється так, щоб **не погіршити дедлайни**, тобто блоки не повинні створювати нові «ризикові» завдання.

  Основна мета – зняти піки, не порушуючи дедлайнів.

- **Режим балансування**

  Якщо перевантажених днів більше немає, алгоритм переходить у режим згладжування: джерела – умовний «топ 50%» найбільш завантажених днів; цілі – «топ 50%» найменш завантажених; допускається перенесення блоків так, щоб покращити рівномірність, навіть якщо окремі блоки наближаються до ефективних дедлайнів (але все одно обмежено функцією оцінки й перевірками).

У кожному режимі алгоритм працює з **випадково обраними блоками й днями** в заданих множинах, що дозволяє уникнути «жорсткого» детермінованого сценарію і дає шанс знайти різні локальні покращення.

Процедура Hill Climbing у реалізації

Реалізована процедура відповідає загальному опису з теоретичної частини (розділ 2.4), але у практичному коді має чіткі технічні обмеження та евристики. Загальний цикл виглядає так (рис. 3.2):

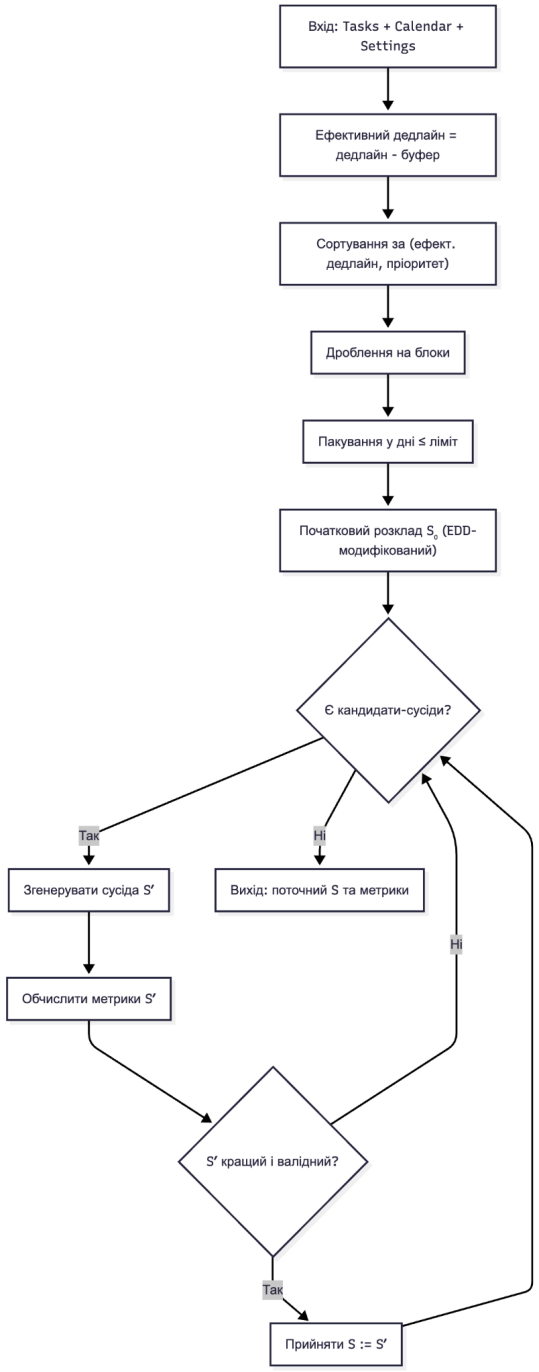


Рисунок 3.2 - загальний алгоритм оптимізації

- **Оцінка початкового стану.** На основі вхідного schedule обчислюються ScheduleMetrics та початковий score.
- **Ітеративне покращення (до maxIterations).** На кожній ітерації алгоритм:
  - Перераховує актуальні метрики та визначає **режим роботи**:
    - є перевантажені дні → режим виправлення;

    - перевантажених днів немає → режим балансування.
  - Вибирає кандидатні дні:
    - множину fromDays (перевантажені або просто найбільш завантажені);
    - множину toDays (дні з вільною місткістю або найбільш «легкі»).
  - Випадковим чином обирає блок із дня fromDay і пробує перенести частину його годин у день toDay.

    Перенесення обмежене:
    - не більше **50% годин** цього блоку;
    - не менше 0.5 години (щоб зміни були «відчутними»).
  - Формує **кандидатний розклад** schedule' і для нього повторно обчислює метрики та score'.
  - Якщо score' < score (тобто розклад покращився за обраною функцією оцінки) – зміна **приймається**:
    - поточний розклад оновлюється;
    - метрики та score замінюються на нові.
  - Якщо жодного прийнятного покращення не вдалося знайти протягом певної кількості спроб, або досягнуто maxIterations, цикл завершується – вважається, що досягнуто локального мінімуму.
- **Повернення результату.**
  - Модуль повертає оновлений розклад, метрики та isOptimized: true.

Таким чином, реалізований Hill Climbing дотримується класичної схеми «покращуй, поки можеш», але з чіткими технічними обмеженнями на кількість ітерацій і розмір кроків, що забезпечує прийнятний час роботи для інтерактивного застосунку.

3.5. Користувацький інтерфейс і сценарії роботи користувача

Програмний засіб реалізовано як односторінковий веб-застосунок із декількома логічними розділами, між якими користувач перемикається через верхнє або бокове меню (рис. 3.3).

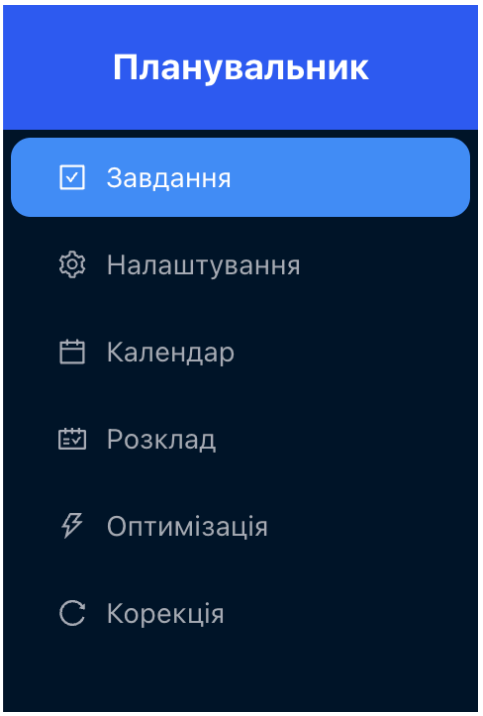


Рисунок 3.3 - меню навігації

Основні екрани:

- **«Мої завдання»** – введення й редагування переліку навчальних задач;
- **«Налаштування»** – конфігурація періоду планування та загальних параметрів (буфер, лише будні тощо);
- **«Календар»** – налаштування доступних годин по кожному дню;
- **«Розклад»** – запуск генерації та перегляд побудованого розкладу з основними метриками;
- **«Оптимізація»** – запуск локальної оптимізації й порівняння розкладу до/після покращення.

Усі елементи інтерфейсу побудовані на базі бібліотеки Ant Design (таблиці, форми, модальні діалоги) з використанням Tailwind CSS для стилізації, що забезпечує однорідний зовнішній вигляд і швидкий відгук інтерфейсу.

Екран «Мої завдання»

Екран «Мої завдання» призначений для введення та підтримки актуального списку навчальних задач (рис. 3.4).

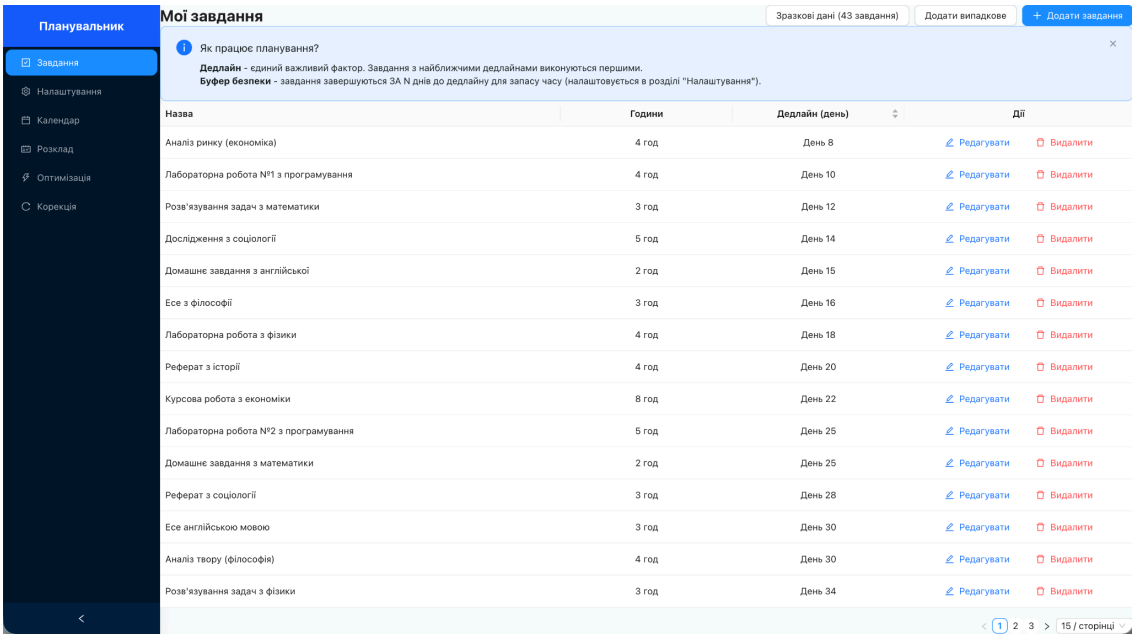


Рисунок 3.4 - екран завдань

Основні елементи:

- кнопка **додавання завдання**, що відкриває модальне вікно з формою;
- табличний список завдань із такими колонками:
  - назва завдання;
  - тривалість у годинах;
  - дедлайн (у вигляді календарної дати);
  - службові дії (редагування, видалення).
- можливість сортування списку щонайменше за дедлайном, щоб найближчі завдання були «вище» у таблиці.

У формі додавання/редагування (рис 3.5) користувач заповнює:

- **назву** завдання;
- **орієнтовну тривалість** (totalHours);
- **дедлайн** (календарна дата всередині обраного періоду);

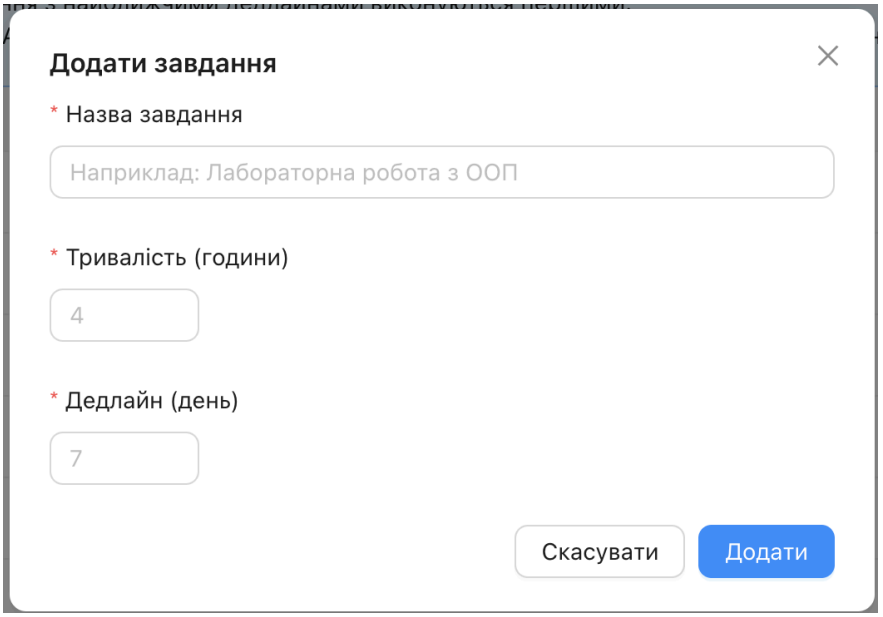


Рисунок 3.5 - екран додавання завдань

Поле пріоритету в моделі даних присутнє, однак у поточній версії інтерфейсу воно не виведене як «обов'язкова» опція, а всі завдання створюються з однаковим значенням пріоритету. Це відповідає вибору математичної моделі без явних ваг задач.

Таким чином, екран «Мої завдання» забезпечує повний CRUD-набір операцій над сутністю Task і виступає вихідною точкою формування вхідних даних для модуля планування.

Екрани «Налаштування» та «Календар»

Екран «Налаштування»

У цьому розділі (рис. 3.6) користувач задає загальні параметри планування:

- **кінцеву дату періоду** (наприклад, кінець семестру або сесії);
- опцію **«лише будні дні»** (якщо увімкнено – календар початково заповнюється тільки понеділком–п'ятницею);
- перелік **додатково виключених дат** (свята, відрядження, інші дні без навчання);

– **кількість буферних днів** до дедлайнів (bufferDays), яка впливає на те, наскільки раніше дедлайну система намагатиметься завершувати завдання.

Після збереження налаштувань автоматично перераховується довжина планувального горизонту й ініціалізується масив днів у календарі.

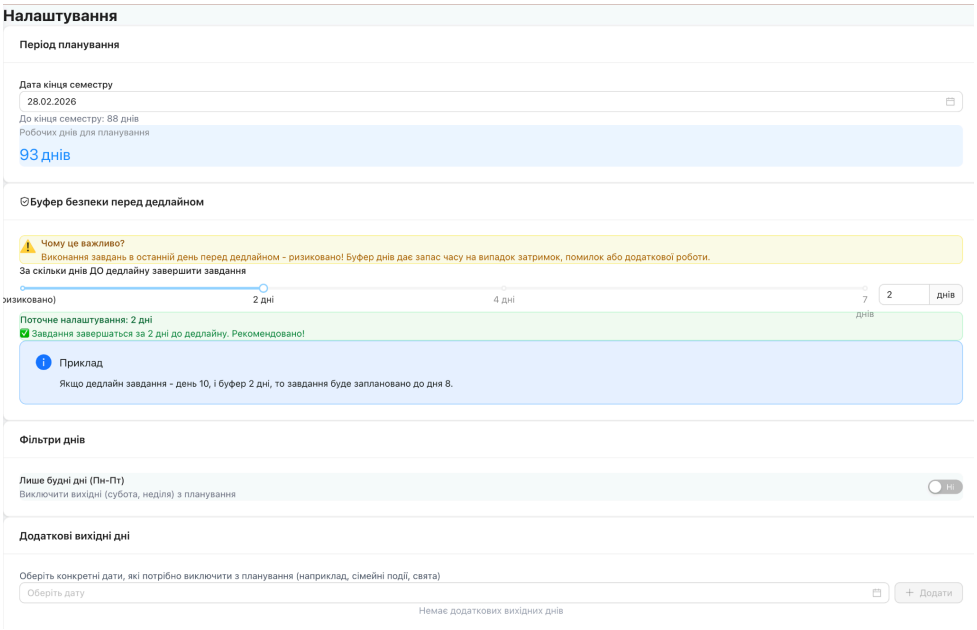


Рисунок 3.6 - екран налаштувань

Екран «Календар»

Цей екран (рис. 3.7) відображає таблицю днів планувального горизонту (CalendarDay[]) з такими полями:

– дата;
– номер дня (dayIndex);
– доступні години навчання (capacityHours);
– ознака «день виключений» (isExcluded).

Користувач може:

– змінити кількість доступних годин для конкретного дня;
– повністю виключити день із планування (наприклад, встановивши isExcluded = true);
– скористатися масовими операціями (заповнення однакового значення годин для всіх робочих днів тощо).

Календар служить джерелом значень добових лімітів capacityHours, які суворо дотримуються модулем EDD-core при побудові розкладу.

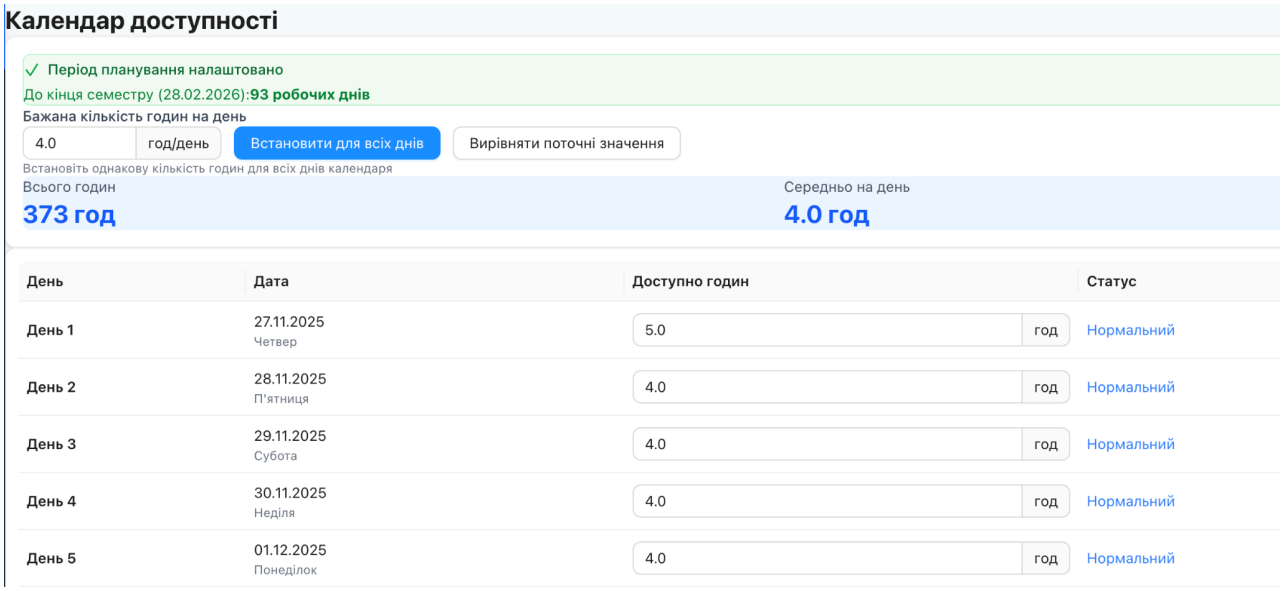


Рисунок 3.7 - екран налаштувань календаря

Екран «Розклад»

На цьому екрані (рис. 3.8) користувач запускає процес побудови розкладу й переглядає його результат.

Основні елементи:

- кнопка **«Згенерувати розклад»**, яка:
  - перевіряє, чи є в системі завдання та коректно налаштований календар;
  - викликає через сховище функцію generateInitialSchedule(...);
  - на час виконання блокує повторний запуск і показує індикатор прогресу.
- блок із **ключовими метриками розкладу** у вигляді карток:
  - загальна кількість годин;
  - середнє добове навантаження;
  - пікове навантаження;
  - кількість перевантажених днів;

  - кількість завдань у зоні ризику;
  - дисперсія навантаження.
- візуалізація розкладу по днях:

  як правило, у вигляді сітки або списку «карток дня», де кожна картка містить:
  - дату та індикатор навантаження (наприклад, прогрес-бар відносно capacityHours чи маркер перевищення);
  - перелік завдань, запланованих на цей день, з вказаною кількістю годин;
  - окреме підсвічування днів, у яких є блоки після дедлайнів.

Якщо завдань немає або календар не налаштовано, користувач отримує відповідне повідомлення із пропозицією повернутися до відповідних розділів.

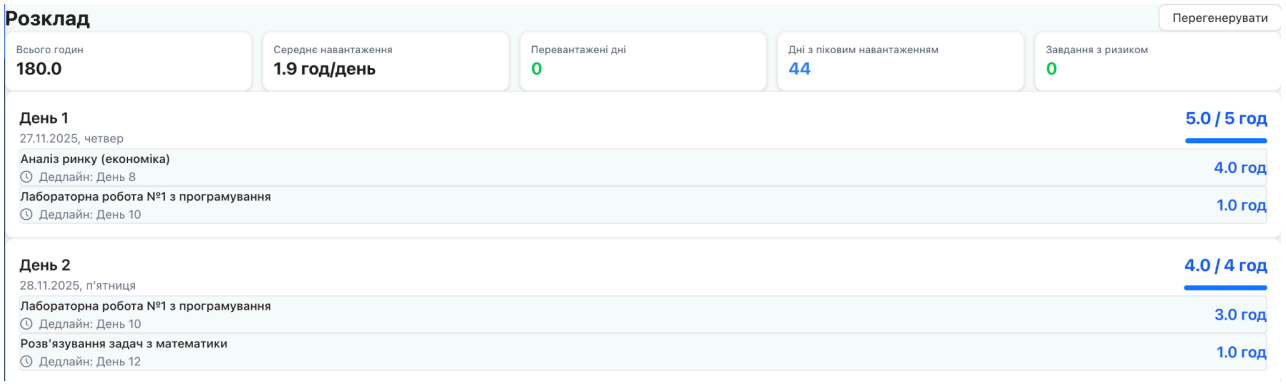


Рисунок 3.8 - екран розкладу

Екран «Оптимізація»

Цей екран (рис 3.9) надає інтерфейс до модуля локальної оптимізації Hill Climbing.

- Користувач бачить поточні метрики розкладу «до оптимізації» (як і на екрані «Розклад»).
- Кнопка **«Оптимізувати розклад»** запускає функцію optimizeSchedule(...) через сервісний шар; на час виконання кнопка блокується, відображається індикатор процесу.
- Після завершення оптимізації показуються:
  - оновлені метрики «після оптимізації»;

– різниця між показниками (наприклад, зменшення кількості перевантажених днів або пікового навантаження).

Це дозволяє студенту в наочній формі побачити, як застосування оптимізатора вплинуло на рівномірність та безпечність розкладу.

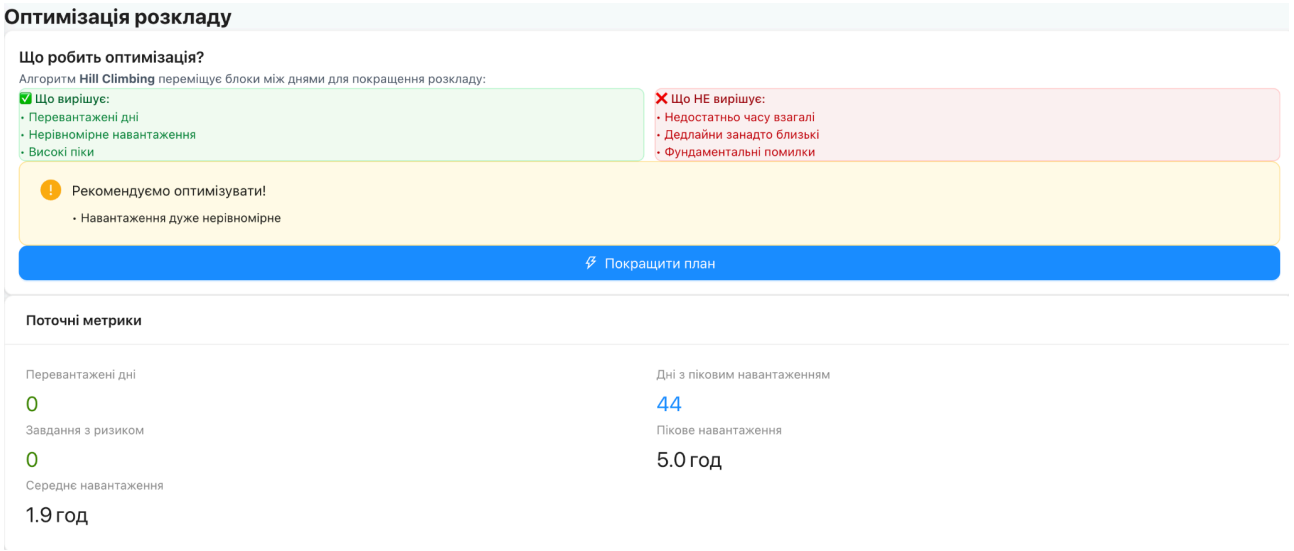


Рисунок 3.9 - екран оптимізації

Типові сценарії роботи користувача

У рамках розробленого інтерфейсу можна виділити кілька типових сценаріїв використання для тестування. Тестування проведено у ручному режимі.

Сценарій 1. Початкове налаштування й побудова першого розкладу

– Користувач відкриває застосунок, переходить до **«Налаштування»** й задає:
  – кінець періоду (наприклад, дата екзаменаційної сесії);
  – режим лише будніх днів (за бажанням);
  – кількість буферних днів до дедлайнів.
– На екрані **«Календар»** уточнює доступні години на кожен день, коригуючи вихідні, свята та «завантажені» дні.
– На екрані **«Мої завдання»** додає всі актуальні навчальні задачі з дедлайнами та оцінками тривалості.

- На екрані **«Розклад»** натискає «Згенерувати розклад» і отримує базовий поденний план роботи.

За потреби користувач одразу може перейти до **«Оптимізації»**, щоб згладити піки навантаження.

Сценарій 2. Щоденне використання та оновлення плану

- Студент щодня відкриває **«Розклад»** і дивиться, які завдання й у якій кількості годин заплановані на сьогодні.
- Після фактичного виконання задач він:
  - або видаляє виконані завдання з розділу **«Мої завдання»**;
  - або коригує їх (наприклад, зменшує тривалість або змінює дедлайн, якщо завдання частково перенесене).
- За потреби студент змінює доступні години на майбутні дні у **«Календарі»** (наприклад, якщо з’явилася нова робота чи інші зобов’язання).
- Потім знову переходить до **«Розкладу»** і запускає генерацію нового плану на оновленому наборі даних.

Таким чином, перепланування реалізується через **повторну побудову** розкладу з урахуванням виконаних завдань і зміненого календаря.

Сценарій 3. Аналіз якості розкладу

- Після побудови або оптимізації розкладу користувач аналізує блок метрик на екрані **«Розклад»**:
  - чи немає надто великої кількості перевантажених днів;
  - яке максимальне навантаження;
  - чи є завдання в зоні ризику прострочення.
- Якщо показники незадовільні, студент може:
  - змінити буфер до дедлайнів;
  - скоригувати дату завершення періоду або доступні години;
  - перегенерувати розклад і, за потреби, повторно застосувати оптимізацію.

Такий цикл «налаштування → планування → аналіз → корекція» відображає практичний спосіб використання системи упродовж семестру й пов’язує алгоритмічні рішення з реальними діями користувача.

У підсумку, користувацький інтерфейс забезпечує послідовну підтримку всіх основних етапів роботи з персональним планом: від введення задач і ресурсів до побудови, аналізу та покрокового вдосконалення індивідуального розкладу навчального навантаження.

3.6. Експериментальні дослідження та оцінка ефективності

Метою експериментальних досліджень є оцінка того, наскільки модифікований метод EDD та додаткова локальна оптимізація впливають на якість індивідуального розкладу студента у порівнянні з класичним алгоритмом EDD без ресурсних обмежень. Для порівняння використовувався один і той самий набір завдань та календар доступності, а змінювався лише спосіб побудови розкладу.

Якість розкладу оцінювалась за набором метрик, що обчислюються модулем аналізу розкладу:

- Середнє погодинне добове навантаження (год/день)
- Пікове навантаження у день (год)
- Перевантажені дні (за замовчуванням > 4 год) (шт)
- Дні з піковим навантаженням коли усі наявні години використані (шт)
- Завдання з ризиком прострочки (шт)
- Дисперсія добового навантаження (год²)

$$\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 \tag{3.1}$$

де:

x_i - навантаження в день i (кількість годин)

$\bar{x}$ - середнє навантаження по всіх днях

n - кількість днів у календарі

Порівняння підходів (класичний EDD, модифікований EDD, EDD + hill climbing)

У дослідженні порівнювалися три варіанти побудови розкладу:

- **Класичний EDD**

  Завдання впорядковуються за зростанням дедлайнів, після чого послідовно «укладаються» у календар без обмеження на кількість годин у день і без буфера до дедлайну. Такий підхід добре мінімізує запізнення відносно термінів, однак повністю ігнорує реалістичність щоденного навантаження студента.

- **Модифікований EDD (з буфером та добовим лімітом)**

  В якості бази також використовується Earliest Due Date, але для кожного завдання вводиться **ефективний дедлайн** (дедлайн мінус буфер k днів), а календар містить для кожного дня обмеження місткості (кількість доступних годин). Алгоритм розбиває великі завдання на блоки та розподіляє їх по днях так, щоб не перевищувати добовий ліміт.

- **Модифікований EDD + локальна оптимізація (hill climbing)**

  На третьому етапі поверх розкладу, отриманого модифікованим EDD, запускається процедура локальної оптимізації. Оператор сусідства виконує перестановки та перенесення блоків між днями у межах допустимого діапазону, не порушуючи дедлайнів та добових лімітів. Цільова функція мінімізує пікове навантаження, кількість перевантажених днів та дисперсію розподілу годин між днями.

У таблиці 3.7 наведено порівняння значень основних метрик для трьох підходів.

Таблиця 3.7 – Порівняння якості розкладу для різних підходів

Метрика	**Класичний EDD**	**Модифікований EDD (до оптимізації)**	**Модифікований EDD + hill climbing**
Середнє навантаження, год/день	2,9	2,9	2,9
Пікове навантаження, год у найгірший день	12,0	4,0	4,0
Перевантажені дні (> 4 год), шт	12	0	0
Дні з піковим погодинним навантаженням, шт	22	45	14
Дисперсія добового навантаження, год²	5,5	3,27	0,44
Завдання з ризиком прострочки, шт	0	0	0

Як видно з таблиці, усі три підходи мають однакове середнє добове навантаження, що природно, оскільки загальна кількість годин роботи над завданнями не змінюється. Відмінності проявляються саме у структурі розподілу навантаження по днях.

Основні відмінності можна побачити у наступних полях:

Пік: **12 → 4 год;** Дні з піковим навантаженням: **45 → 14;** Дисперсія: **5.5 → 0.44 год²**

Аналіз отриманих результатів

Отримані результати зведені в табл. 3.7 і наочно проілюстровані на рис. 3.10–3.12.

По осі абсцис на всіх графіках відкладено три підходи:

– Класичний EDD;
– Модифікований EDD (з буфером та добовим лімітом, без оптимізації);
– Модифікований EDD + hill climbing.

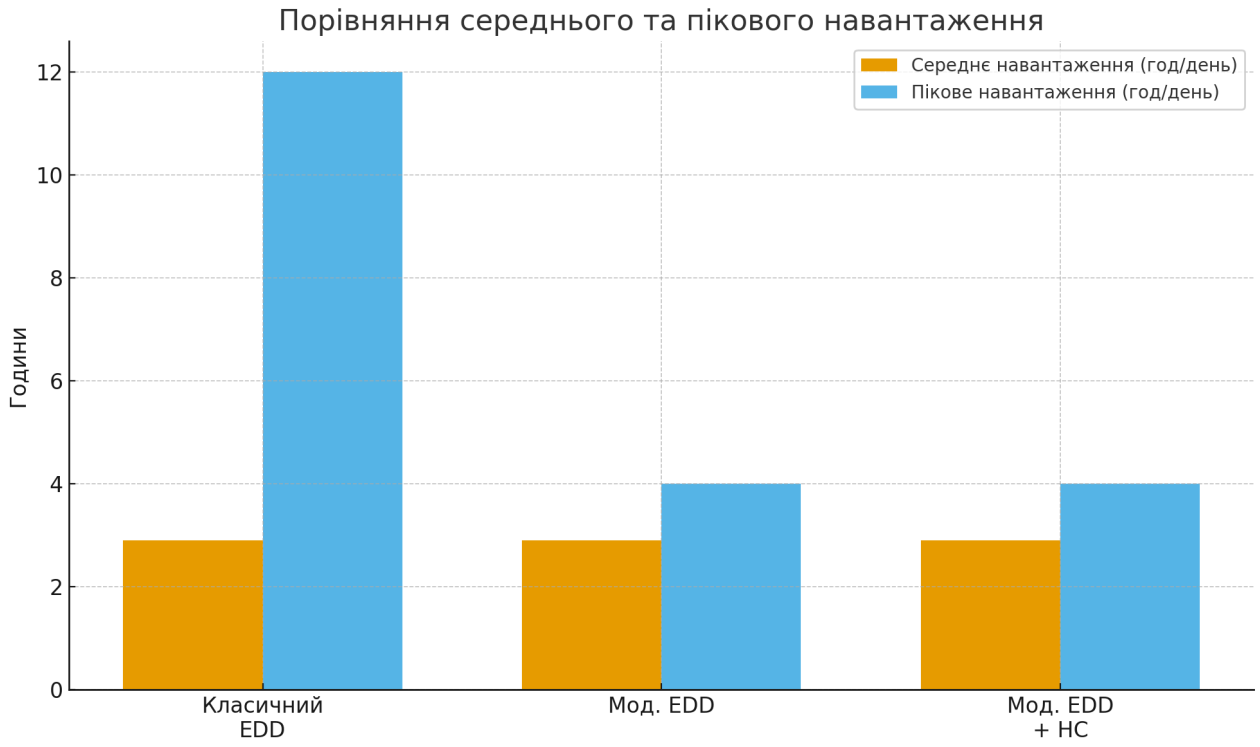


Рисунок 3.10 - Пікове навантаження та середнє навантаження

На рис. 3.10 показано порівняння середнього та пікового навантаження для трьох підходів.

– **Середнє добове навантаження** у всіх випадках однакове й дорівнює 2,9 год/день. Це очікувано, оскільки сумарний обсяг робіт (кількість годин по всіх завданнях) не змінюється ні при модифікації алгоритму, ні при локальній оптимізації.
– **Пікове навантаження**:

– для класичного EDD досягає **12 годин** у найгірший день, що фактично неможливо для реального студента й означає суттєве перевантаження;
– для модифікованого EDD та EDD + hill climbing пікове навантаження обмежено **4 годинами**, що відповідає заданому добовому ліміту.

Це наочно демонструє головний ефект модифікації методу: при тому самому середньому навантаженні план стає більш рівномірним — жоден день не вимагає нереалістичної кількості годин.

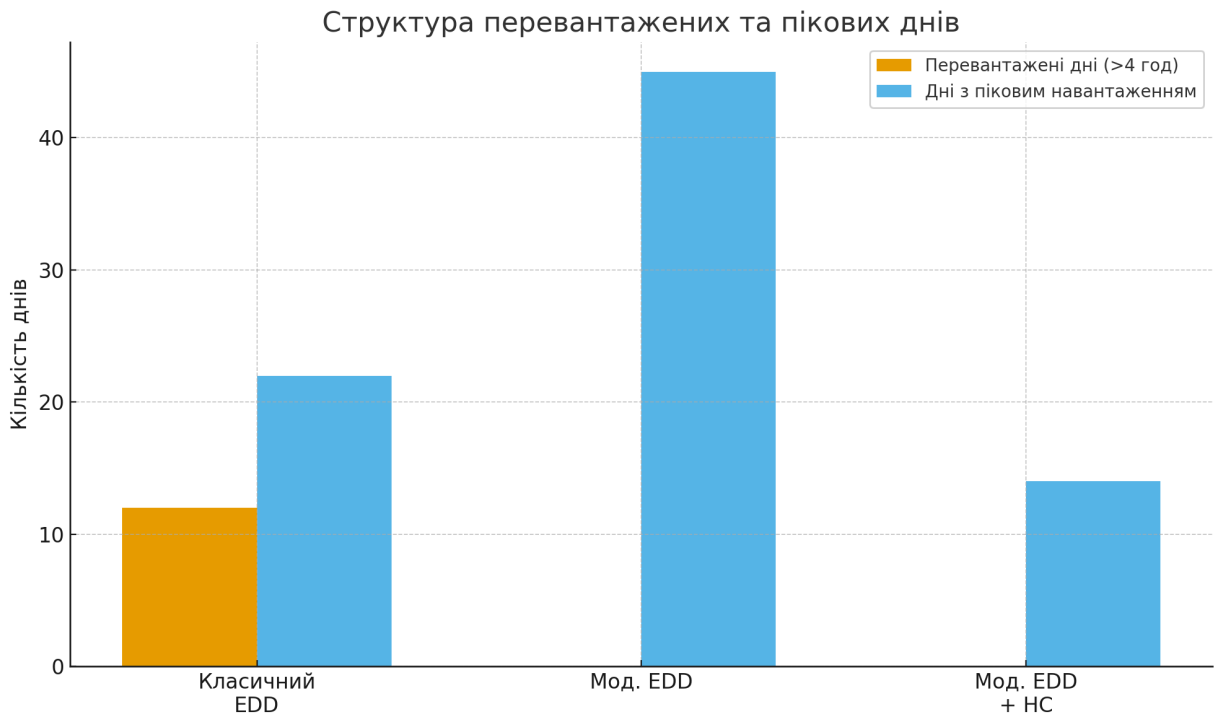


Рисунок 3.11 - перевантажені та пікові дні

На рис. 3.11 наведено кількість **перевантажених днів** (де навантаження перевищує 4 години) та **днів із піковим навантаженням** (днів, у яких використано весь доступний добовий ресурс).

– Для **класичного EDD**:

  - **12 перевантажених днів** – тобто в значній частині періоду студент змушений працювати більше, ніж допустимий поріг;
  - **22 дні з піковим навантаженням**, що свідчить про сильні коливання навантаження (часто «все або нічого»).
- Для **модифікованого EDD**:
  - **0 перевантажених днів** – добовий ліміт спрацьовує як жорстке обмеження;
  - **45 днів з піковим навантаженням** – календар максимально щільно «забитий», майже кожен робочий день використовується повністю.
- Для **EDD + hill climbing**:
  - **0 перевантажених днів** зберігається;
  - кількість «пікових» днів зменшується до **14**, тобто системі вдається перерозподілити години так, щоб частина днів мала навантаження нижче ліміту, без створення нових перевантажень.

Іншими словами, базовий модифікований EDD перетворює план на «щільний, але безпечний» (усі дні майже під зав’язку), тоді як додавання локальної оптимізації робить графік більш комфортним: залишається запас по часу в окремі дні, де студент може відпочити або виконати непередбачені завдання.

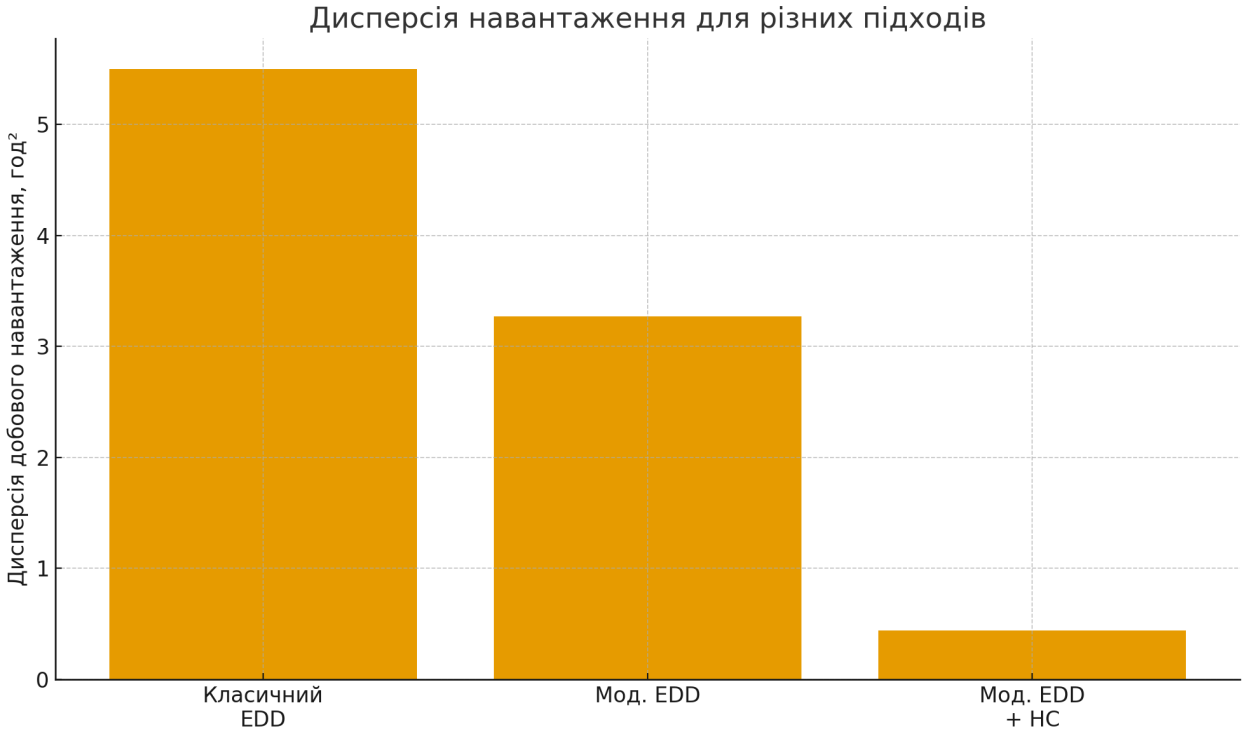


Рисунок 3.12 - дисперсія навантаження

На рис. 3.12 показано значення **дисперсії добового навантаження** для трьох підходів:

- Класичний EDD: **5,5 год²**;
- Модифікований EDD: **3,27 год²**;
- Модифікований EDD + hill climbing: **0,44 год²**.

Дисперсія інтерпретується як числова міра «нерівномірності» навантаження:

- висока дисперсія означає, що є дуже «важкі» й дуже «легкі» дні;
- низька дисперсія – що навантаження по днях приблизно однакове.

Перехід від 5,5 до 3,27 уже свідчить про значне згладжування піків завдяки введенню добових лімітів і буфера, проте **лише поєднання модифікованого EDD з локальною оптимізацією** дозволяє досягти майже рівномірного розкладу із дисперсією 0,44. Фактично це означає, що відхилення від середнього значення 2,9 год/день у більшості днів є незначними.

Якщо інтерпретувати ці результати не тільки формально, а й з погляду повсякденного навчання:

- класичний EDD «заганяє» студента в режим **«12 годин сьогодні – нічого завтра»**, що може бути прийнятно на коротких проміжках, але небезпечно для тривалого періоду (вигорання, зрив дедлайнів, накопичення стресу);
- модифікований EDD гарантує відсутність днів із надмірним навантаженням, але практично **вичерпує весь доступний час** майже щодня — студент живе в постійному стані «на межі»;
- модифікований EDD + hill climbing дає збалансований варіант: усі дедлайни витримано, **немає перевантажених днів**, а також з'являється низка днів із меншою кількістю годин, які можуть слугувати буфером на випадок непередбачених задач або просто як «дихальні паузи».

Таким чином, графіки підтверджують висновки з табличного аналізу: при однаковому середньому навантаженні запропонований метод (модифікований EDD із локальною оптимізацією) формує якісно інший, більш здоровий і стійкий у часі індивідуальний навчальний розклад. Це безпосередньо відповідає поставленій у роботі меті – **покращення ефективності планування навчального процесу студента**.

Отримані результати демонструють, що:

- **Класичний EDD** формує математично «ефективний» розклад щодо дедлайнів, але створює **нереалістичні піки навантаження**: у найгірший день студент має виконати до 12 годин роботи, а кількість днів з перевищенням порогового значення 4 години сягає 12. Дисперсія навантаження по днях є найбільшою, що відповідає сильно нерівномірному графіку.
- **Модифікований EDD** із буфером та добовим лімітом усуває перевантажені дні: пікове навантаження обмежується 4 годинами, а кількість перевантажених днів стає нульовою. При цьому значно зменшується дисперсія добового навантаження (з 5,5 до 3,27), однак

кількість днів, у яких використано всю доступну місткість, збільшується до 45 – це свідчить про те, що розклад хоча й реалістичний, але ще досить «жорсткий» щодо заповнення календаря.

– **Додавання hill climbing** дозволяє зберегти переваги модифікованого EDD (відсутність перевантажених днів, те саме пікове навантаження 4 години), але суттєво покращує рівномірність розкладу: кількість днів із піковим навантаженням зменшується з 45 до 14, а дисперсія добового навантаження падає до 0,44 год². Це означає, що година роботи студента розподіляється по календарю набагато рівномірніше, без «гарячих» тижнів та різких стрибків між днями.

Важливо, що кількість задач з ризиком прострочки в усіх трьох випадках дорівнює нулю, тобто навіть класичний EDD за обраним набором даних встигає укластися в дедлайни. Перевага запропонованого методу проявляється не в «успішності» щодо термінів, а в **якості та комфортності** розкладу для студента: відсутності перевантажених днів та суттєвому зниженні варіативності щоденного навантаження.

Графічне подання результатів (рис 3.13) додатково ілюструє, що модифікація EDD та подальша локальна оптимізація послідовно покращують кожну з обраних метрик якості розкладу.

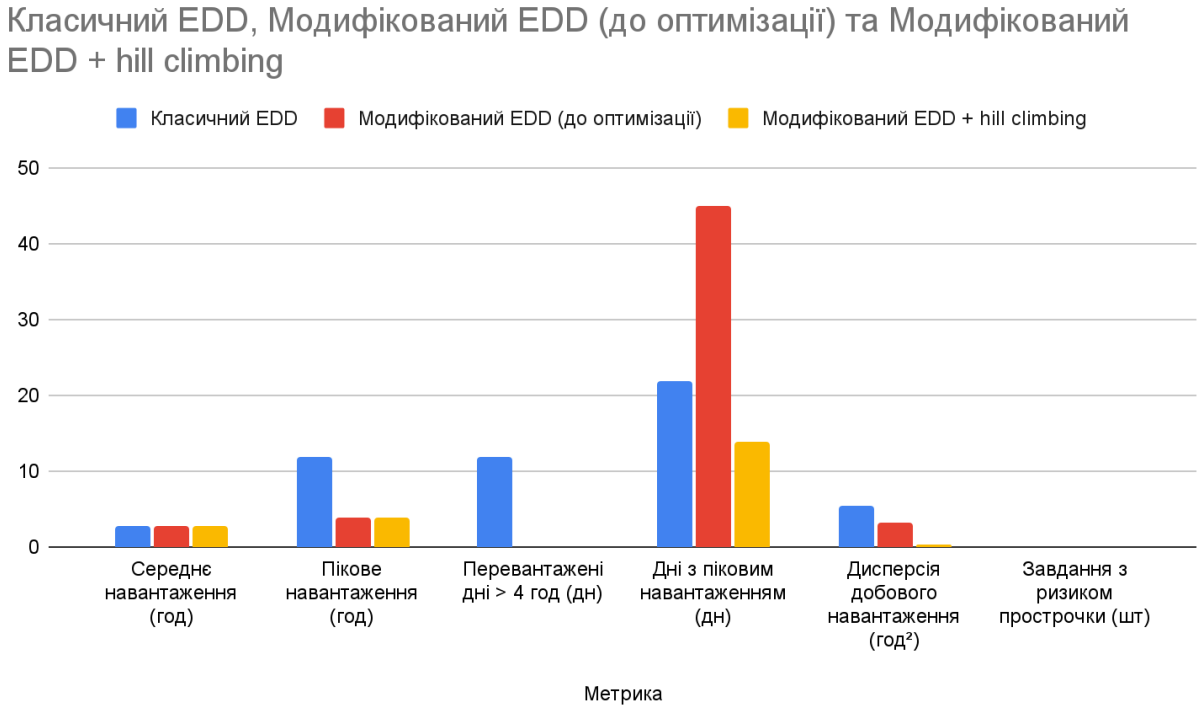


Рисунок 3.13 - гістограма з проаналізованими метриками

Узагальнюючи, проведений експеримент показує, що запропонований підхід дозволяє зберегти потрібний обсяг роботи, але **перетворити його на більш рівномірний та реалістичний план**, який краще відповідає обмеженням часу студента й зменшує ризик накопичення стресу наприкінці семестру.

3.7. Висновки до 3 розділу

У цьому розділі було описано програмну реалізацію методу автоматизованого планування навчального навантаження, сформульованого в розділі 2. Показано, що запропонований підхід повністю втілено у вигляді односторінкового веб-застосунку, який працює на боці клієнта й не потребує окремого серверного компонента. Архітектура системи побудована за шаровим принципом: презентаційний шар (інтерфейс на основі React та Ant Design), шар управління станом (Zustand-store), алгоритмічне ядро (модулі EDD-core та локальної оптимізації), а також шар локального зберігання даних у localStorage. Такий поділ спрощує розвиток застосунку та дозволяє в перспективі винести

алгоритмічні обчислення на сервер без зміни логіки користувацького інтерфейсу.

Запропонована модель даних (сутності Task, CalendarDay, ScheduleBlock, ScheduleMetrics, UserSettings) безпосередньо відображає математичну постановку задачі: тривалості завдань, дедлайни, денні ліміти, буфер до дедлайнів та метрики якості мають явні програмні представлення. Модуль планування (EDD-core) реалізує модифікований алгоритм Earliest Due Date з урахуванням добових обмежень і буфера, розподіляє години завдань по днях і формує початковий розклад у вигляді набору блоків. Поверх цього розкладу працює модуль локальної оптимізації за схемою hill climbing, який за допомогою локальних переносів годин між днями зменшує кількість перевантажених днів, пікове навантаження та нерівномірність розподілу.

Користувацький інтерфейс забезпечує послідовну підтримку всього циклу роботи зі студентським планом: від введення завдань і налаштування календаря доступності до генерації, аналізу та поліпшення розкладу. Основні сценарії (початкове налаштування, щоденне використання, повторне планування) були перевірені шляхом ручного тестування, що підтвердило коректність взаємодії між модулями й відповідність поведінки системи поставленим функціональним вимогам. У сукупності описані рішення демонструють, що розроблений програмний засіб є працездатною реалізацією запропонованого методу та може використовуватися як основа для подальшого розширення функціональності.

Отримані в ході експериментальних досліджень результати підтвердили доцільність запропонованих модифікацій базового методу EDD. Порівняння трьох підходів показало, що класичний EDD, хоч і забезпечує вкладання в дедлайни, формує нереалістичний для студента розклад з піковим навантаженням до 12 годин на день та значною кількістю перевантажених днів. Запровадження добового ліміту та буфера до дедлайнів у модифікованому EDD повністю усуває перевантажені дні та зменшує дисперсію добового

навантаження, але призводить до дуже щільного заповнення календаря, коли більшість днів використовує доступний час повністю.

Додавання етапу локальної оптимізації за схемою hill climbing дозволило зберегти всі переваги модифікованого EDD (обмеження пікового навантаження до 4 годин і відсутність перевантажених днів), водночас суттєво покращивши рівномірність розподілу роботи в часі. Кількість днів із максимальною завантаженістю зменшилася, а дисперсія добового навантаження стала майже вдесятеро меншою порівняно з класичним EDD, що означає практично рівний обсяг роботи протягом усього періоду планування. Таким чином, модифікований EDD у поєднанні з локальною оптимізацією забезпечує не лише формальне дотримання дедлайнів, а й якісно більш комфортний та стійкий для студента план роботи, що і було ключовою ціллю розробленого програмного засобу.

4. МАРКЕТИНГОВИЙ АНАЛІЗ СТАРТАП-ПРОЄКТУ ТА ПЛАН ВПРОВАДЖЕННЯ РІШЕННЯ

4.1. Опис ідеї проєкту

Ідея стартап-проєкту полягає у створенні веб-застосунку – особистого планувальника навчального навантаження студента, який автоматично формує збалансований розклад виконання навчальних завдань на основі модифікованого алгоритму EDD та процедур локальної оптимізації (hill climbing). Застосунок враховує індивідуальний графік студента, дедлайни, трудомісткість завдань та обмеження по часу, мінімізуючи піки навантаження, ймовірність зриву дедлайнів і пов’язаний зі стресом ризик.

На відміну від універсальних таск-менеджерів та календарів, запропонований продукт орієнтований саме на навчальний процес: роботу з предметами, лабораторними й курсовими, модулями, сесією, повторенням матеріалу. В перспективі продукт може виступати як окремий B2C-сервіс для студентів та як модуль індивідуального планування в LMS університетів або EdTech-платформ.

Таблиця 4.1 – Опис ідеї стартап-проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Розробка веб-застосунку – особистого планувальника навчального навантаження, який автоматично будує збалансований розклад виконання навчальних завдань з урахуванням дедлайнів, оцінюваної трудомісткості, індивідуального графіка студента та обмежень за часом. В основі лежить модифікований алгоритм EDD з локальною оптимізацією розкладу.	Індивідуальне планування навчального навантаження студентів ЗВО (бакалаври, магістри). Підтримка студентів із гнучким / заочним графіком, поєднання навчання і роботи. Інтеграція як модуль до LMS університетів (Moodle тощо) та онлайн-платформ (курси, Bootcamp-и). Використання як навчальний приклад для дисциплін з оптимізаційних алгоритмів і веб-розробки.	Зменшення піків навантаження та ризику зриву дедлайнів. Підвищення передбачуваності та керованості навчального процесу для студента. Зниження рівня стресу за рахунок прозорого плану дій на кожен день/тиждень. Економія часу на ручне планування і перепланування завдань. Можливість швидко адаптувати розклад при зміні графіка чи появі нових завдань.

Для обґрунтування конкурентних переваг ідеї виконаємо порівняння з існуючими рішеннями – загальними таск-менеджерами (Trello, Asana, Notion), календарями (Google Calendar), а також профільними study-planner’ами.

Таблиця 4.2 – Порівняння техніко-економічних характеристик ідеї з конкурентами *(W – слабка сторона, N – нейтральна, S – сильна)*

№	Техніко-економічні характеристики ідеї	Мій проєкт	Google Calendar + Tasks	Trello / Notion	Типові study-planner'и
1	Автоматичне побудова розкладу за дедлайнами та трудомісткістю (оптимізаційний алгоритм, а не просто список задач)	S	W (лише нагадування і події, без оптимізації)	W–N (можливо вручну задати, але без оптимізації)	N (частково планують, але без формальної оптимізації)
2	Урахування індивідуального графіка доступності (вікон, роботи, особистих справ) при складанні плану	S	W (події можна ставити, але автоматичної адаптації немає)	W (гнучкі дошки, але без часової моделі)	N
3	Автоматичне перепланування при зриві дедлайнів / зміні обмежень	N	W	W	W–N
4	Орієнтація на навчальні завдання (модулі, сесія, курсові, повторення) і профільні метрики	S	W	W	N–S (частина орієнтована на навчання, але без глибоких метрик)
5	Прозорість і пояснюваність рекомендацій (пояснення «чому саме так сплановано день»)	S	W	W	N

Кінець таблиці 4.2

№	Техніко-економічні характеристики ідеї	Мій проєкт	Google Calendar + Tasks	Trello / Notion	Типові study-planner’и
6	Простота впровадження (web, без потреби в складній інфраструктурі)	S	S	S	S
7	Можливість інтеграції з LMS та календарями (API, експорт/імпорт подій)	N	S (шляхом сторонніх інтеграцій)	N	N

Узагальнюючи, запропонований проєкт займає нішеву позицію між універсальними таск-менеджерами та класичними study-planner’ами, роблячи наголос на формалізованому алгоритмічному плануванні та адаптивному перепланування.

4.2. Технологічний аудит ідеї проєкту

Реалізація продукту ґрунтується на сучасних та добре задокументованих веб-технологіях, які є доступними та поширеними на ринку розробки програмного забезпечення:

- клієнтська частина: React + TypeScript;
- збереження стану: локальне сховище браузера / потенційний перехід до хмарної БД;
- алгоритмічне ядро: реалізація модифікованого EDD та hill climbing в JavaScript/TypeScript;
- інтеграції: взаємодія з календарями (Google Calendar, iCal/ICS) через існуючі API;
- розгортання: хостинг як SPA / PWA на стандартній веб-платформі.

Жодних унікальних чи важкодоступних технологій для реалізації проєкту не потрібно, що знижує технологічні ризики та витрати на старті.

Таблиця 4.3 – Технологічна здійсненність ідеї проєкту

№	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Веб-застосунок для персоналізованого планування навчального навантаження з алгоритмічним ядром (модифікований EDD + локальна оптимізація)	React, TypeScript, HTML/CSS; реалізація алгоритмів планування у вигляді окремих модулів; локальне сховище (LocalStorage/IndexedDB)	Технології є стандартом де-факто у веб-розробці	Відкриті бібліотеки, велика спільнота, безкоштовні інструменти розробки
2	Інтеграція з календарями та потенційно з LMS	REST/JSON API Google Calendar, універсальний формат iCal/ICS, вебхуки; API LMS (Moodle тощо)	API широко використовуються та добре документовані	Доступні через публічні SDK; використання спрощується готовими клієнтськими бібліотеками
3	Адаптивне перепланування та аналітика якості розкладу	Власні модулі оцінки якості (метрики нерівномірності, завантаження по днях тощо), можливе подальше використання бібліотек для ML/статистики	Реалізується на рівні бізнес-логіки застосунку	Не потребує спеціального обладнання, достатньо типового серверного/клієнтського середовища

Кінець таблиці 4.3

№	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
4	Розгортання MVP та подальше масштабування	Хостинг SPA (Vercel, Netlify, власний сервер), контейнеризація (Docker), CI/CD	Технології широко використовуються в галузі	Мають безкоштовні/умовно-безкоштовні тарифи на старті, низький поріг входу

Висновок: технологічна реалізація продукту є повністю можливою, ризики, пов’язані з технологічним стеком, мінімальні; обрана технологія – веб-застосунок на основі React + TypeScript з алгоритмічним ядром на стороні клієнта/серверу – є доцільною.

4.3. Аналіз ринкових можливостей запуску стартап-проєкту

Попередня характеристика потенційного ринку

Таблиця 4.4 – Попередня характеристика потенційного ринку

№	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	На ринку інструментів планування часу й завдань домінують 5–7 глобальних гравців (Google, Microsoft, Notion, Todoist, Trello тощо) та велика кількість менших сервісів, включно з нішевими study-planner’ами.
2	Загальний обсяг продаж, грн/ум. од.	Ринок цифрових інструментів продуктивності та EdTech оцінюється в значні обсяги й демонструє тенденцію до зростання; сегмент персонального планування навчання становить порівняно невелику, але зростаючу частку цього ринку.

Кінець таблиці 4.4

№	Показники стану ринку	Характеристика
3	Динаміка ринку	Ринок зростає за рахунок цифровізації освіти, поширення онлайн-курсів, гнучких форм навчання та поєднання навчання з роботою; попит на інструменти самоорганізації й самоменеджменту стабільно збільшується.
4	Наявність обмежень для входу	Формальних регуляторних бар'єрів небагато; основні обмеження пов'язані з конкуренцією, необхідністю створення привабливої ціннісної пропозиції та дотриманням вимог щодо захисту персональних даних.
5	Специфічні вимоги до стандартизації та сертифікації	Для B2C-сегменту спеціальної сертифікації, як правило, не потрібно; при роботі з університетами можуть висуватися вимоги до безпеки, сумісності з LMS, відповідності політикам конфіденційності.
6	Середня норма рентабельності в галузі, %	Для SaaS-продуктів у сфері продуктивності та EdTech характерна середня або вище середньої рентабельність за умови досягнення достатньої кількості активних користувачів та оптимізації витрат на інфраструктуру й маркетинг.

Висновок: ринок є конкурентним, але має стійку позитивну динаміку, низькі формальні бар'єри для входу та помітний тренд на персоналізацію навчання, що робить його привабливим для запуску нішевого продукту з чітко окресленою ціннісною пропозицією.

Характеристика потенційних клієнтів

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проєкту

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці цільових груп клієнтів	Вимоги споживачів до товару
1	Необхідність системно планувати навчальні завдання, уникати перевантаження й зривів дедлайнів	Студенти бакалаврату та магістратури денної форми навчання	Активно користуються смартфонами та веб-сервісами; часто мають нерівномірні періоди активності (сесія, дедлайни по курсових).	Простий та інтуїтивний інтерфейс; швидке додавання завдань; зрозумілий щоденний/тижневий план; підтримка української та, бажано, англійської мови.
2	Балансування навчання і роботи / інших зобов’язань при обмеженому часі	Студенти, які поєднують навчання з роботою; студенти-заочники, учасники онлайн-курсів	Планують час жорсткіше, мають фіксовані «вікна» для навчання; орієнтовані на практичний результат.	Гнучке налаштування доступного часу; адаптивне перепланування при зміні графіка; можливість бачити навантаження по тижнях/місяцях.
3	Підвищення успішності та керованості навчального процесу	Університети, факультети, освітні платформи (B2B-сегмент)	Орієнтуються на інтеграцію з існуючими LMS, аналітичні звіти, а не на індивідуальну роботу з планувальником.	Інтеграція з LMS; агреговані звіти по студентських навантаженнях; відповідність політикам безпеки; можливість брендованого інтерфейсу.

Фактори загроз та можливостей

Таблиця 4.6 – Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренти	Наявність глобальних продуктів (календарі, таск-менеджери, study-planner’и), які вже мають широку аудиторію	Диференціація за рахунок алгоритмічного планування, прозорих рекомендацій та фокусу на навчанні; безкоштовний / freemium-доступ на старті; активна комунікація з цільовою аудиторією.
2	Обмежені фінансові ресурси на розробку й маркетинг	Недостатнє фінансування може уповільнити розвиток продукту та його просування	Ітеративний розвиток MVP з пріоритетом на ключовий функціонал; використання low-cost каналів просування (соціальні мережі, співпраця з університетами, участь у хакатонах); пошук грантів / партнерств.
3	Зміна освітніх форматів або політик щодо використання сторонніх сервісів	Обмеження на використання зовнішніх інструментів у деяких університетах, вимоги до захисту даних	Наголос на безпеці та приватності (мінімальний збір даних, можливість локальної обробки); пропозиція self-hosted-версії для інституцій; приведення політик конфіденційності у відповідність до вимог ЗВО.
4	Недовіра користувачів до автоматичних рекомендацій	Частина користувачів може не довіряти алгоритмічному розкладу, воліючи ручне планування	Надання пояснень до розкладу («чому саме так»); можливість ручного коригування; ситуаційні підказки, що демонструють переваги використання алгоритму.

Таблиця 4.7 – Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Зростання ринку EdTech та онлайн-освіти	Збільшення кількості студентів, що навчаються дистанційно або у змішаному форматі, і потребують інструментів самоменеджменту	Орієнтація на інтеграцію з онлайн-платформами; створення спеціальних сценаріїв для курсів, Bootcamp-ів, інтенсивів.
2	Популярність персоналізованих рішень	Користувачі очікують індивідуальних рекомендацій та адаптивних систем	Створення механізмів персоналізації (налаштування пріоритетів, стилю роботи); подальше використання даних для адаптації рекомендацій.
3	Партнерства з університетами та студентськими організаціями	Можливість отримати канали доступу до цільової аудиторії, зворотний зв’язок та пілотні групи	Запуск пілотних проєктів у ЗВО; спільні ініціативи зі студентськими радами; проведення опитувань та досліджень.
4	Безкоштовні або умовно-безкоштовні інструменти розповсюдження	Соціальні мережі, сторінки факультетів, Telegram-канали, спільноти студентів	Активна присутність у соцмережах; створення корисного контенту (гайди з планування, матеріали про тайм-менеджмент); реферальні програми.

Аналіз конкурентного середовища

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку

№	Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1	Тип конкуренції: монополістична конкуренція	Існує низка схожих за призначенням продуктів, але кожен має власний набір функцій, UX та цінову модель	Необхідно відрізнятися унікальною ціннісною пропозицією (алгоритмічне планування саме для навчання) та якістю UX; важливо чітко формулювати позиціонування.
2	Рівень конкурентної боротьби: глобальний	Основні конкуренти мають міжнародний ринок та мультимовну підтримку	Орієнтація на локальний ринок (українська, можливо двомовний продукт) на старті з подальшою можливістю інтернаціоналізації; фокус на потребах локальних студентів.
3	Галузева ознака: EdTech / продуктивність	Продукти використовуються у сфері освіти та особистої ефективності	Необхідно враховувати освітні цикли (семестри, сесії), специфіку навчальних навантажень; пропонувати функції, релевантні саме до навчальних задач.
4	Конкуренція за видами товарів: між спеціалізованими study-planner’ами та універсальними таск-менеджерами	Студенти можуть обирати між простими todo-додатками й спеціалізованими інструментами	Пропонувати достатню простоту та водночас додану цінність (автоматичний розклад, аналітика), щоб виправдати перехід із загальних інструментів на спеціалізований.

Кінець таблиці 4.8

№	Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
5	Характер конкурентних переваг: цінові та нецінові	Багато базових продуктів безкоштовні або freemium	Використовувати freemium-модель: безкоштовний базовий функціонал + платні розширення (аналітика, інтеграції); робити акцент на нецінових перевагах (зручність, алгоритми, локалізація).
6	Інтенсивність конкуренції за бренд	Відомі бренди (Google, Microsoft, Notion) мають високу впізнаваність	Створювати впізнавану айдентику (назва, візуальний стиль), підкреслювати спеціалізацію на навчанні, використовувати історії користувачів (case studies).

Таблиця 4.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Опис	Існуючі study-planner’и, універсальні таск-менеджери та календарі	Нові стартапи у сфері EdTech, великі платформи, що можуть розширити функціонал планування	Провайдери хмарних сервісів, API календарів, платформи розповсюдження (app stores)	Студенти, ЗВО, онлайн-платформи	Паперові планери, звичайні списки справ, неструктуровані записи в месенджерах

Висновки: загрози з боку конкурентів та товарів-замінників є суттєвими, однак вузька спеціалізація на навчальному навантаженні та алгоритмічне планування дозволяють створити диференційований продукт.

Фактори конкурентоспроможності та їх оцінка

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування
1	Якість алгоритмічного планування та перепланування	Ключова відмінність від звичайних списків завдань і календарів; саме цей фактор дає вимірюваний ефект – зменшення піків навантаження, зривів дедлайнів.
2	Орієнтація саме на навчальні завдання	Більшість конкурентів загального призначення; фокус на навчанні дозволяє краще адаптувати інтерфейс, термінологію та сценарії використання.
3	Зручність та інтуїтивність інтерфейсу	Студенти використовують планувальник щодня; складний або перевантажений інтерфейс швидко призведе до відмови від продукту.
4	Інтеграції з календарями та LMS	Забезпечують безшовну роботу з існуючою інфраструктурою користувачів і ЗВО.
5	Пояснюваність рекомендацій	Можливість показати «чому так сплановано день» підвищує довіру до системи та сприяє навчанню тайм-менеджменту.
6	Локалізація та адаптація під український контекст	Підтримка української мови, локальних освітніх особливостей, календаря сесій тощо.
7	Модель монетизації (freemium)	Дозволяє залучити широку базу користувачів, не створюючи бар’єр входу, і поступово монетизувати частину з них.

Таблиця 4.11 – Порівняльний аналіз сильних та слабких сторін системи планування навчального навантаження

№	Фактор конкурентоспроможності	Бали 1–20 (важливість)	Мій продукт	Google Calendar + Tasks	Trello / Notion	Типові study-planner’и
1	Якість алгоритмічного планування	20	0 (базовий рівень)	-2	-2	-1
2	Перепланування при змінах	18	0	-2	-2	-1
3	Орієнтація на навчання	17	0	-3	-2	+1
4	Зручність інтерфейсу для студента	16	0	+1	0	0
5	Інтеграції з календарями/LMS	15	0	+2	+1	-1
6	Пояснюваність рекомендацій	14	0	-2	-2	-1
7	Локалізація, українська мова	14	0	-1	-1	0
8	Вартість для кінцевого користувача	13	0	0	0	0

Примітка: «0» для мого продукту означає базову точку відліку; від’ємні значення вказують на гірші позиції конкурентів, додатні – на кращі (там, де це доречно). Ти можеш деталізувати цю таблицю під конкретні обрані продукти-аналогі.

Таблиця 4.12 – SWOT-аналіз стартап-проєкту

Сильні сторони (S):	Слабкі сторони (W):
– спеціалізація на навчальному навантаженні; – алгоритмічне планування й перепланування; – прозорі метрики якості розкладу; – зручний веб-інтерфейс, орієнтований на студентів; – можливість інтеграції з календарями та LMS.	– відсутність відомого бренду; – обмежені ресурси на маркетинг та підтримку; – початково невелика база користувачів; – необхідність пояснювати користувачам цінність алгоритмічних рекомендацій.
Можливості (O):	Загрози (T):
– зростання ринку EdTech та онлайн-освіти; – попит на персоналізовані інструменти навчання; – партнерства із ЗВО та студентськими організаціями; – використання продукту як навчального кейсу в IT-дисциплінах.	– конкуренція з глобальними гравцями; – поява нових study-planner’ів з подібним функціоналом; – зміни у політиках університетів щодо сторонніх сервісів; – ризики, пов’язані з безпекою та приватністю даних.

Таблиця 4.13 – Альтернативи ринкового впровадження стартап-проєкту

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Запуск безкоштовної beta-версії для студентів із базовим функціоналом; збір зворотного зв’язку та метрик використання	Висока, ресурси на MVP вже частково наявні	1–2 місяці
2	Контент-маркетинг: публікація статей, гайдів з тайм-менеджменту для студентів на профільних ресурсах та в соцмережах	Потрібні переважно час та мінімальний бюджет	1–3 місяці (безперервно)

Кінець таблиці 4.13

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
3	Партнерства з факультетами й студентськими організаціями, пілотні впровадження в окремих групах	Потребують додаткових зусиль на комунікацію та підтримку	3–6 місяців
4	Участь у хакатонах, EdTech-конкурсах, акселераторах	Потрібні час і мінімальні кошти на участь / підготовку	2–4 місяці

4.4. Розроблення ринкової стратегії проєкту

Вибір цільових груп та базових стратегій

Таблиця 4.14 – Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит у межах сегменту	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Студенти бакалаврату та магістратури денної форми, що мають високий рівень завантаженості (технічні спеціальності тощо)	Висока: активно шукають способи організації навчання, користуються додатками	Високий потенціал, особливо у періоди сесій	Середня: багато загальних інструментів, мало вузькоспеціалізованих	Відносно проста: достатньо онлайн-просування

Кінець таблиці 4.14

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит у межах сегменту	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
2	Студенти, які поєднують навчання з роботою, студенти-заочники, слухачі онлайн-курсів	Висока, за умови чіткої демонстрації користі	Середній –високий	Невисока: мало рішень, орієнтованих саме на цю групу	Потребує більш таргетованих каналів (спільноти, партнерства)
3	Університети, факультети, EdTech-платформи	Середня: потрібні докази ефективності та стабільності продукту	Середній, але з високою вартістю контракту	Вища: конкуренція з LMS і комплексними EdTech-рішеннями	Вхід складніший: потрібні переговори, інтеграції, пілотні проєкти

Обрані цільові групи: на початковому етапі доцільно сфокусуватися на індивідуальних студентах (групи 1–2), використовуючи B2C-фокус із freemium-моделлю. B2B-сегмент (університети, платформи) розглядається як наступний етап розвитку.

Таблиця 4.15 – Визначення базової стратегії розвитку

Обрана альтернатива розвитку проєкту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Розвиток нішевого EdTech-продукту для студентів з поступовим масштабуванням на B2B-сегмент	Стратегія концентрованого (нішевого) маркетингу на старті; можливий перехід до ширшого охоплення	Алгоритмічне планування та перепланування; спеціалізація на навчальному навантаженні; локалізація під український контекст; інтеграції з календарями	Стратегія диференціації в рамках нішевого ринку (виділення за рахунок унікального функціоналу та UX)

Таблиця 4.16 – Визначення базової стратегії конкурентної поведінки

Питання	Відповідь	Стратегія конкурентної поведінки
Чи є проєкт «першопрохідцем» на ринку?	Ні, однак у вибраній ніші (алгоритмічне планування навчального навантаження з перепланування) конкуренція невисока	
Чи буде компанія шукати нових споживачів або забирати існуючих у конкурентів?	Обидва варіанти: залучення нових користувачів, які раніше не користувалися планувальниками, та перетягування частини користувачів від універсальних таск-менеджерів	
Чи буде компанія копіювати основні характеристики товару конкурента?	Базовий функціонал (списки завдань, дедлайни, нагадування) має бути не гіршим за конкурентів, але акцент робиться на унікальних можливостях алгоритмічного планування	Стратегія заняття конкурентної ніші з елементами фокусованої диференціації

Таблиця 4.17 – Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проєкту	Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту
1	Простота використання, швидке додавання завдань, наочний календар	Диференціація в межах ніші	Легкий старт, інтуїтивний інтерфейс	«Розумний, але простий планувальник для студента»
2	Реальне зменшення стресу та ризику зриву дедлайнів	Диференціація через алгоритмічні переваги	Алгоритмічне планування, перепланування, метрики навантаження	«Помічник, який не дасть завалити сесію»
3	Безкоштовний вхід і можливість користуватися базовою версією без оплати	Freemium-модель	Безкоштовний базовий функціонал; справедлива ціна за «профі»-можливості	«Спробуй безкоштовно – користуйся щодня, плати лише за advanced-функції»

4.5. Розроблення маркетингової програми стартап-проєкту

Концепція товару, ціноутворення, збуту та комунікацій

Таблиця 4.18 – Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
1	Планування великої кількості навчальних завдань з різними дедлайнами	Автоматичний збалансований розклад з урахуванням складності завдань, дедлайнів та доступного часу	Формалізований алгоритм (модифікований EDD + hill climbing), що зменшує піки навантаження

Кінець таблиці 4.18

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
2	Адаптація до змін у графіку (робота, хвороба, перенесення занять)	Швидке перепланування з урахуванням нових обмежень	Вбудовані процедури локальної оптимізації, що дозволяють «перебудувати» розклад без ручної перетасовки
3	Потреба розуміти, чому план виглядає саме так	Пояснюваність: короткі пояснення до рекомендацій («чому це завдання сьогодні»)	Більшість конкурентів не дають пояснень, лише список завдань; ми допомагаємо ще й навчатись тайм-менеджменту
4	Зручна щоденна робота з інструментом	Мінімальний час взаємодії з інтерфейсом, зрозумілі екрани «Сьогодні», «Тиждень», «Розклад»	UX, спроєктований під конкретні сценарії студентів; локалізація, адаптація до навчального календаря

Таблиця 4.19 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
1. Товар за задумом	Інструмент для керованого виконання навчальних завдань із мінімізацією стресу та зривів дедлайнів за рахунок алгоритмічного планування й перепланування.
2. Товар у реальному виконанні	Веб-застосунок «особистий планувальник навчального навантаження студента», що включає: екрани «Мої завдання», «Розклад», «Календар», «Оптимізація»; модулі введення завдань (предмет, дедлайн, трудомісткість, тип роботи); модулі побудови розкладу (модифікований EDD, hill climbing); відображення плану по днях/тижнях; базова аналітика (завантаження по днях, кількість завдань в роботі); інтеграцію з календарями (імпорт/експорт подій); підтримку української мови.

Кінець таблиці 4.19

Рівні товару	Сутність та складові
3. Товар із підкріпленням	До продажу / використання: простий onboarding (короткий туторіал, приклад семестрового розкладу); безкоштовний базовий тариф; публічний roadmap і канал зворотного зв'язку. Після продажу / у платній версії: пріоритетна підтримка; розширена аналітика; можливість експорту звітів для куратора/менторів; Захист від копіювання: збереження прав на алгоритмічні рішення й інтерфейс; у разі масштабування – реєстрація торговельної марки.

Таблиця 4.20 – Визначення меж встановлення ціни

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
Більшість універсальних планерів часу (Google Calendar, базові todo-додатки) – безкоштовні	Частина study-planner'ів та продуктивність-додатків працює за freemium-моделлю з ціною 3–10 USD/міс.	Студенти мають обмежений дохід; готовність платити за корисний інструмент є, але в невеликому розмірі	Нижня межа – безкоштовний базовий тариф; верхня – платний план 2–5 USD/міс. з розширеним функціоналом (аналітика, інтеграції, підтримка)

Таблиця 4.21 – Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Студенти самостійно обирають та встановлюють цифрові інструменти; рішення часто приймається під впливом рекомендацій друзів та відгуків у мережі	Надання простого механізму реєстрації/авторизації; підтримка web-версії та, за можливості, мобільної; забезпечення швидкого доступу до продукту без складних процедур придбання	Мінімальна: прямий збут через сайт / онлайн-платформи	Прямий онлайн-збут: веб-сайт продукту, сторінки у соцмережах, публікація в каталогах/маркетплейсах; для B2B – прямі продажі та пілотні впровадження у ЗВО

Таблиця 4.22 – Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуютьс я цільові клієнти	Ключові позиції, обрані для позиціонув ання	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Щоденно користуються смартфонами, соцмережами, месенджерами; активно реагують на рекомендації одногрупників	Instagram, Telegram-кан али факультетів, студентські спільноти, TikTok, Discord	«Розумний планувальн ик, який допомагає пережити сесію без завалів»	Пояснити, що продукт реально зменшує хаос у навчанні й не потребує багато часу для налаштування	Короткі відео/пости «до/після» (як виглядає тиждень студента без планувальника і з ним); історії реальних користувачів

Кінець таблиці 4.22

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
2	Орієнтовані на ефективність та баланс між навчанням і роботою	Професійні спільноти, LinkedIn, тематичні Telegram-канали, блоги про продуктивність	«Інструмент для поєднання навчання й роботи без вигорання»	Показати, що планувальник допомагає оптимізувати час і тримати під контролем дедлайни	Лендінг з конкретними сценаріями («приклад тижня студента, який працює 20 год/тиждень і навчається на повну»)
3	Представники ЗВО, які шукають інструменти підтримки студентів	Офіційні канали університетів, конференції, професійні спільноти викладачів	«Модуль індивідуального планування в екосистемі LMS університету»	Продемонструвати, що продукт може доповнити LMS, підвищити успішність та керованість навантаження	Презентаційні матеріали, пілотні кейси, демо-доступ для факультетів

4.6. Висновки до розділу 4

У даному розділі виконано маркетинговий аналіз стартап-проєкту – веб-застосунку для розподілу навчального навантаження студента. Сформульовано зміст ідеї, визначено її основні техніко-економічні характеристики та їх порівняння з існуючими аналогами. Проведено технологічний аудит, який показав, що реалізація продукту на основі

веб-технологій (React + TypeScript) та алгоритмічних модулів (модифікований EDD, hill climbing) є технічно здійсненною та не потребує рідкісних або дорогих технологій.

На основі аналізу ринку EdTech та інструментів продуктивності виявлено, що сегмент персональних планувальників навчального навантаження має позитивну динаміку розвитку й водночас залишається недостатньо заповненим продуктами, орієнтованими саме на навчальний контекст. Визначено цільові групи користувачів (студенти денної форми, студенти, що працюють, освітні інституції), для яких розроблено стратегії позиціонування, охоплення ринку та конкурентної поведінки. Проведено SWOT-аналіз, який підтвердив, що сильні сторони проєкту (алгоритмічне планування, спеціалізація на навчальному процесі, локалізація) дозволяють компенсувати слабкі сторони, пов’язані з браком ресурсу та недостатньою впізнаваністю бренду.

Сформовано маркетингову програму стартап-проєкту, що включає концепцію товару (три рівні продукту), підхід до ціноутворення (freemium-модель з платними розширеннями), систему збуту (прямий онлайн-збут із подальшим виходом у B2B-сегмент) та стратегію маркетингових комунікацій для ключових сегментів аудиторії. Отримані результати створюють передумови для подальшого практичного впровадження розробленого програмного забезпечення та його комерціалізації.

Висновки

У ході виконання магістерської дисертації на тему **«Метод та програмне забезпечення для автоматизованого розподілу навчального навантаження студента з урахуванням індивідуального графіку»** було розв’язано комплекс задач, пов’язаних з аналізом проблеми нерівномірного навчального навантаження, розробкою математичного методу його розподілу та створенням працюючого програмного засобу.

У першому розділі проаналізовано сучасний стан задачі планування індивідуальної навчальної діяльності студента, розглянуто існуючі інструменти (календарі, таск-трегери, LMS-системи) та класичні підходи до розкладання завдань. Показано, що наявні рішення здебільшого зосереджуються на фіксації дедлайнів і не враховують обмежену кількість доступних годин студента, що призводить до пікових перевантажень наприкінці періоду та підвищує ризик зриву дедлайнів і вигорання. На основі аналізу сформульовано вимоги до програмного засобу, орієнтованого саме на рівномірний розподіл навантаження з урахуванням індивідуального календаря студента.

У другому розділі формалізовано задачу розподілу навчального навантаження як задачу побудови індивідуального розкладу, в якому кожне завдання має тривалість, дедлайн та може бути розбите на дрібніші блоки, а кожен день календаря характеризується місткістю (доступною кількістю годин). Визначено набір метрик оцінювання якості розкладу: максимальне добове навантаження, кількість перевантажених днів, дисперсія добового навантаження, кількість днів із піковим навантаженням та кількість завдань із ризиком прострочки.

На основі аналізу підходів до розкладання обґрунтовано вибір комбінованого методу на базі модифікованого алгоритму Earliest Due Date (EDD) та локальної оптимізації типу hill climbing. Розроблено модифікований метод EDD, у якому введено поняття ефективного дедлайну (дедлайн мінус буфер), добові ліміти завантаження та механізм дроблення великих задач на блоки. Поверх

отриманого розкладу застосовано локальну оптимізацію, що за допомогою операцій перестановки та перенесення блоків мінімізує піки навантаження і згладжує розподіл годин у часі, не порушуючи дедлайнів та добових обмежень.

У третьому розділі розроблено архітектуру та реалізовано веб-застосунок-планувальник на основі стеку React + TypeScript з локальним зберіганням стану. Система включає модулі введення завдань і доступних годин, генерації розкладу модифікованим EDD, запуску локальної оптимізації, аналізу метрик і візуалізації результатів у вигляді календаря. Забезпечено повний цикл «планування – аналіз – адаптація»: користувач може змінювати набір завдань, доступність часу, відмічати виконані блоки й ініціювати повторну генерацію розкладу з урахуванням реального стану.

Проведені експериментальні дослідження на типових наборах завдань показали, що класичний EDD без ресурсних обмежень формує нереалістичний графік із піковим навантаженням до 12 годин на день та значною кількістю перевантажених днів. Впровадження добового ліміту й буфера до дедлайнів у модифікованому EDD усуває перевантажені дні та знижує дисперсію добового навантаження, але робить календар занадто щільним. Додавання локальної оптимізації дозволяє зберегти відсутність перевантажень і водночас суттєво згладити навантаження: кількість днів із піковим навантаженням зменшилась із 45 до 14, а дисперсія добового навантаження – з 3,27 до 0,44 год², при нульовій кількості завдань з ризиком прострочки. Це свідчить про досягнення поставленої мети – **покращення ефективності планування навчального процесу студента**.

Таким чином, у роботі проаналізовано проблему нерівномірного навчального навантаження, сформульовано й розв’язано задачу автоматизованого розподілу навчальних завдань з урахуванням індивідуального графіку, розроблено метод і програмний засіб, які забезпечують рівномірний і керований у часі план роботи студента.

Наукова новизна одержаних результатів магістерської дисертації:

- запропоновано метод розподілу навчального навантаження студента, який поєднує **модифікований алгоритм Earliest Due Date** з ефективним дедлайном (дедлайн – буфер), добовими лімітами навантаження та дробленням задач на блоки з урахуванням індивідуального календаря доступних годин;
- сформовано інтегровану систему метрик оцінювання якості індивідуального розкладу (максимальне добове навантаження, кількість перевантажених днів, кількість днів із піковим навантаженням, дисперсія добового навантаження, кількість ризикових завдань), орієнтовану саме на комфорт та стійкість навчального процесу студента.
- **набула подальшого розвитку оптимізація типу hill climbing** у задачах планування навчальних завдань шляхом розробки спеціалізованої цільової функції для згладжування навантаження та набору операцій сусідства, що забезпечують покращення розкладу без порушення дедлайнів і добових обмежень;

Практичне значення одержаних результатів

- Розроблено та реалізовано веб-застосунок – особистий планувальник навчального навантаження студента, який може використовуватись як реальний інструмент навчального самоменеджменту для побудови, аналізу та корекції індивідуального графіку роботи.
- Запропонований метод розподілу навантаження, набір метрик та архітектурні рішення можуть бути використані при створенні модулів індивідуальних планів у складі університетських LMS-систем та інших освітніх платформ (особливо для студентів із гнучким або заочним графіком навчання).

- Стек технологій (React + TypeScript, локальне збереження стану) та виділені алгоритмічні модулі можуть бути використані як основа для подальшого розгортання продукту в реальних умовах, а також як приклад

навчального проєкту для дисциплін, пов’язаних з розробкою веб-застосунків, оптимізаційних алгоритмів та систем підтримки прийняття рішень.

Після виконання поставлених задач і проведення експериментальної оцінки доведено, що мети магістерської роботи досягнуто.

Список використаних джерел

1. **Pinedo M. L.** Scheduling: Theory, Algorithms, and Systems : *монографія* / M. L. Pinedo. – 5th ed. – Springer, 2016. – 670 p.
2. **Russell S. J., Norvig P.** Artificial Intelligence: A Modern Approach : *підручник* / S. J. Russell, P. Norvig. – 4th ed. – Pearson, 2020. – 1136 p.
3. **Cormen T. H., Leiserson C. E., Rivest R. L., Stein C.** Introduction to Algorithms : *підручник* / T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. – 4th ed. – MIT Press, 2022. – 1312 p.
4. **React** [Електронний ресурс] // Фреймворк для розробки веб додатків. – Режим доступу: https://react.dev/.
5. **Brucker P.** Scheduling Algorithms : *monograph* / P. Brucker. – 5th ed. – Springer, 2007. – 371 p.
6. **Korte B., Vygen J.** Combinatorial Optimization: Theory and Algorithms : *monograph* / B. Korte, J. Vygen. – 6th ed. – Springer, 2018. – 698 p.
7. **Garey M. R., Johnson D. S.** Computers and Intractability: A Guide to the Theory of NP-Completeness : *monograph* / M. R. Garey, D. S. Johnson. – Freeman, 1979. – 338 p.
8. **Chen M. C., Sze S. N., Goh S. L., Sabar N. R., Kendall G.** A Survey of University Course Timetabling Problem: Perspectives, Trends and Opportunities // *IEEE Access*. – 2021. – Vol. 9. – P. 146032–146062.
9. **Siew E. S. K., Sze S. N., Goh S. L., Kendall G., Sabar N. R., Abdullah S.** A Survey of Solution Methodologies for Exam Timetabling Problems // *IEEE Access*. – 2024. – Vol. 12. – P. 1–20.
10. **Bykov Y., Petrovic S.** A Step Counting Hill Climbing Algorithm Applied to University Examination Timetabling // *Journal of Scheduling*. – 2016. – Vol. 19, No. 4. – P. 479–492.

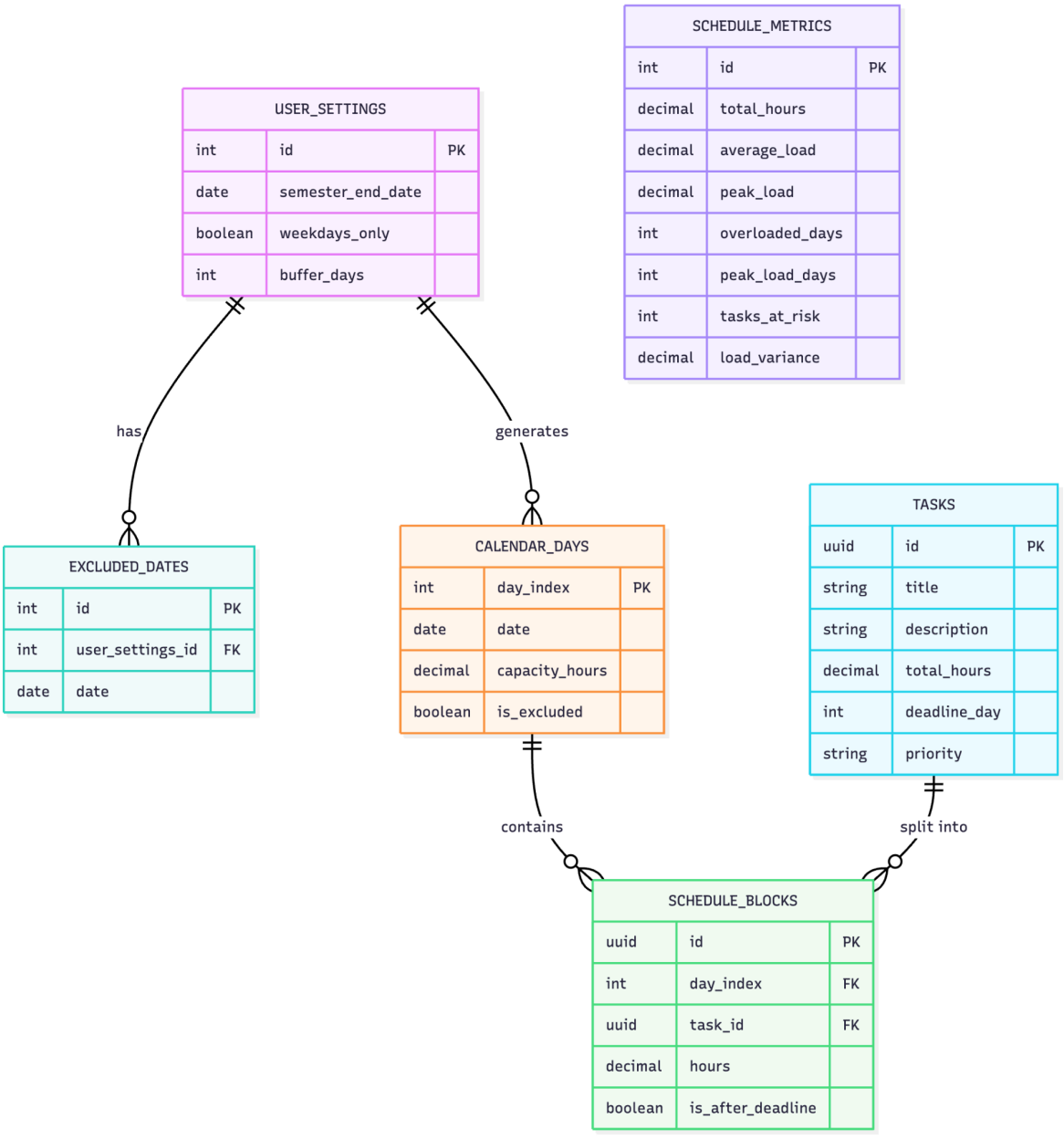
11. **Al-Betar M. A.** A β-hill climbing optimizer for examination timetabling problem // *Journal of Ambient Intelligence and Humanized Computing*. – 2021. – Vol. 12. – P. 653–666.
12. **Гайтан О. М.** Автоматизація генерації розкладу навчального процесу університету // *Вчені записки ТНУ ім. В. І. Вернадського. Серія: Технічні науки*. – 2020. – Т. 31(70), № 2, ч. 1. – С. 58–66.
13. **Логінова Н. І., Янковський О. Г., Лобода Ю. Г., Толокнов А. А.** Програмна реалізація задачі автоматизованого формування розподілу навчального навантаження науково-педагогічних працівників кафедри // *Вчені записки ТНУ ім. В. І. Вернадського. Серія: Технічні науки*. – 2021. – Т. 32(71), № 1, ч. 1. – С. 110–117.
14. **Lingshetti G., Pawar S., Shinde M., Sayyed H.** AI-Powered Smart Study Planner: Enhancing Personalized Learning Through Intelligent Scheduling // *International Journal of Scientific Research in Science, Engineering and Technology*. – 2025. – Vol. 12, No. 3. – P. 148–155.
15. **Haritha M., Mercy Angel A., Renuga S., Rithika G.** Smart Study Scheduler: An AI-Based Time and Syllabus Management Tool // *International Journal of Research Publication and Reviews*. – 2025. – Vol. 6, Issue 10. – P. 6414–6418.
16. **Varma P. A., Sujatha S., Krishna U. R., Harini M., Vardhan S. H.** A Novel Smart Study Plan for the Students // *International Journal of Research Publication and Reviews*. – 2025. – Vol. 6, Issue 4. – P. 1125–1133.
17. **Vorobyeva K. I., Belous S., Savchenko N. V., Smirnova L. M., Nikitina S. A., Zhdanov S. P.** Personalized learning through AI: Pedagogical approaches and critical insights // *Contemporary Educational Technology*. – 2025. – Vol. 17(2). – Article ep574 (23 p.).
18. **Moodle** [Електронний ресурс] // Офіційний сайт системи управління навчанням Moodle. – Режим доступу: https://moodle.org/.

19. **Google Calendar** [Електронний ресурс] // Сервіс календарного планування від Google. – Режим доступу: https://workspace.google.com/products/calendar/.
20. **Trello** [Електронний ресурс] // Веб-додаток для управління завданнями на дошках Kanban. – Режим доступу: https://trello.com/.
21. **Asana** [Електронний ресурс] // Онлайн-система управління проєктами та завданнями. – Режим доступу: https://asana.com/.

Додатки

ДОДАТОК А

Схема бази даних



ДОДАТОК Б

Лістинг коду

AvailabilityCalendar.tsx

```
import React, { useEffect, useState } from 'react';
import { Card, InputNumber, Button, Space, Table, Alert } from 'antd';
import type { ColumnsType } from 'antd/es/table';
import { useAppStore } from '../../store/useAppStore';
import type { CalendarDay } from '../../types';
import dayjs from 'dayjs';

export function AvailabilityCalendar() {
 const {
  calendar,
  numberOfDays,
  userSettings,
  setCalendarDay,
  setAllDaysCapacity,
  initializeCalendar
 } = useAppStore();

 const [desiredHoursPerDay, setDesiredHoursPerDay] = useState<number>(4);

 useEffect(() => {
  if (calendar.length === 0) {
   initializeCalendar();
  }
 }, []);

 const handleCapacityChange = (dayIndex: number, value: number | null) => {
  if (value !== null && value >= 0) {
   setCalendarDay(dayIndex, value);
  }
 };

 const handleSetAllCapacity = () => {
  if (desiredHoursPerDay > 0) {
   setAllDaysCapacity(desiredHoursPerDay);
  }
```

```
};

const handleEqualizeDays = () => {
  const avgCapacity = Math.round(
    calendar.reduce((sum, day) => sum + day.capacityHours, 0) / calendar.length
  );
  const rounded = avgCapacity || 4;
  setDesiredHoursPerDay(rounded);
  setAllDaysCapacity(rounded);
};

const getDayName = (dateStr?: string) => {
  if (!dateStr) return '';
  const date = dayjs(dateStr);
  const days = ['Неділя', 'Понеділок', 'Вівторок', 'Середа', 'Четвер', 'П\'ятниця', 'Субота'];
  return days[date.day()];
};

const columns: ColumnsType<CalendarDay> = [
  {
    title: 'День',
    dataIndex: 'dayIndex',
    key: 'dayIndex',
    width: '15%',
    render: (day: number) => (
      <span className="font-semibold">День {day}</span>
    )
  },
  {
    title: 'Дата',
    dataIndex: 'date',
    key: 'date',
    width: '25%',
    render: (date: string) => (
      <div>
        <div>{dayjs(date).format('DD.MM.YYYY')}</div>
        <div className="text-xs text-gray-500">{getDayName(date)}</div>
      </div>
    )
  },
  {
```

```
    title: 'Доступно годин',
    dataIndex: 'capacityHours',
    key: 'capacityHours',
    width: '30%',
    render: (hours: number, record: CalendarDay) => (
      <InputNumber
        min={0}
        max={24}
        step={0.5}
        value={hours}
        onChange={(value) => handleCapacityChange(record.dayIndex, value)}
        addonAfter="год"
        className="w-full"
      />
    )
  },
  {
    title: 'Статус',
    key: 'status',
    width: '30%',
    render: (_, record: CalendarDay) => {
      if (record.capacityHours === 0) {
        return <span className="text-red-500">Недоступний</span>;
      } else if (record.capacityHours < 2) {
        return <span className="text-orange-500">Обмежений</span>;
      } else if (record.capacityHours >= 6) {
        return <span className="text-green-500">Високий</span>;
      } else {
        return <span className="text-blue-500">Нормальний</span>;
      }
    }
  }
];

const totalCapacity = calendar.reduce((sum, day) => sum + day.capacityHours, 0);
const avgCapacity = calendar.length > 0 ? (totalCapacity / calendar.length).toFixed(1) : 0;

return (
  <div className="space-y-3">
    <div className="flex justify-between items-center">
      <h2 className="text-2xl font-bold">Календар доступності</h2>
```

```
    </div>

    <Card className="shadow-sm" bodyStyle={{ padding: '16px' }}>
      <div className="space-y-3">
        {userSettings.semesterEndDate ? (
          <div className="bg-green-50 border border-green-200 p-4 rounded-lg">
            <div className="flex items-center gap-2 mb-2">
              <span className="text-green-600 text-lg">✓</span>
              <span className="font-medium text-green-800">
                Період планування налаштовано
              </span>
            </div>
            <div className="text-sm text-green-700">
              До кінця семестру ({dayjs(userSettings.semesterEndDate).format('DD.MM.YYYY')}):
              <strong className="ml-1">{numberOfDays} робочих днів</strong>
            </div>
            {userSettings.weekdaysOnly && (
              <div className="text-sm text-green-600 mt-1">
                • Виключено вихідні дні (субота, неділя)
              </div>
            )}
            {userSettings.excludedDates.length > 0 && (
              <div className="text-sm text-green-600 mt-1">
                • Додатково виключено {userSettings.excludedDates.length} {userSettings.excludedDates.length === 1 ?
'день' : userSettings.excludedDates.length < 5 ? 'дні' : 'днів'}
              </div>
            )}
          </div>
        ) : (
          <Alert
            message="Налаштуйте період планування"
            description={
              <span>
                Перейдіть в розділ <strong>"Налаштування"</strong> щоб вказати дату кінця семестру
                та інші параметри планування
              </span>
            }
            type="warning"
            showIcon
          />
        )}
```

```
<div className="space-y-3">
  <div>
    <label className="block text-sm font-medium mb-2 text-gray-700">
      Бажана кількість годин на день
    </label>
    <div className="flex gap-3 items-center">
      <InputNumber
        min={0.5}
        max={24}
        step={0.5}
        value={desiredHoursPerDay}
        onChange={(value) => setDesiredHoursPerDay(value || 4)}
        addonAfter="год/день"
        className="w-48"
      />
      <Button type="primary" onClick={handleSetAllCapacity}>
        Встановити для всіх днів
      </Button>
      <Button type="default" onClick={handleEqualizeDays}>
        Вирівняти поточні значення
      </Button>
    </div>
    <p className="text-xs text-gray-500 mt-1">
      Встановіть однакову кількість годин для всіх днів календаря
    </p>
  </div>
</div>

<div className="bg-blue-50 p-4 rounded-lg">
  <div className="grid grid-cols-2 gap-4">
    <div>
      <div className="text-sm text-gray-600">Всього годин</div>
      <div className="text-2xl font-bold text-blue-600">{totalCapacity} год</div>
    </div>
    <div>
      <div className="text-sm text-gray-600">Середньо на день</div>
      <div className="text-2xl font-bold text-blue-600">{avgCapacity} год</div>
    </div>
  </div>
</div>
```

```
      </div>
    </Card>

    <Card className="shadow-sm" bodyStyle={{ padding: '12px' }}>
      <Table
        columns={columns}
        dataSource={calendar}
        rowKey="dayIndex"
        size="small"
        pagination={false}
        locale={{
          emptyText: 'Налаштуйте період планування в розділі Налаштування'
        }}
      />
    </Card>

    <div className="bg-yellow-50 border border-yellow-200 rounded-lg p-4">
      <p className="text-sm text-gray-700">
        💡 <strong>Порада:</strong> Вкажіть реальну кількість годин, яку ви можете виділити на навчання кожного дня.
        Враховуйте лекції, особисті справи та відпочинок.
      </p>
    </div>
  </div>
 );
}
```

ScheduleView.tsx

```
import React from 'react';
import { Button, Card, Empty, Spin, Alert, Tag, Progress } from 'antd';
import {
 CalendarOutlined,
 WarningOutlined,
 CheckCircleOutlined,
 ClockCircleOutlined
} from '@ant-design/icons';
import { useAppStore } from '../../store/useAppStore';
import { groupBlocksByDay, calculateScheduleMetrics, findTasksAtRisk } from '../../algorithms/utils';
```

```
import dayjs from 'dayjs';

export function ScheduleView() {
 const {
  tasks,
  calendar,
  schedule,
  isGenerating,
  generateSchedule,
  error,
  clearError
 } = useAppStore();

 const handleGenerate = async () => {
  clearError();
  await generateSchedule();
 };

 if (error) {
  return (
   <Alert
    message="Помилка"
    description={error}
    type="error"
    closable
    onClose={clearError}
   />
  );
 }

 if (isGenerating) {
  return (
   <div className="flex flex-col items-center justify-center py-20">
    <Spin size="large" />
    <p className="mt-4 text-gray-600">Генерується розклад...</p>
   </div>
  );
 }

 if (!schedule) {
  return (
```

```
    <div className="space-y-4">
      <div className="flex justify-between items-center">
        <h2 className="text-2xl font-bold">Розклад</h2>
      </div>

      <Card className="shadow-sm">
        <Empty
          description="Розклад ще не згенеровано"
          image={Empty.PRESENTED_IMAGE_SIMPLE}
        >
          <Button
            type="primary"
            size="large"
            icon={<CalendarOutlined />}
            onClick={handleGenerate}
            disabled={tasks.length === 0 || calendar.length === 0}
          >
            Згенерувати розклад
          </Button>
          {tasks.length === 0 && (
            <p className="mt-2 text-gray-500">Спочатку додайте завдання</p>
          )}
        </Empty>
      </Card>
    </div>
  );
}

const daySchedules = groupBlocksByDay(schedule.blocks, calendar);
const metrics = calculateScheduleMetrics(schedule.blocks, tasks, calendar);
const tasksAtRisk = findTasksAtRisk(tasks, schedule.blocks);

const getTaskById = (id: string) => tasks.find(t => t.id === id);

return (
  <div className="space-y-3">
    <div className="flex justify-between items-center">
      <h2 className="text-2xl font-bold">Розклад</h2>
      <Button onClick={handleGenerate} size="middle">
        Перегенерувати
      </Button>
```

```
    </div>

    {/* Метрики */}
    <div className="grid grid-cols-2 md:grid-cols-3 lg:grid-cols-5 gap-3">
      <Card className="shadow-sm" bodyStyle={{ padding: '12px' }}>
        <div className="text-xs text-gray-600">Всього годин</div>
        <div className="text-xl font-bold">{metrics.totalHours.toFixed(1)}</div>
      </Card>
      <Card className="shadow-sm" bodyStyle={{ padding: '12px' }}>
        <div className="text-xs text-gray-600">Середнє навантаження</div>
        <div className="text-xl font-bold">{metrics.averageLoad.toFixed(1)} год/день</div>
      </Card>
      <Card className="shadow-sm" bodyStyle={{ padding: '12px' }}>
        <div className="text-xs text-gray-600">Перевантажені дні</div>
        <div className={`text-xl font-bold ${metrics.overloadedDays > 0 ? 'text-red-500' : 'text-green-500'}`}>
          {metrics.overloadedDays}
        </div>
      </Card>
      <Card className="shadow-sm" bodyStyle={{ padding: '12px' }}>
        <div className="text-xs text-gray-600">Дні з піковим навантаженням</div>
        <div className={`text-xl font-bold ${metrics.peakLoadDays > 0 ? 'text-blue-500' : 'text-gray-500'}`}>
          {metrics.peakLoadDays}
        </div>
      </Card>
      <Card className="shadow-sm" bodyStyle={{ padding: '12px' }}>
        <div className="text-xs text-gray-600">Завдання з ризиком</div>
        <div className={`text-xl font-bold ${metrics.tasksAtRisk > 0 ? 'text-orange-500' : 'text-green-500'}`}>
          {metrics.tasksAtRisk}
        </div>
      </Card>
    </div>

    {/* Попередження */}
    {(metrics.overloadedDays > 0 || metrics.tasksAtRisk > 0) && (
      <Alert
        message="Увага! Потрібні корективи"
        description={
          <div>
            {metrics.overloadedDays > 0 && (
              <div>• Є {metrics.overloadedDays} перевантажених днів - спробуйте збільшити доступність в
календарі</div>
```

```
        )}
        {metrics.tasksAtRisk > 0 && (
          <div>• {metrics.tasksAtRisk} завдань можуть не встигнути до дедлайну - дивіться деталі нижче</div>
        )}
        <div className="mt-2">Спробуйте оптимізувати розклад або змінити параметри</div>
      </div>
    }
    type="warning"
    showIcon
  />
)}

{/* Успішне повідомлення */}
{schedule.isOptimized && (
  <Alert
    message="Розклад оптимізовано"
    type="success"
    showIcon
    icon={<CheckCircleOutlined />}
  />
)}

{/* Список завдань з ризиком */}
{tasksAtRisk.length > 0 && (
  <Card
    className="shadow-sm border-2 border-orange-300"
    bodyStyle={{ padding: '16px' }}
    title={
      <div className="flex items-center gap-2">
        <WarningOutlined className="text-orange-500" />
        <span>Завдання з ризиком запізнення ({tasksAtRisk.length})</span>
      </div>
    }
  >
    <div className="space-y-2">
      <p className="text-sm text-gray-600 mb-3">
        Ці завдання можуть не встигнути до дедлайну через недостатню кількість вільного часу:
      </p>
      {tasksAtRisk.map((taskId) => {
        const task = getTaskById(taskId);
        if (!task) return null;
```

```
// Знайти всі блоки цього завдання
const taskBlocks = schedule.blocks.filter(b => b.taskId === taskId);
const scheduledHours = taskBlocks.reduce((sum, b) => sum + b.hours, 0);
const remainingHours = task.totalHours - scheduledHours;
const maxDay = taskBlocks.length > 0
 ? Math.max(...taskBlocks.map(b => b.dayIndex))
 : 0;
const afterDeadline = maxDay > task.deadlineDay;

return (
 <div
  key={taskId}
  className="p-3 rounded-lg border border-orange-200 bg-orange-50"
 >
  <div className="flex justify-between items-start">
   <div className="flex-1">
    <div className="font-medium text-gray-900">{task.title}</div>
    <div className="text-sm text-gray-600 mt-1 space-y-1">
     <div className="flex items-center gap-2">
      <ClockCircleOutlined />
      <span>Дедлайн: День {task.deadlineDay}</span>
     </div>
     {afterDeadline && maxDay > 0 && (
      <div className="text-red-600 font-medium">
       ⚠️ Виконується після дедлайну (День {maxDay})
      </div>
     )}
     {remainingHours > 0 && (
      <div className="text-orange-600">
       Не вистачає {remainingHours.toFixed(1)} год для завершення
      </div>
     )}
     {scheduledHours === 0 && (
      <div className="text-red-600 font-medium">
       Завдання взагалі не заплановане
      </div>
     )}
    </div>
   </div>
   <div className="text-right ml-4">
```

```
              <div className="text-sm text-gray-500">Потрібно</div>
              <div className="text-lg font-bold text-orange-600">
                {task.totalHours} год
              </div>
              {scheduledHours > 0 && (
                <div className="text-xs text-gray-500 mt-1">
                  Заплановано: {scheduledHours.toFixed(1)} год
                </div>
              )}
            </div>
          </div>
        </div>
      );
    })}
    <div className="mt-3 p-3 bg-blue-50 rounded-lg border border-blue-200">
      <p className="text-sm text-blue-800">
        <strong>💡 Рекомендації:</strong>
      </p>
      <ul className="text-sm text-blue-700 mt-1 ml-4 list-disc space-y-1">
        <li>Збільште доступність годин у розділі "Календар"</li>
        <li>Перенесіть дедлайни завдань на пізніші дні</li>
        <li>Зменште буфер безпеки в розділі "Налаштування"</li>
        <li>Спробуйте оптимізувати розклад</li>
      </ul>
    </div>
  </div>
 </Card>
)}

{/* Розклад по днях */}
<div className="space-y-2">
  {daySchedules.map((day) => {
    const loadPercentage = (day.totalHours / day.capacityHours) * 100;
    const isOverloaded = day.isOverloaded;
    const date = dayjs(day.date);

    return (
      <Card
        key={day.dayIndex}
        className={`shadow-sm ${isOverloaded ? 'border-2 border-red-400' : ''}`}
        bodyStyle={{ padding: '16px' }}
```

```
>
  <div className="space-y-2">
    <div className="flex justify-between items-start">
      <div>
        <h3 className="text-lg font-semibold flex items-center gap-2">
          День {day.dayIndex}
          {isOverloaded && (
            <Tag color="red" icon={<WarningOutlined />}>
              Перевантажено
            </Tag>
          )}
        </h3>
        <p className="text-sm text-gray-500">
          {date.format('DD.MM.YYYY, dddd')}
        </p>
      </div>
      <div className="text-right">
        <div className={`text-xl font-bold ${isOverloaded ? 'text-red-500' : 'text-blue-600'}`}>
          {day.totalHours.toFixed(1)} / {day.capacityHours} год
        </div>
        <Progress
          percent={Math.min(loadPercentage, 100)}
          status={isOverloaded ? 'exception' : 'normal'}
          showInfo={false}
          size="small"
        />
      </div>
    </div>

    {day.blocks.length > 0 ? (
      <div className="space-y-1">
        {day.blocks.map((block) => {
          const task = getTaskById(block.taskId);
          if (!task) return null;

          const isAtRisk = tasksAtRisk.includes(task.id);

          return (
            <div
              key={block.id}
              className={`p-2 rounded border ${
```

```
                isAtRisk
                  ? 'bg-orange-50 border-orange-200'
                  : 'bg-gray-50 border-gray-200'
                }`}
              >
                <div className="flex justify-between items-center">
                  <div className="flex-1">
                    <div className="flex items-center gap-2">
                      <span className="font-medium">{task.title}</span>
                      {isAtRisk && (
                        <Tag color="orange" icon={<WarningOutlined />}>
                          Ризик
                        </Tag>
                      )}
                    </div>
                    <div className="text-sm text-gray-500 flex items-center gap-2 mt-1">
                      <ClockCircleOutlined />
                      {/* TODO: дедлайн */}
                      Дедлайн: День {task.deadlineDay}
                    </div>
                  </div>
                  <div className="text-right">
                    <div className="text-lg font-semibold text-blue-600">
                      {block.hours.toFixed(1)} год
                    </div>
                  </div>
                </div>
              </div>
            );
          })}
        </div>
      ) : (
        <div className="text-center py-4 text-gray-400">
          Вільний день
        </div>
      )}
    </div>
  </Card>
);
})}
</div>
```

```
  </div>
 );
}
```

hillClimbing.ts

```
import type { Task, CalendarDay, ScheduleBlock, Schedule } from '../types';
import { calculateDayLoads, calculateScheduleMetrics, generateId } from './utils';

/**
* Оцінює якість розкладу (нижче = краще)
*/
function evaluateSchedule(
 blocks: ScheduleBlock[],
 tasks: Task[],
 calendar: CalendarDay[]
): number {
 const metrics = calculateScheduleMetrics(blocks, tasks, calendar);
  // Комбінуємо різні фактори в єдину оцінку
 // Пріоритети:
 // 1. Перевантажені дні (критично!)
 // 2. Завдання з ризиком (важливо)
 // 3. Пікове навантаження (для комфорту)
 // 4. Дисперсія (рівномірність)
 const score =
   metrics.overloadedDays * 1000 +      // Найвищий пріоритет
   metrics.tasksAtRisk * 500 +          // Дуже важливо
   metrics.peakLoad * 50 +              // Збільшено з 2 до 50!
   metrics.loadVariance * 20;           // Збільшено з 10 до 20!
  return score;
}

/**
* Знаходить перевантажені дні
*/
function findOverloadedDays(
 blocks: ScheduleBlock[],
 calendar: CalendarDay[]
): number[] {
 const loads = calculateDayLoads(blocks, calendar);
 const overloadedDays: number[] = [];
```

```
  calendar.forEach(day => {
   const load = loads.get(day.dayIndex) || 0;
   if (load > day.capacityHours) {
     overloadedDays.push(day.dayIndex);
   }
  });
  return overloadedDays;
}

/**
* Знаходить дні з вільною місткістю
*/
function findDaysWithCapacity(
 blocks: ScheduleBlock[],
 calendar: CalendarDay[],
 excludeDays: number[] = []
): number[] {
 const loads = calculateDayLoads(blocks, calendar);
 const daysWithCapacity: number[] = [];
  calendar.forEach(day => {
   if (excludeDays.includes(day.dayIndex)) return;

   const load = loads.get(day.dayIndex) || 0;
   if (load < day.capacityHours) {
     daysWithCapacity.push(day.dayIndex);
   }
  });
  return daysWithCapacity;
}

/**
* Намагається перенести частину блоку в інший день
*/
function tryMoveBlock(
 blocks: ScheduleBlock[],
 fromDay: number,
 toDay: number,
 tasks: Task[],
 calendar: CalendarDay[],
 bufferDays: number = 2,
 allowPastDeadline: boolean = false
```

```
): ScheduleBlock[] | null {
 const loads = calculateDayLoads(blocks, calendar);
  // Знаходимо блоки у перевантаженому дні
 const blocksInDay = blocks.filter(b => b.dayIndex === fromDay);
 if (blocksInDay.length === 0) return null;
  // Вибираємо випадковий блок для перенесення
 const blockToMove = blocksInDay[Math.floor(Math.random() * blocksInDay.length)];
  // Перевіряємо дедлайн завдання
 const task = tasks.find(t => t.id === blockToMove.taskId);
 if (!task) {
   console.log(`[tryMoveBlock] Task not found for block`, blockToMove.taskId);
   return null;
 }
  // Розраховуємо ефективний дедлайн з урахуванням буфера
 // (так само як у EDD планувальнику)
 const effectiveDeadline = Math.max(1, task.deadlineDay - bufferDays);
  // Не переносимо блок після ефективного дедлайну (тільки якщо це критично)
 // Для балансування дозволяємо переміщення вперед
 if (!allowPastDeadline && toDay > effectiveDeadline && fromDay <= effectiveDeadline) {
   console.log(`[tryMoveBlock] Cannot move past effective deadline (strict mode)`, {
     task: task.title,
     fromDay,
     toDay,
     deadline: task.deadlineDay,
     effectiveDeadline,
     bufferDays
   });
   return null;
 }
  // Логування для балансування
 if (allowPastDeadline && toDay > effectiveDeadline) {
   console.log(`[tryMoveBlock] Balancing mode: allowing move past effective deadline`, {
     task: task.title,
     fromDay,
     toDay,
     deadline: task.deadlineDay,
     effectiveDeadline,
     bufferDays
   });
 }
  // Розраховуємо скільки можемо перенести
```

```
 const toCapacity = calendar.find(d => d.dayIndex === toDay)?.capacityHours || 0;
 const toLoad = loads.get(toDay) || 0;
 const availableInTo = toCapacity - toLoad;
  if (availableInTo <= 0) return null;
  // Переміщуємо частину блоку (до 50% або весь блок, якщо він малий)
 // Це дозволяє ефективніше балансувати навантаження
 const maxToMove = Math.max(0.5, blockToMove.hours * 0.5);
 const hoursToMove = Math.min(blockToMove.hours, availableInTo, maxToMove);
  if (hoursToMove < 0.5) return null;
  // Створюємо новий розклад з переміщенням
 const newBlocks = blocks.filter(b => b.id !== blockToMove.id);
  // Якщо переносимо не весь блок, залишаємо частину
 if (blockToMove.hours > hoursToMove) {
   newBlocks.push({
     ...blockToMove,
    hours: blockToMove.hours - hoursToMove
   });
 }
  // Додаємо новий блок у цільовий день
 newBlocks.push({
   id: generateId(),
   dayIndex: toDay,
   taskId: blockToMove.taskId,
   hours: hoursToMove
 });
  return newBlocks;
}

/**
* Знаходить дні з найвищим навантаженням для балансування
*/
function findHighLoadDays(
 blocks: ScheduleBlock[],
 calendar: CalendarDay[]
): number[] {
 const loads = calculateDayLoads(blocks, calendar);
  // Сортуємо дні за навантаженням (від найбільшого до найменшого)
 const sortedDays = calendar
   .map(day => ({
     dayIndex: day.dayIndex,
     load: loads.get(day.dayIndex) || 0,
```

```
    capacity: day.capacityHours
  }))
  .filter(day => day.load > 0)
  .sort((a, b) => b.load - a.load);
 // Беремо топ 50% найбільш завантажених днів
 const topCount = Math.max(1, Math.ceil(sortedDays.length * 0.5));
 return sortedDays.slice(0, topCount).map(d => d.dayIndex);
}

/**
* Знаходить дні з найнижчим навантаженням для балансування
*/
function findLowLoadDays(
 blocks: ScheduleBlock[],
 calendar: CalendarDay[],
 excludeDays: number[] = []
): number[] {
 const loads = calculateDayLoads(blocks, calendar);
 // Сортуємо дні за навантаженням (від найменшого до найбільшого)
 const sortedDays = calendar
  .filter(day => !excludeDays.includes(day.dayIndex))
  .map(day => ({
    dayIndex: day.dayIndex,
    load: loads.get(day.dayIndex) || 0,
    capacity: day.capacityHours,
    available: day.capacityHours - (loads.get(day.dayIndex) || 0)
  }))
  .filter(day => day.available > 0.5) // Є хоча б 0.5 год вільного місця
  .sort((a, b) => a.load - b.load);
 // Беремо топ 50% найменш завантажених днів
 const topCount = Math.max(1, Math.ceil(sortedDays.length * 0.5));
 return sortedDays.slice(0, topCount).map(d => d.dayIndex);
}

/**
* Виконує одну ітерацію Hill Climbing
*/
function performIteration(
 blocks: ScheduleBlock[],
 tasks: Task[],
 calendar: CalendarDay[],
```

```
 bufferDays: number = 2
): { blocks: ScheduleBlock[], improved: boolean } {
 const currentScore = evaluateSchedule(blocks, tasks, calendar);
  // Спочатку намагаємося вирішити перевантаження (пріоритет!)
 const overloadedDays = findOverloadedDays(blocks, calendar);
 let fromDays: number[];
 let toDays: number[];
 let isBalancingMode = false;
  if (overloadedDays.length > 0) {
   // Є перевантажені дні - працюємо з ними
   fromDays = overloadedDays;
   toDays = findDaysWithCapacity(blocks, calendar, overloadedDays);
   isBalancingMode = false;
   console.log('[Optimization] Mode: Fixing overloaded days', {
     overloadedDays,
     daysWithCapacity: toDays
   });
 } else {
   // Немає перевантаження - балансуємо навантаження
   fromDays = findHighLoadDays(blocks, calendar);
   toDays = findLowLoadDays(blocks, calendar, fromDays);
   isBalancingMode = true;
   console.log('[Optimization] Mode: Balancing load', {
     highLoadDays: fromDays,
     lowLoadDays: toDays,
     currentScore
   });
 }
  if (fromDays.length === 0 || toDays.length === 0) {
   console.log('[Optimization] No days to work with', { fromDays, toDays });
   return { blocks, improved: false };
 }
  // Пробуємо різні комбінації перенесень
 let bestBlocks = blocks;
 let bestScore = currentScore;
 let foundImprovement = false;
 let attemptedMoves = 0;
 let successfulMoves = 0;
  for (const fromDay of fromDays) {
   for (const toDay of toDays) {
     // Обмеження відстані: для перевантажень - суворе, для балансування - немає
```

```
    if (!isBalancingMode && Math.abs(fromDay - toDay) > 10) {
      continue; // Тільки для режиму виправлення перевантажень
    }

    attemptedMoves++;
    // В режимі балансування дозволяємо переміщення за ефективний дедлайн
    const newBlocks = tryMoveBlock(blocks, fromDay, toDay, tasks, calendar, bufferDays, isBalancingMode);
    if (!newBlocks) continue;

    successfulMoves++;
    const newScore = evaluateSchedule(newBlocks, tasks, calendar);

    if (newScore < bestScore) {
      console.log(`[Optimization] Found improvement! ${currentScore} → ${newScore}`, {
        from: fromDay,
        to: toDay,
        improvement: currentScore - newScore
      });
      bestBlocks = newBlocks;
      bestScore = newScore;
      foundImprovement = true;
    }
  }
 }
  if (!foundImprovement) {
   console.log('[Optimization] No improvement found in iteration', {
    attemptedMoves,
    successfulMoves,
    currentScore
   });
  }
  return { blocks: bestBlocks, improved: foundImprovement };
}

/**
* Оптимізує розклад використовуючи Hill Climbing
*/
export function optimizeSchedule(
 schedule: Schedule,
 tasks: Task[],
 calendar: CalendarDay[],
```

```
 bufferDays: number = 2,
 maxIterations: number = 200
): Schedule {
 console.log('[Optimization] Starting optimization', {
  totalBlocks: schedule.blocks.length,
  maxIterations,
  bufferDays
 });
 const initialMetrics = calculateScheduleMetrics(schedule.blocks, tasks, calendar);
 console.log('[Optimization] Initial metrics', initialMetrics);
 let currentBlocks = [...schedule.blocks];
 let iterations = 0;
 let improved = true;
 while (improved && iterations < maxIterations) {
  const result = performIteration(currentBlocks, tasks, calendar, bufferDays);
  currentBlocks = result.blocks;
  improved = result.improved;
  iterations++;

  if (!improved) {
   console.log(`[Optimization] Stopped at iteration ${iterations} (no improvement)`);
  }
 }
 const finalMetrics = calculateScheduleMetrics(currentBlocks, tasks, calendar);
 console.log('[Optimization] Finished!', {
  iterations,
  initialScore: evaluateSchedule(schedule.blocks, tasks, calendar),
  finalScore: evaluateSchedule(currentBlocks, tasks, calendar),
  initialMetrics,
  finalMetrics
 });
 return {
  blocks: currentBlocks,
  isOptimized: true,
  generatedAt: new Date().toISOString()
 };
}

/**
* Перепланує після пропущених блоків
*/
```

```
export function rescheduleAfterMiss(
 missedBlocks: ScheduleBlock[],
 currentSchedule: Schedule,
 tasks: Task[],
 calendar: CalendarDay[],
 currentDay: number,
 bufferDays: number = 2
): Schedule {
 // Видаляємо пропущені блоки з розкладу
 const remainingBlocks = currentSchedule.blocks.filter(
   block => !missedBlocks.find(mb => mb.id === block.id)
 );
  // Створюємо оновлений календар (тільки майбутні дні)
 const futureCalendar = calendar.filter(day => day.dayIndex >= currentDay);
  if (futureCalendar.length === 0) {
   return {
     blocks: remainingBlocks,
     isOptimized: false,
     generatedAt: new Date().toISOString()
   };
 }
  // Розраховуємо залишкову місткість днів
 const loads = calculateDayLoads(remainingBlocks, calendar);
 const remainingCapacity = new Map<number, number>();
  futureCalendar.forEach(day => {
   const load = loads.get(day.dayIndex) || 0;
   remainingCapacity.set(day.dayIndex, day.capacityHours - load);
 });
  // Перерозподіляємо пропущені години
 const newBlocks = [...remainingBlocks];
  for (const missedBlock of missedBlocks) {
   let hoursToAllocate = missedBlock.hours;
   const task = tasks.find(t => t.id === missedBlock.taskId);
   if (!task) continue;

   // Розраховуємо ефективний дедлайн з урахуванням буфера
   const effectiveDeadline = Math.max(1, task.deadlineDay - bufferDays);

   // Спочатку намагаємося розмістити години до ефективного дедлайну
   const daysBeforeEffectiveDeadline = futureCalendar
     .filter(day => day.dayIndex <= effectiveDeadline)
```

```
    .sort((a, b) => a.dayIndex - b.dayIndex); // Від раннього до пізнього

  for (const day of daysBeforeEffectiveDeadline) {
    if (hoursToAllocate <= 0) break;

    const available = remainingCapacity.get(day.dayIndex) || 0;
    if (available <= 0) continue;

    const hoursToAssign = Math.min(hoursToAllocate, available);

    newBlocks.push({
      id: generateId(),
      dayIndex: day.dayIndex,
      taskId: task.id,
      hours: hoursToAssign
    });

    remainingCapacity.set(day.dayIndex, available - hoursToAssign);
    hoursToAllocate -= hoursToAssign;
  }

  // Якщо не вмістилося, використовуємо буферні дні (між effectiveDeadline і deadlineDay, НЕ включаючи день
дедлайну!)
  if (hoursToAllocate > 0 && effectiveDeadline < task.deadlineDay - 1) {
    const bufferDaysRange = futureCalendar
      .filter(day => day.dayIndex > effectiveDeadline && day.dayIndex < task.deadlineDay) // < а не <=
      .sort((a, b) => a.dayIndex - b.dayIndex);

    for (const day of bufferDaysRange) {
      if (hoursToAllocate <= 0) break;

      const available = remainingCapacity.get(day.dayIndex) || 0;
      if (available <= 0) continue;

      const hoursToAssign = Math.min(hoursToAllocate, available);

      newBlocks.push({
        id: generateId(),
        dayIndex: day.dayIndex,
        taskId: task.id,
        hours: hoursToAssign
```

```
    });

    remainingCapacity.set(day.dayIndex, available - hoursToAssign);
    hoursToAllocate -= hoursToAssign;
   }
  }

  // Крайній випадок: якщо ще залишилися години, ставимо в день дедлайну (краще ніж після дедлайну)
  if (hoursToAllocate > 0) {
   const deadlineDayData = futureCalendar.find(day => day.dayIndex === task.deadlineDay);
   if (deadlineDayData) {
    const available = remainingCapacity.get(task.deadlineDay) || 0;
    if (available > 0) {
     const hoursToAssign = Math.min(hoursToAllocate, available);

     newBlocks.push({
      id: generateId(),
      dayIndex: task.deadlineDay,
      taskId: task.id,
      hours: hoursToAssign
     });

     remainingCapacity.set(task.deadlineDay, available - hoursToAssign);
     hoursToAllocate -= hoursToAssign;
    }
   }
  }

  // Якщо все ще не вмістилося, ставимо після дедлайну
  if (hoursToAllocate > 0) {
   const daysAfterDeadline = futureCalendar
    .filter(day => day.dayIndex > task.deadlineDay)
    .sort((a, b) => a.dayIndex - b.dayIndex);

   for (const day of daysAfterDeadline) {
    if (hoursToAllocate <= 0) break;

    const available = remainingCapacity.get(day.dayIndex) || 0;
    if (available <= 0) continue;

    const hoursToAssign = Math.min(hoursToAllocate, available);
```

```
      newBlocks.push({
        id: generateId(),
        dayIndex: day.dayIndex,
        taskId: task.id,
        hours: hoursToAssign
      });

      remainingCapacity.set(day.dayIndex, available - hoursToAssign);
      hoursToAllocate -= hoursToAssign;
    }
  }
 }
 return {
  blocks: newBlocks,
  isOptimized: false,
  generatedAt: new Date().toISOString()
 };
}
```

eddScheduler.ts

```
import type { Task, CalendarDay, ScheduleBlock, Schedule } from '../types';
import { generateId } from './utils';

/**
* Отримує числове значення пріоритету для сортування
*/
function getPriorityValue(priority: string): number {
 switch (priority) {
  case 'high':
    return 1;
  case 'medium':
    return 2;
  case 'low':
    return 3;
  default:
    return 2;
 }
}
```

```
/**
* Сортує завдання за EDD (Earliest Due Date)
* 1. Спочатку за дедлайном (найближчий перший)
* 2. При однакових дедлайнах - за пріоритетом (high > medium > low)
* 3. При однакових дедлайнах і пріоритетах - за тривалістю (більші раніше)
*/
function sortTasksByUrgency(tasks: Task[]): Task[] {
 return [...tasks].sort((a, b) => {
  // 1. Головний критерій - дедлайн (раніше = важливіше)
  if (a.deadlineDay !== b.deadlineDay) {
   return a.deadlineDay - b.deadlineDay;
  }

  // 2. При однакових дедлайнах - пріоритет
  const priorityA = getPriorityValue(a.priority);
  const priorityB = getPriorityValue(b.priority);
  if (priorityA !== priorityB) {
   return priorityA - priorityB;
  }

  // 3. При однакових дедлайнах і пріоритетах - тривалість (більші раніше, щоб почати завчасно)
  return b.totalHours - a.totalHours;
 });
}

/**
* Розподіляє години завдання від дня 1 до effectiveDeadline
* EDD: завдання з ранніми дедлайнами займають найближчі вільні дні
* Буфер = мінімальна відстань до дедлайну, але можна раніше
*/
function allocateTaskHours(
 task: Task,
 calendar: CalendarDay[],
 remainingCapacity: Map<number, number>,
 bufferDays: number = 2
): ScheduleBlock[] {
 const blocks: ScheduleBlock[] = [];
 let hoursToAllocate = task.totalHours - (task.completedHours || 0);
  if (hoursToAllocate <= 0) {
  return blocks;
 }
```

```
 // Ефективний дедлайн = реальний дедлайн - буфер днів
// Це ОСТАННІЙ можливий день для завершення (не раніше!)
const effectiveDeadline = Math.max(1, task.deadlineDay - bufferDays);
 console.log(`    [allocate] Task: ${task.title}, need ${hoursToAllocate}h, effectiveDeadline: ${effectiveDeadline}`);
 // EDD: Розподіляємо ВПЕРЕД від дня 1 до effectiveDeadline
// Завдання з раннім дедлайном займе ранні дні (бо обробляється першим)
// Буфер обмежує МАКСИМУМ, але не МІНІМУМ
const sortedDays = [...calendar]
  .filter(day => day.dayIndex <= effectiveDeadline)
  .sort((a, b) => a.dayIndex - b.dayIndex); // ВПЕРЕД: від дня 1
 for (const day of sortedDays) {
  if (hoursToAllocate <= 0) break;

  const available = remainingCapacity.get(day.dayIndex) || 0;
  if (available <= 0) {
    console.log(`    [allocate] Day ${day.dayIndex}: skipped (no capacity)`);
    continue;
  }

  const hoursToAssign = Math.min(hoursToAllocate, available);

  blocks.push({
    id: generateId(),
    dayIndex: day.dayIndex,
    taskId: task.id,
    hours: hoursToAssign
  });

  remainingCapacity.set(day.dayIndex, available - hoursToAssign);
  hoursToAllocate -= hoursToAssign;
  console.log(`    [allocate] Day ${day.dayIndex}: assigned ${hoursToAssign}h (${available - hoursToAssign}h left)`);
}
 // Якщо ще залишилися години, спробуємо використати буферні дні (між effectiveDeadline і realDeadline, НЕ
включаючи день дедлайну!)
 if (hoursToAllocate > 0 && effectiveDeadline < task.deadlineDay - 1) {
  console.log(`    [allocate] Still need ${hoursToAllocate}h, trying buffer days (${effectiveDeadline +
1}-${task.deadlineDay - 1})`);
  const bufferDaysRange = calendar
    .filter(day => day.dayIndex > effectiveDeadline && day.dayIndex < task.deadlineDay) // < а не <=, щоб не
включати день дедлайну
    .sort((a, b) => a.dayIndex - b.dayIndex); // Від раннього до пізнього
```

```
  for (const day of bufferDaysRange) {
   if (hoursToAllocate <= 0) break;

   const available = remainingCapacity.get(day.dayIndex) || 0;
   if (available <= 0) {
    console.log(`    [allocate] Day ${day.dayIndex} (buffer): skipped (no capacity)`);
    continue;
   }

   const hoursToAssign = Math.min(hoursToAllocate, available);

   blocks.push({
    id: generateId(),
    dayIndex: day.dayIndex,
    taskId: task.id,
    hours: hoursToAssign
   });

   remainingCapacity.set(day.dayIndex, available - hoursToAssign);
   hoursToAllocate -= hoursToAssign;
   console.log(`    [allocate] Day ${day.dayIndex} (buffer): assigned ${hoursToAssign}h`);
  }
 }
 // Крайній випадок: якщо ще залишилися години, ставимо в день дедлайну (краще ніж після дедлайну)
 if (hoursToAllocate > 0) {
  const deadlineDay = calendar.find(day => day.dayIndex === task.deadlineDay);
  if (deadlineDay) {
   const available = remainingCapacity.get(task.deadlineDay) || 0;
   if (available > 0) {
    const hoursToAssign = Math.min(hoursToAllocate, available);

    blocks.push({
     id: generateId(),
     dayIndex: task.deadlineDay,
     taskId: task.id,
     hours: hoursToAssign
    });

    remainingCapacity.set(task.deadlineDay, available - hoursToAssign);
    hoursToAllocate -= hoursToAssign;
```

```
      console.log(`    [allocate] Day ${task.deadlineDay} (DEADLINE DAY!): assigned ${hoursToAssign}h `);
    }
   }
  }
  // Якщо все ще залишилися години, це означає перевантаження - ставимо після дедлайну
  if (hoursToAllocate > 0) {
   console.log(`    [allocate]  OVERLOAD! Still need ${hoursToAllocate}h, scheduling after deadline`);
   const daysAfterDeadline = calendar
    .filter(day => day.dayIndex > task.deadlineDay)
    .sort((a, b) => a.dayIndex - b.dayIndex);

   for (const day of daysAfterDeadline) {
    if (hoursToAllocate <= 0) break;

    const available = remainingCapacity.get(day.dayIndex) || 0;
    if (available <= 0) {
     console.log(`    [allocate] Day ${day.dayIndex} (after deadline): skipped (no capacity)`);
     continue;
    }

    const hoursToAssign = Math.min(hoursToAllocate, available);

    blocks.push({
     id: generateId(),
     dayIndex: day.dayIndex,
     taskId: task.id,
     hours: hoursToAssign
    });

    remainingCapacity.set(day.dayIndex, available - hoursToAssign);
    hoursToAllocate -= hoursToAssign;
    console.log(`    [allocate] Day ${day.dayIndex} (after deadline): assigned ${hoursToAssign}h`);
   }
  }
  if (hoursToAllocate > 0) {
   console.log(`    [allocate]  CRITICAL: ${hoursToAllocate}h could not be scheduled at all!`);
  }
  return blocks;
 }

 /**
```

```
* Генерує початковий розклад використовуючи модифікований EDD
*/
export function generateInitialSchedule(
 tasks: Task[],
 calendar: CalendarDay[],
 bufferDays: number = 2
): Schedule {
 console.log('[EDD] Starting initial schedule generation', {
  totalTasks: tasks.length,
  totalDays: calendar.length,
  bufferDays,
  totalHours: tasks.reduce((sum, t) => sum + t.totalHours, 0),
  totalCapacity: calendar.reduce((sum, d) => sum + d.capacityHours, 0)
 });
 // Ініціалізуємо залишкову місткість днів
 const remainingCapacity = new Map<number, number>();
 calendar.forEach(day => {
  remainingCapacity.set(day.dayIndex, day.capacityHours);
 });
 // Сортуємо завдання за терміновістю
 const sortedTasks = sortTasksByUrgency(tasks);
 console.log('[EDD] Tasks sorted by urgency:', sortedTasks.map(t => ({
  title: t.title,
  deadline: t.deadlineDay,
  hours: t.totalHours,
  priority: t.priority
 })));
 // Розподіляємо коже завдання
 const allBlocks: ScheduleBlock[] = [];
 for (const task of sortedTasks) {
  console.log(`[EDD] Allocating task: ${task.title} (${task.totalHours}h, deadline: ${task.deadlineDay})`);
  const taskBlocks = allocateTaskHours(task, calendar, remainingCapacity, bufferDays);
  console.log(`[EDD] Task allocated to ${taskBlocks.length} blocks:`, taskBlocks.map(b => ({
   day: b.dayIndex,
   hours: b.hours
  })));
  allBlocks.push(...taskBlocks);
 }
 // Логуємо фінальний стан розкладу
 const dayLoads = new Map<number, number>();
 allBlocks.forEach(block => {
```

```
  const current = dayLoads.get(block.dayIndex) || 0;
  dayLoads.set(block.dayIndex, current + block.hours);
 });
 console.log('[EDD] Final schedule by days:');
 calendar.forEach(day => {
  const load = dayLoads.get(day.dayIndex) || 0;
  const capacity = day.capacityHours;
  console.log(`  Day ${day.dayIndex}: ${load}/${capacity} hours (${capacity - load} free)`);
 });
 return {
  blocks: allBlocks,
  isOptimized: false,
  generatedAt: new Date().toISOString()
 };
}
```

ДОДАТОК В

Результати перевірки роботи на співпадіння

StrikePlagiarism.com

Дата звіту 12/7/2025
Дата редагування 12/8/2025

Документ прийнятий

Звіт подібності

Метадані

Назва організації
National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute
Заголовок
ІП43-мп_Курильченко_ПЗ
Автор Науковий керівник / Експерт
КурильченкоЛісовиченко О.І.
підрозділ
ФІОТ, К-а інформатики та програмної інженерії

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

15619
Кількість слів

121263
Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

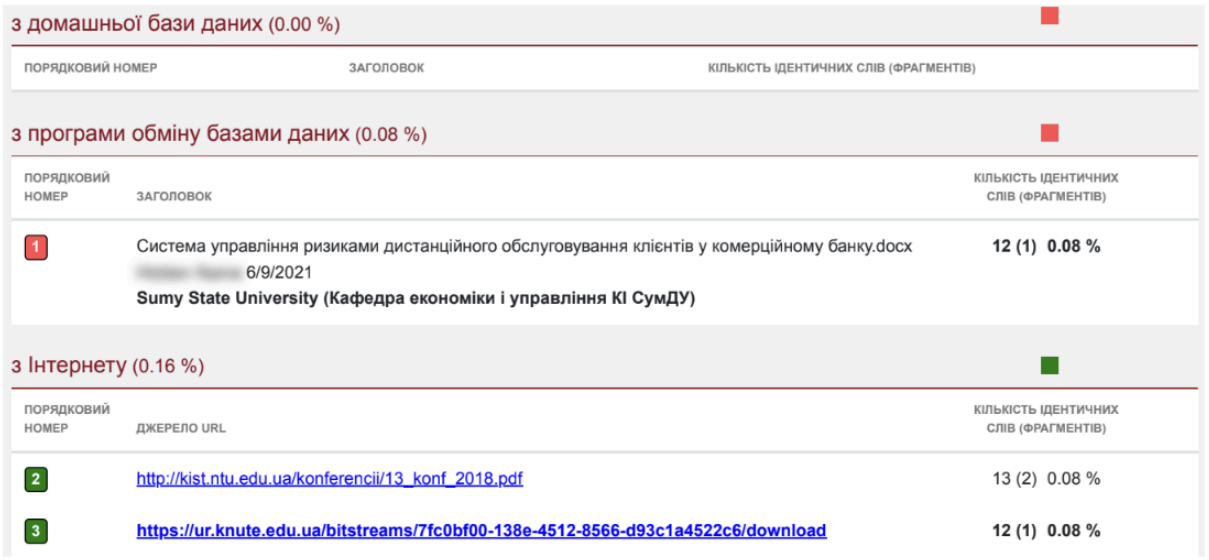
Заміна букв	ß	1
Інтервали	A→	0
Мікропробіли	¶	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	4

Джерела

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://ur.knute.edu.ua/bitstreams/7fc0bf00-138e-4512-8566-d93c1a4522c6/download	12 0.08 %
2	Система управління ризиками дистанційного обслуговування клієнтів у комерційному банку.docx [redacted] 6/9/2021 Sumy State University (Кафедра економіки і управління КІ СумДУ)	12 0.08 %
3	http://kist.ntu.edu.ua/konferencii/13_konf_2018.pdf	8 0.05 %
4	http://kist.ntu.edu.ua/konferencii/13_konf_2018.pdf	5 0.03 %



з домашньої бази даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з програми обміну базами даних (0.08 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Система управління ризиками дистанційного обслуговування клієнтів у комерційному банку.docx [redacted] 6/9/2021 **Sumy State University (Кафедра економіки і управління КІ СумДУ)**	**12 (1) 0.08 %**

з Інтернету (0.16 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
2	http://kist.ntu.edu.ua/konferencii/13_konf_2018.pdf	13 (2) 0.08 %
3	**https://ur.knute.edu.ua/bitstreams/7fc0bf00-138e-4512-8566-d93c1a4522c6/download**	**12 (1) 0.08 %**

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)