Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Веб-застосунок «Розумний розклад»
08-54.БКР.029.00.000 ПЗ

Виконав: студент 4 курсу, групи 2СП-21 б
спеціальності123 — «Комп'ютерна інженерія»
(шифр і назва напряму підготовки, спеціальності)
освітня програма — «Системне програмування»
Коваль М.В.
(прізвище та ініціали)

Керівник: к.пед.н., доц. каф. ОТ
Добровольська Н.В.
(прізвище та ініціали)
Рецензент: д.т.н., проф. каф. ПЗ
Ліщинська Л.Б.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д

"___" _____2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо—кваліфікаційний рівень бакалвр
Галузь знань — 12 Інформаційні технології
Спеціальність — 123 Комп'ютерна інженерія
Освітня програма — Системне програмування

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д
_____
"21" березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Ковалю Максиму Віталійовичу

1 Тема проєкту: Веб-застосунок «Розумний розклад», керівник роботи к.пед.н., доц. каф. ОТ Добровольська Н.В, затверджено наказом вищого навчального закладу від «20» березня 2025 року №97.

2 Строк подання студентом проєкту: 13.05.2025

3 Вихідні дані до проєкту: створення розумнуго розкладу для тих, хто поєднує роботу та навчання.

4 Зміст розрахунково-пояснювальної записки: Розумний розклад для тих, хто поєднує роботу та навчання.

5 Перелік графічного матеріалу: веб-застосунок.

6 Консультанти розділів роботи приведені в таблиці 1.

7 Дата видачі завдання 21.03.2025 р.

8 Календарний план приведений в таблиці 2.

3) презентацію у форматі pdf з його підписом ФОТО або СКАН;

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-3	к.пед.н., доц. каф. ОТ Добровольська Н.В.	21.03.25	13.05.25
Нормоконтроль	ас. каф. ОТ Швець С. І.	14.05.25	30.05.25

Таблиця 2 — Календарний план

№ з/п	Назва етапів дипломної Роботи	Строк виконання	Примітка
1	Постановка задачі роботи	21.03.25	виконано
2	Пошук матеріалів та інформаційних джерел	21.03.25— 30.03.25	виконано
3	Аналіз існуючих прототипів та визначення вимог до створюваної системи	21.03.25— 30.03.25	виконано
4	Обгрунтування структури та принципів роботи системи	31.03.25— 13.04.25	виконано
5	Розробка й тестування апаратно-програмної системи	14.04.25— 27.04.25	виконано
7	Оформлення пояснювальної записки та ілюстративного матеріалу	28.04.25— 11.05.25	виконано
8	Перевірка якості виконання бакалаврської кваліфікаційної роботи та усунення недоліків	13.05.25	виконано

Студент Максим КОВАЛЬ

Керівник роботи к.пед.н., доц. Наталія ДОБРОВОЛЬСЬКА

АНОТАЦІЯ

УДК 621.3

Коваль М.В. Веб-застосунок «Розумний розклад». Бакалаврська кваліфікаційна робота (проєкт) зі спеціальності 123 «Комп'ютерна інженерія», освітня програма «Системне програмування». Вінниця: ВНТУ, 2025. 75с.

75 с. На укр. мові. Бібліогр.: 23 назв; рис.: 20; табл. 11.

Розроблено клієнтська частина з використанням React (TypeScript) і Tailwind CSS, серверна логіка – на Node.js/Express із безпечною автентифікацією JWT, а зберігання даних реалізовано на PostgreSQL з кешуванням через Redis. Застосовано Google Calendar API для інтеграції календарних подій. В основі планування подій закладено алгоритмічні підходи – greedy, генетичні та методи гілки й межі – з урахуванням пріоритетів, обмежень і навантаження.

Практична цінність проєкту — у зниженні когнітивного навантаження, підвищенні продуктивності та покращенні балансу між навчанням і роботою. Результати засвідчують перспективність подальшого розвитку проєкту в напрямах мікросервісної архітектури, машинного навчання, мобільних клієнтів і A/B тестування.

Ключові слова: розумний розклад, React, Node.js, Docker, CI/CD, оптимізація розкладу, інтеграція з календарем, PWA.

ABSTRACT

UDK 621.3

Koval M.V. Web application "Smart Schedule". Bachelor's qualification work (project) in specialty 123 "Computer Engineering", educational program "System Programming". Vinnytsia: VNTU, 2025. 75 p.

75 p. In Ukrainian. Bibliography: 23 titles; Fig.: 20; Table. 11.

The client part was developed using React (TypeScript) and Tailwind CSS, the server logic was implemented on Node.js/Express with secure JWT authentication, and data storage was implemented on PostgreSQL with caching via Redis. The Google Calendar API was used to integrate calendar events. The event planning is based on algorithmic approaches - greedy, genetic, and branch and bound methods - taking into account priorities, constraints, and load.

The practical value of the project is in reducing cognitive load, increasing productivity and improving the balance between learning and work. The results indicate the prospects for further development of the project in the areas of microservice architecture, machine learning, mobile clients and A/B testing.

Keywords: smart scheduling, React, Node.js, Docker, CI/CD, schedule optimization, calendar integration, PWA.

ЗМІСТ

ВСТУП

У сучасному динамічному світі ефективне управління часом стає ключовим фактором успіху, особливо для людей, які поєднують кілька сфер діяльності, таких як робота та навчання. Широке розповсюдження цифрових технологій створює можливості для розробки інструментів, що допомагають оптимізувати щоденні процеси, однак універсальні рішення не завжди враховують специфічні потреби окремих груп користувачів. Одним із важливих напрямів є створення інтелектуальних веборієнтованих систем планування, які забезпечують не лише облік подій, а й активну допомогу в організації часу.

На фоні зростання кількості студентів, що працюють, та підвищення вимог до їхньої самоорганізації, виникає потреба в спеціалізованих онлайн-інструментах. Стандартні календарі та планувальники часто виявляються недостатньо гнучкими для управління складними, змінними графіками роботи та навчання, не пропонують автоматичної оптимізації та не інтегрують усі необхідні аспекти життя користувача в єдину систему. Актуальною є задача створення веб-застосунку, який би поєднував алгоритми розумного планування, інтуїтивний інтерфейс для легкої взаємодії, кросплатформеність та інтеграцію з популярними сервісами календарів. При цьому важливими є не лише технічна реалізація алгоритмів оптимізації, а й зручність використання (UX), адаптивність інтерфейсу до потреб користувача та надання дієвих інструментів для подолання часових конфліктів.

Актуальність теми обумовлена необхідністю забезпечення працюючих студентів високоякісним веб-інструментом для ефективного управління часом, що автоматизує рутинні процеси планування, пропонує інтелектуальні рішення для оптимізації розкладу, відповідає сучасним стандартам UI/UX-дизайну та є технічно надійним і відкритим до подальшого розвитку.

Метою дослідження є розробка інтерактивної веб-системи розумного розкладу для осіб, що поєднують роботу та навчання, з підтримкою автоматичної оптимізації графіку, реалізацією адаптивного інтерфейсу користувача та інтеграцією із зовнішніми сервісами календарів.

Для досягнення поставленої мети потрібно вирішити такі **завдання:**

— проаналізувати існуючі підходи до управління часом для працюючих студентів та сучасні технології для створення інтелектуальних систем планування;

— визначити оптимальну архітектуру, алгоритми оптимізації та технологічний стек (фреймворки, бібліотеки, API) для реалізації логіки розумного розкладу та інтеграцій;

— спроєктувати структуру інтерфейсу користувача, забезпечивши інтуїтивність введення даних, наочність візуалізації розкладу та адаптивність до різних пристроїв;

— реалізувати функціональність введення робочих та навчальних графіків, автоматичного пошуку вільних слотів, оптимізації розкладу на основі пріоритетів;

— забезпечити тестування системи на функціональну коректність, ефективність алгоритмів та зручність використання, проаналізувати результати взаємодії користувачів із застосунком;

— провести оптимізацію продуктивності та покращення UX/UI на основі отриманих результатів тестування.

Об’єктом дослідження є процес проектування, розробки та оптимізації веборієнтованих систем для інтелектуального управління часом та автоматизованого планування розкладу.

Предметом дослідження є архітектура, алгоритми оптимізації та адаптації, функціональні можливості, користувацький інтерфейс та механізми інтеграції із зовнішніми сервісами у процесі реалізації веб-системи розумного розкладу для працюючих студентів.

Практична цінність розробленого застосунку полягає в наданні працюючим студентам ефективного інструменту для збалансування робочого навантаження та навчального процесу, зменшення стресу, пов'язаного з браком часу, та підвищення загальної продуктивності. Створена система може бути використана індивідуально або запропонована як допоміжний сервіс у закладах

освіти. Гнучкість архітектури та використані технології дозволяють у майбутньому розширювати функціонал системи, додаючи підтримку нових типів завдань, складніші алгоритми ШІ для персоналізації або адаптуючи її для інших категорій користувачів зі складними графіками.

Апробація матеріалів проєкту виконано в доповіді Молодь в науці: дослідження, проблеми, перспективи (МН-2025).

Публікація за темою проєкту [10]: Коваль М. В., Добровольська Н.В. Веб-застосунок «Розумний розклад» // Молодь в науці: дослідження, проблеми, перспективи (МН-2025). – Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25175.

1 ТЕОРЕТИЧНІ ТА ТЕХНОЛОГІЧНІ ЗАСАДИ СТВОРЕННЯ СИСТЕМИ РОЗУМНОГО РОЗКЛАДУ

1.1 Аналіз підходів до планування часу для працюючих студентів

Питання ефективного управління часом для студентів, які поєднують навчання з роботою, є вкрай актуальним. Значна кількість студентів стикається з проблемою нестачі часу, перевтомою, а також зниженням ефективності як в навчанні, так і в професійній діяльності. У таких умовах особливо важливо застосовувати перевірені та адаптивні методи тайм-менеджменту, які допоможуть збалансувати навчальні та робочі зобов'язання з особистими потребами, що, у свою чергу, сприятиме підвищенню продуктивності та зменшенню рівня стресу [1].

Одним з найпоширеніших методів управління часом є система GTD (Getting Things Done), розроблена Девідом Алленом. Ця методика базується на чітких принципах і передбачає створення докладних списків завдань, їх розподіл за важливістю та терміновістю, а також регулярний перегляд цих списків для актуалізації планів і завдань. GTD також рекомендує використання спеціальних папок або програмних додатків для сортування завдань за пріоритетами і контроль їх виконання. Завдяки систематичності та простоті застосування, методика дозволяє значно знизити рівень стресу і підвищити якість виконання задач, що особливо важливо для студентів, які працюють, через постійну завантаженість та необхідність швидко реагувати на зміни у розкладі [2].

Ще одним популярним методом управління часом є метод Pomodoro, запропонований Франческо Чірілло. Основна ідея цього підходу полягає в тому, що робочий процес розбивається на певні часові інтервали (найчастіше 25 хвилин), які називаються «помідорами». Після завершення кожного інтервалу робиться коротка перерва тривалістю приблизно 5 хвилин, а після чотирьох таких циклів робиться більш тривала перерва у 15-30 хвилин. Цей підхід допомагає зберігати високу концентрацію на завданнях, зменшує ймовірність перевтоми і сприяє загальному підвищенню продуктивності праці. Особливо корисним він є для працюючих студентів, які мають дуже щільний графік і

потребують регулярних коротких перерв для відновлення сил і збереження продуктивності протягом тривалого часу [3].

Рисунок 1.1 — Демонстрація матриці Ейзенхаузера

Метод пріоритетів Ейзенхауера також широко застосовується студентами, які поєднують роботу і навчання. Цей метод передбачає розподіл завдань на чотири основні категорії відповідно до їхньої важливості та терміновості: завдання, які є терміновими та важливими, виконуються негайно; важливі, але не термінові — плануються на майбутнє; термінові, але не важливі — делегуються, а ті, що не є ані терміновими, ані важливими — відкладаються або повністю виключаються зі списку справ (див. рис. 1.1). Цей підхід допомагає студентам уникнути перевантаження другорядними завданнями і зосередитися

на найважливіших цілях, що особливо важливо в умовах обмежених часових ресурсів і високих вимог до ефективності виконання завдань .

Окрім традиційних методів управління часом, значну популярність серед студентів набувають сучасні цифрові інструменти. Серед таких інструментів особливо слід виділити Google Calendar, Notion та Trello. Google Calendar дозволяє ефективно планувати робочі і навчальні завдання, автоматично нагадує про майбутні події, підтримує інтеграцію з іншими сервісами Google та синхронізується між різними пристроями. Notion надає широкі можливості для ведення баз даних завдань, планування та організації інформації у зручному вигляді. Trello, в свою чергу, орієнтований на візуалізацію процесу виконання завдань, дозволяючи створювати інтерактивні дошки, картки і списки з чітким відображенням прогресу по кожному завданню. Використання таких інструментів допомагає студентам, які працюють, краще контролювати свій час, своєчасно реагувати на зміни та ефективно керувати своїми завданнями [4].

В умовах активного поєднання роботи та навчання студентам критично необхідно використовувати програмні засоби, що забезпечують не лише базове планування, а й гнучке управління завданнями з урахуванням частих змін у розкладі. Сучасні інструменти планування повинні підтримувати синхронізацію з різними пристроями, автоматичні сповіщення про наближення дедлайнів, можливість інтеграції з комунікаційними платформами для швидкого обміну інформацією та аналітику використання часу. Багато працездатних студентів поєднують кілька проєктів одночасно, тому імперативом стає можливість створювати власні шаблони подій, настроювати пріоритети та фільтри, а також переглядати статистику виконання завдань за обраний період. Крім того, важливими є функції спільного доступу до календарів та завдань, які дозволяють координувати колективні активності або ділитися власним розкладом з колегами та викладачами. Усі перелічені вимоги роблять актуальним детальне порівняння функціональних характеристик різних програмних засобів для планування та управління часом.

для організації графіків та планування є Google Calendar. Цей програмний засіб характеризується простим і зручним інтерфейсом, що робить його доступним навіть для користувачів без спеціальних технічних навичок. Google Calendar дозволяє легко створювати та редагувати події, налаштовувати повторювані заходи, а також автоматично отримувати нагадування на електронну пошту або мобільний пристрій. Завдяки можливості синхронізації з різними пристроями та інтеграції з іншими сервісами Google, такими як Gmail, Google Meet або Google Tasks, студенти можуть ефективно планувати навчальні та робочі завдання. Проте серед недоліків варто зазначити обмежені аналітичні функції, недостатню гнучкість у налаштуванні пріоритетності задач та неможливість докладного планування великих проектів.

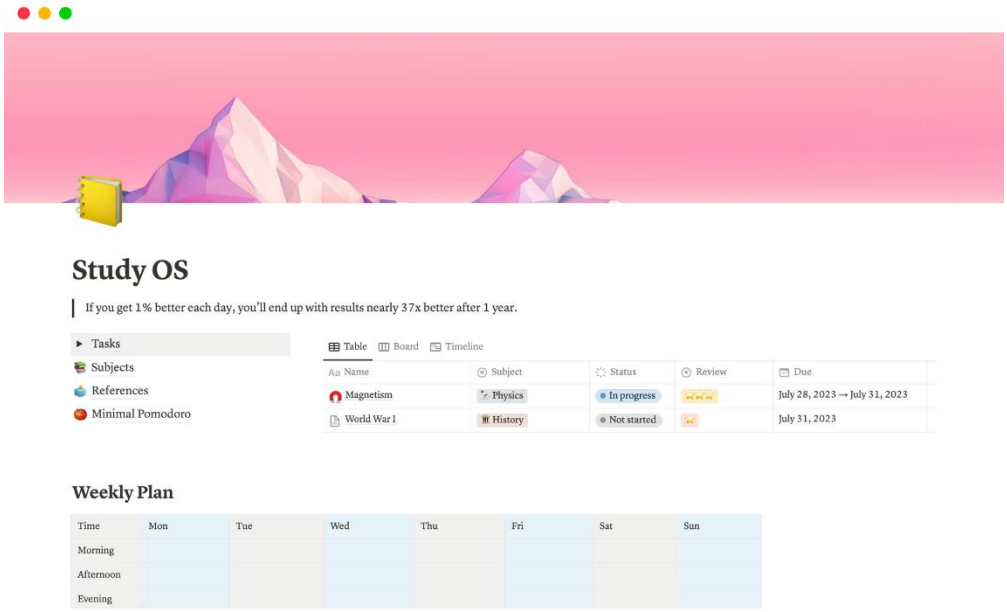

Рисунок 1.2 — Приклад реалізованого шаблону для ведення навчання

Програмний засіб Notion є потужним інструментом для організації особистої та командної роботи, що вирізняється високим рівнем гнучкості й адаптивності. Його головною перевагою є можливість повного налаштування робочого простору відповідно до потреб користувача (див. рис. 1.2). Завдяки цьому Notion підходить як для ведення особистих щоденників, списків завдань і навчальних конспектів, так і для управління складними проєктами в командному середовищі. Користувачі можуть створювати власні шаблони, які відповідають

специфіці їхньої діяльності, формувати багатоаспектні бази даних, у яких можна зберігати будь-яку інформацію — від коротких нотаток до структурованих таблиць із фільтрами, тегами та формулами. Вбудовані механізми надання статусів, позначок і переглядів (наприклад, Kanban-дошки, календарі, списки або галереї) дають змогу здійснювати аналітичний моніторинг прогресу, контролювати виконання завдань і швидко адаптуватися до змін.

Notion також інтегрує функції спільної роботи, включаючи коментування, сповіщення та можливість редагування в реальному часі, що робить його зручним інструментом для колективної взаємодії. Усе це дозволяє об’єднувати в одному інтерфейсі різноманітні аспекти управління інформацією — від особистого тайм-менеджменту до повноцінного управління проєктами.

Однак, незважаючи на переваги, важливо зазначити, що Notion має доволі високий поріг входження. Велика кількість функцій, елементів для налаштування та відсутність заздалегідь структурованих рішень можуть створити труднощі для нових користувачів. Щоб ефективно використовувати всі можливості платформи, необхідно витратити певний час на ознайомлення з інтерфейсом, логікою побудови сторінок, типами вмісту та доступними шаблонами. Тому, хоча Notion є універсальним і надзвичайно гнучким інструментом, його ефективність значною мірою залежить від цифрової грамотності користувача та готовності інвестувати час у налаштування.

Ще одним ефективним інструментом для управління задачами є Trello. Його основною перевагою є проста та зрозуміла візуальна структура організації завдань у вигляді дошок, списків та карток. Користувач може швидко бачити статус виконання завдань, переміщуючи їх між списками відповідно до прогресу роботи. Trello особливо добре підходить для роботи у команді, адже надає можливості легкого спільного доступу та редагування задач усіма учасниками проекту в режимі реального часу. Втім, серед недоліків Trello варто відзначити недостатньо розвинені функції календарного планування, а також обмежені можливості інтеграції з іншими сервісами без використання додаткових розширень чи плагінів.

Для детальнішого порівняння зазначених програмних засобів доцільно використовувати порівняльну таблицю 1.1, яка дає змогу чітко побачити переваги та недоліки кожної програми (див. табл. 1.1).

Таблиця 1.1 — Характеристики популярних сервісів з планування

Критерій	Google Calendar	Notion	Trello
Простота використання	Дуже висока; інтуїтивний інтерфейс	Середня; потребує часу на ознайомлення	Висока; мінімальні елементи керування
Можливості інтеграції	Поглиблена з Gmail, Meet, Tasks	Широка; бази даних, сторонні API	Обмежена без плагінів та Power-Ups
Мобільні додатки	Наявні для iOS та Android; синхронізація	Наявні; повна функціональність	Наявні; базовий функціонал
Гнучкість налаштувань	Фіксовані шаблони подій	Дуже висока; власні шаблони, бази даних	Середня; колонки та картки
Спільна робота	Спільні календарі, сповіщення	Розширені права доступу, коментарі	Реальний час, призначення карток
Аналітичні можливості	Мінімальні звіти	Детальні звітні сторінки, статуси	Лише сторонні Power-Ups

Отже, вибір конкретного програмного забезпечення залежить від індивідуальних особливостей та потреб користувачів, специфіки завдань, які необхідно виконати, а також від умов навчального і робочого процесів. Правильно підібраний інструмент може суттєво підвищити ефективність роботи та зменшити рівень стресу, пов’язаного з організацією часу та завдань.

Традиційні методи планування часу, серед яких ведення паперових щоденників, журналів та роздрукованих розкладів, довго залишалися основним інструментом організації діяльності. Паперові формати мають низку переваг, зокрема тактильність і візуальне сприйняття завдань, що допомагає краще запам’ятовувати інформацію. Додавання нотаток, коригувальних відміток чи використання різнокольорових маркерів створює відчуття повного контролю над графіком без необхідності в технічному обладнанні. Універсальність паперових

носіїв також полягає в їхній незалежності від доступу до інтернету чи електропостачання, що забезпечує стабільність збереження даних навіть в умовах обмежених ресурсів [6]. Проте ці ж переваги водночас породжують суттєві обмеження. Формат паперу не дозволяє швидко масштабувати розклад або автоматично оновлювати інформацію. Ручне сортування та передрук завдань вимагає додаткового часу, а відсутність автоматичних нагадувань створює ризик пропуску важливих дедлайнів.

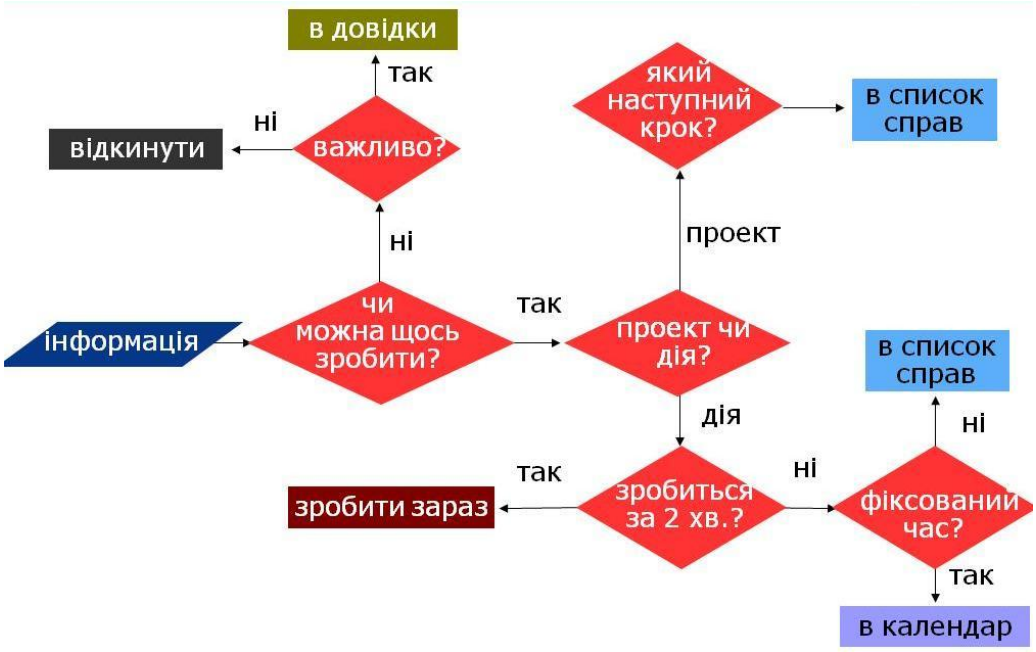

Рисунок 1.3 — Реалізація методики GTD

Методика GTD (Getting Things Done) Девіда Аллена відзначається високою структурованістю та чітким набором рекомендацій, починаючи від збору всіх завдань у «скриньку» та завершуючи щотижневим переглядом (див. рис. 1.3). Цей підхід спрямований на зниження когнітивного навантаження та покращення концентрації, що підтверджується дослідженнями ефективності GTD у професійному середовищі [7]. Однак, для працюючих студентів GTD може виявитися досить ресурсоємним через необхідність регулярного рев'ю та планування, що не завжди можливо при непередбачуваних змінах у розкладі навчання та роботи.

Метод Pomodoro, заснований на поділі робочого часу на короткі інтервали з регулярними перервами, продемонстрував високу ефективність у підвищенні продуктивності та зменшенні втоми. Згідно з дослідженням Rosa та співавторів, застосування цього підходу дозволяє підвищити загальну результативність роботи середньостатистичних офісних працівників на 20–25 %, що пояснюється чіткою структурованістю часу, зменшенням прокрастинації та кращим управлінням увагою. Проте, незважаючи на ці позитивні результати, ефективність Pomodoro напряму залежить від умов його реалізації. Метод вимагає стабільного робочого середовища, відсутності зовнішніх відволікань, а також внутрішньої дисципліни користувача, який повинен чітко дотримуватися встановлених інтервалів і не порушувати логіку чергування роботи й відпочинку.

У контексті студентської діяльності це може становити певні труднощі. Навчання, особливо в умовах гнучкого або змішаного графіка, часто супроводжується непередбачуваними змінами в розкладі, необхідністю швидко перемикатися між завданнями, виконувати термінові доручення або брати участь у спонтанних заходах. У таких випадках студенти можуть не мати змоги завершити запланований інтервал без перерв або, навпаки, відчувати тиск через необхідність дотримуватися режиму, що призводить до зниження гнучкості в управлінні власним часом. Як наслідок, це може не лише зменшити ефективність застосування Pomodoro у студентському середовищі, але й викликати фрустрацію та додатковий стрес. Отже, хоча метод Pomodoro є корисним інструментом для концентрації та підвищення продуктивності, його застосування серед студентів потребує адаптації до особливостей їхнього розкладу та гнучкішого підходу до управління інтервалами.

Матриця Ейзенхауера, що класифікує завдання за важливістю та терміновістю, дозволяє мінімізувати втрати часу на другорядні активності та чітко фокусуватися на пріоритетних завданнях. Згідно з оглядом Лі та спів. (2021), цей підхід позитивно впливає на якість виконання завдань та сприяє кращому контролю над навантаженням. Водночас матриця менш придатна для обробки великої кількості дрібних, але часто змінних завдань, оскільки механізм

делегування чи відхилення може бути недоречним для студентського середовища, де делегувати навчальні завдання часто неможливо.

Сучасні цифрові інструменти поєднують у собі елементи всіх класичних методів і доповнюють їх автоматизацією, аналітикою та можливістю спільної роботи. Google Calendar та подібні календарні сервіси дозволяють створювати повторювані події, надсилати автоматичні сповіщення, а також аналізувати розподіл часу за допомогою інтегрованих звітів. Notion, завдяки унікальній модульній структурі, об’єднує функції бази даних, нотаток та трекінгу завдань, що робить його потужним інструментом для створення індивідуальних систем управління часом. Trello із візуальним інтерфейсом Kanban сприяє швидкому відстеженню прогресу, особливо в командних проєктах [8].

Водночас цифрові платформи мають і свої недоліки. Дослідження Nguyen (2022) [9] підкреслює, що надмірна кількість налаштувань може призвести до інформаційного перевантаження та зниження мотивації, а залежність від стабільного інтернет-з’єднання та електроживлення робить такі рішення вразливими в умовах нестабільної технічної інфраструктури .

Таким чином, кожен із розглянутих підходів має свої переваги й обмеження. Традиційні методи відзначаються простотою та надійністю, але поступаються в гнучкості та автоматизації. Класичні тайм-менеджмент методи підвищують продуктивність і покращують фокус, проте вимагають значних часових ресурсів на підтримку процесу. Цифрові інструменти поєднують широку функціональність, але можуть стати джерелом вибору та технологічної залежності.

1.2 Огляд програмних рішень для організації розкладу

Визначення вимог є фундаментальним етапом розробки інформаційної системи, адже саме від якості зібраних і формалізованих вимог залежить успішність подальшого проєктування, реалізації та підтримки програмного продукту. У контексті системи «Розумний розклад» вимоги формуються з урахуванням особливостей цільової аудиторії — студентів, які поєднують

навчання та роботу, їхніх очікувань щодо гнучкості розкладу, потреб у нагадуваннях і аналітиці.

Процес створення вимог розпочинається з етапу збирання інформації (elicitation) через інтерв'ю з ключовими зацікавленими сторонами (студенти, викладачі, роботодавці) та опитуваннями. Додатково аналізуються існуючі системи планування та внутрішня документація університетських і робочих процесів, щоб визначити бізнес-правила та обмеження. Результатом цього етапу є попередній перелік функціональних та нефункціональних вимог, оформлений у вигляді заміток, user stories або use case scenarios.

Наступним кроком є аналіз і уточнення зібраних вимог. Аналітик проводить класифікацію та пріоритизацію вимог за критеріями критичності для основного сценарію використання, складності реалізації та впливу на загальну ефективність системи.

Оформлення підтверджених вимог здійснюється у вигляді Специфікації вимог до програмного забезпечення (Software Requirements Specification, SRS). Згідно з рекомендаціями IEEE, документ SRS має містити введення з описом мети та середовища використання, загальний опис системи, детальний опис функціональних можливостей, нефункціональні вимоги (продуктивність, безпека, юзабіліті) та вимоги до інтерфейсів. Для кращого розуміння взаємодії користувачів із системою в SRS включають UML-діаграми випадків використання та сценарії користувача.

Верифікація та валідація вимог передбачає перевірку правильності, повноти та узгодженості специфікації. Переглядові сесії (requirements review) та експертні оцінки дозволяють виявити неточності та пропозиції щодо покращення SRS. Використання прототипів інтерфейсів (wireframes або clickable mockups) забезпечує зворотний зв'язок від кінцевих користувачів до початку розробки, що знижує ризики помилок у реалізації.

Управління вимогами здійснюється за допомогою систем відстеження змін (Requirement Management Tools), що забезпечують прослідковуваність (traceability) вимог — від моменту їх появи до тестового підтвердження. Це

дозволяє відстежувати стан кожної вимоги, встановлювати зв’язки між вимогами, завданнями розробки та тестовими сценаріями.

У контексті «Розумного розкладу» ключові функціональні вимоги включають: реєстрацію та автентифікацію користувача, введення розкладу занять та робочого графіка, налаштування пріоритетів та параметрів оптимізації, генерацію та редагування розкладу, а також механізми інтеграції з зовнішніми сервісами. Нефункціональні вимоги охоплюють продуктивність, безпеку, доступність та зручність користування.

Таким чином, ретельно організований процес створення, аналізу, документування та валідації вимог є ключовою передумовою успіху проєкту, що дозволить «Розумному розкладу» задовольнити потреби працюючих студентів та забезпечити гнучкість, надійність і високу якість користувацького досвіду.

1.3 Методологічні основи побудови системи розумного розкладу

Існує три основні підходи до побудови програмних систем: монолітна архітектура, мікросервісна архітектура та клієнт-серверна архітектура. Кожен із цих підходів має свої особливості, які впливають на масштабованість, надійність та гнучкість системи.

Монолітна архітектура передбачає єдиний розгорнутий блок, у якому містяться всі компоненти — інтерфейс, бізнес-логіка та доступ до даних. Такий підхід спрощує розробку на початкових стадіях, оскільки вся кодова база знаходиться в одному місці, а розгортання здійснюється одним пакетом. Проте з ростом проекту моноліт може призводити до проблем у масштабуванні, оскільки не можна окремо збільшити ресурси для конкретних частин системи. Крім того, висока зв’язність компонент ускладнює обслуговування і впровадження змін, що негативно впливає на гнучкість розробки та швидкість релізів.

Рисунок 1.4 ілюструє відмінності між монолітною архітектурою та мікросервісною архітектурою у побудові програмних систем, а також демонструє, як вони можуть розгортатися на різних типах інфраструктури.

У монолітній архітектурі вся система побудована як єдиний нероздільний блок. Один великий застосунок, що містить усі необхідні функції, такі як автентифікація, обробка даних, бізнес-логіка та інтерфейс користувача, розміщується разом на одному фізичному сервері (Bare Metal). На схемі це показано як блок App Services, що безпосередньо розгортається на Bare Metal (див. рис. 1.4). У такому підході всі частини програми тісно пов’язані між собою: зміна одного компонента часто потребує повторного розгортання всієї системи. Масштабування в цьому випадку відбувається за рахунок масштабування всього додатку, а не окремих його частин.

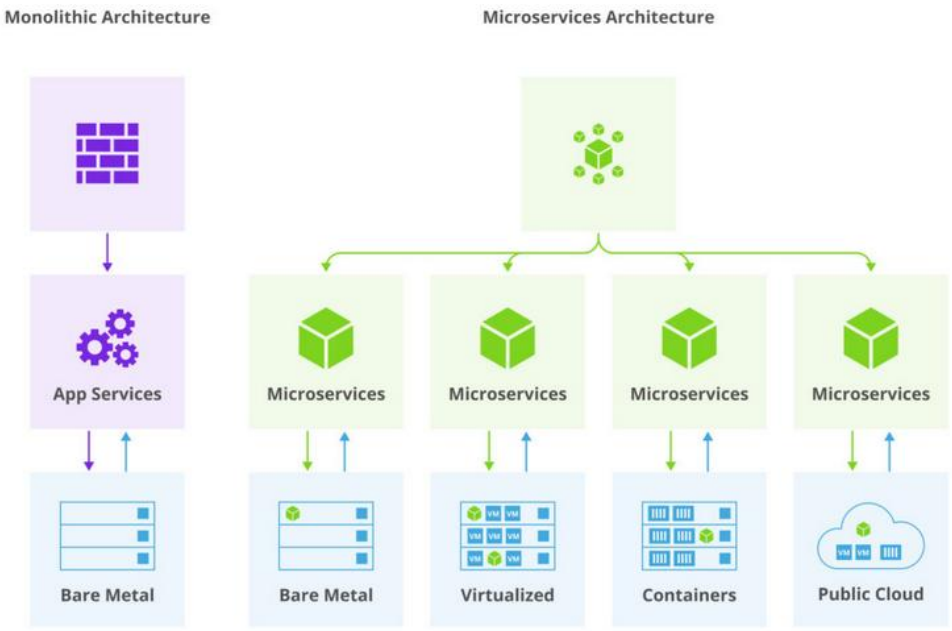

Рисунок 1.4 — Порівняння архітектур додатків

У мікросервісній архітектурі, навпаки, програма розбивається на низку незалежних сервісів, кожен з яких виконує окрему функцію. Ці мікросервіси можуть розгортатися окремо, масштабуватися незалежно один від одного, мати власні бази даних і навіть бути написаними різними мовами програмування (див. рис. 1.4). Кожен мікросервіс може бути розгорнутий на окремій інфраструктурі: на фізичному сервері (Bare Metal), у віртуальному середовищі (Virtualized), у контейнерах (Containers) або в публічній хмарі (Public Cloud). Це забезпечує гнучкість, адаптивність до навантаження та можливість використання сучасних

підходів DevOps. Комунікація між сервісами відбувається через API або шини повідомлень.

Мікросервісна архітектура передбачає розбиття системи на низку незалежних сервісів, кожен із яких виконує конкретну бізнес-функцію та має власні дані. Цей підхід значно полегшує масштабування, оскільки кожен сервіс можна розгортати й масштабувати окремо відповідно до навантаження. Крім того, мікросервіси сприяють автономному розвитку команд, що працюють над різними частинами системи, і підвищують стійкість: відмова одного сервісу не призведе до повного простою всього додатку. Однак розробка, тестування та моніторинг мікросервісів є більш складними, потребують налаштування інфраструктури сервісної мережі (Service Mesh) і відстеження транзакцій через розподілені логи, що збільшує витрати на підтримку.

Клієнт-серверна архітектура базується на поділі на два основні рівні: клієнтська частина (frontend), яка взаємодіє з користувачем, та серверна частина (backend), яка обробляє запити, виконує бізнес-логіку та працює з даними. Така модель дозволяє чітко розділити відповідальність між сторонами, знизити зв’язність та сприяє легшому масштабуванню серверної частини. Клієнт може бути різноманітним — веб-інтерфейс, мобільний додаток або десктоп клієнт.

У порівнянні з монолітом та мікросервісами клієнт-серверна архітектура забезпечує оптимальний баланс між простотою розгортання та гнучкістю. З одного боку, вона дозволяє виходити за межі одного рознесеного блоку, але з іншого — не створює надмірну складність державних комунікацій між мікросервісами. Саме тому для систем із помірним навантаженням та необхідністю швидкого запуску часто обирають клієнт-серверну модель .

У таблиці 1.2 наведено порівняння архітектурних підходів.

Узагальнюючи, можна виділити такі сильні та слабкі сторони:

— монолітна архітектура забезпечує простоту розгортання та зменшує операційні витрати на початковому етапі, але ускладнює майбутній розвиток і масштабування;

— мікросервісна архітектура дає виняткову гнучкість і стійкість, проте потребує розвиненої інфраструктури та ускладнює процеси розробки та підтримки;

— клієнт-серверна архітектура поєднує розділення відповідальності із відносною простотою розгортання, що робить її оптимальним вибором для багатьох веб-систем.

Таблиця 1.2 — Порівняння архітектурних підходів

Архітектурний підхід	Опис	Переваги	Недоліки
Монолітна	Усі компоненти (UI, логіка, дані) в одному розгортанні	Проста розробка і розгортання на початковому етапі	Складне масштабування, високий рівень зв’язності
Мікросервісна	Система розбита на незалежні сервіси з власними даними	Автономія команд, гнучке масштабування, стійкість	Складна інфраструктура (Service Mesh, розподілене логування)
Клієнт-серверна	Відокремлення frontend та backend, кожен шар можна окремо масштабувати	Баланс простоти та гнучкості, чітка ізоляція шарів	Не так гнучко, як мікросервіси, але складніше, ніж моноліт

Трирівнева клієнт-серверна архітектура забезпечує чітку роздільність між шарами презентації, бізнес-логіки та доступу до даних, що сприяє підвищенню масштабованості, тестованості та гнучкості системи. На рівні презентації (presentation layer) реалізується взаємодія з користувачем: тут знаходяться веб-інтерфейс, мобільні клієнти або десктопні додатки, що відповідають за візуалізацію даних і прийом введення. Блок бізнес-логіки (application layer) інкапсулює алгоритми обробки інформації, управління сеансами користувачів та інтеграцію з зовнішніми сервісами. Рівень доступу до даних (data layer) відповідає за збереження та вибірку інформації з баз даних або зовнішніх сховищ.

У таблиці 1.3 наведено структуру трирівневої клієнт-серверної архітектури.

Розподіл обов’язків за шарами дозволяє незалежно розвивати кожен компонент: наприклад, оптимізувати запити до бази даних чи змінювати

інтерфейс користувача без втручання у бізнес-логіку. Завдяки такому підходу логіка взаємодії з клієнтом залишається ізольованою від особливостей зберігання даних, а компоненти бізнес-логіки — від специфіки UI. Це забезпечує кращу керованість кодової бази, полегшує навантажувальне тестування окремих частин системи та пришвидшує реліз нових функцій.

Таблиця 1.3 — Структура трирівневої клієнт-серверної архітектури

Рівень	Функції	Приклади технологій	Масштабування
Presentation Layer	Відображення, прийом введення, валідація даних	React.js, TypeScript, Tailwind CSS, FullCalendar.js	Додавати екземпляри веб-серверів, CDN
Business Logic Layer	Алгоритми оптимізації, обробка запитів, авторизація	Node.js, Express.js, JWT	Горизонтальне масштабування API-серверів
Data Layer	Збереження, вибірка, міграції даних	PostgreSQL, Sequelize, Redis	Репліки БД, розподілене кешування

Крім того, тирівнева модель дозволяє реалізувати масштабування кожного шару за потреби: наприклад, встановити кілька екземплярів веб-сервера для обробки великого числа одночасних клієнтів або додати ресурси до СУБД для обробки складних запитів. Використання load balancer між клієнтами та серверами забезпечує рівномірний розподіл навантаження, а кешування на межі presentation layer (наприклад, CDN або in-memory кеш) знижує затримки у відображенні статичних ресурсів.

Таким чином, обрана трирівнева клієнт-серверна архітектура створює фундамент для побудови надійної, масштабованої та легко підтримуваної системи «Розумний розклад», де зміни в одній частині не призведуть до порушення роботи інших рівнів.

Рівень презентації відповідає за взаємодію системи зі студентом-користувачем та інкапсулює всі компоненти, пов’язані з відображенням даних і прийомом введення. Для реалізації інтерфейсу доцільно застосувати React.js у поєднанні з TypeScript. React.js, завдяки віртуальному DOM та компонентній архітектурі, забезпечує високу продуктивність і можливість розділяти UI на багато невеликих, повторно використовуваних компонентів. TypeScript додає

строгість типізації під час розробки, зменшуючи ризики помилок та полегшуючи рефакторинг коду .

Важливою складовою рівня презентації є бібліотека FullCalendar.js, яка дозволяє швидко і зручно відобразити календар з подіями, реалізувати функції перетягування (drag-and-drop), масштабування та налаштування зовнішнього вигляду. FullCalendar.js легко інтегрується з React через офіційні обгортки, забезпечуючи синхронізацію стану календаря з компонентами додатку .

Для забезпечення доступності системи з різних пристроїв та поліпшення користувацького досвіду доцільно реалізувати адаптивний дизайн із використанням CSS-фреймворку Tailwind CSS або Bootstrap. Використання медіа-запитів та гнучких сіток гарантує коректне відображення інтерфейсу на екранах різних розмірів.

Крім веб-версії, варто передбачити можливість встановлення Progressive Web App (PWA), яка дозволить користувачам працювати з додатком офлайн, отримувати push-сповіщення та встановлювати його на робочий стіл або мобільний пристрій як окремий додаток. Наявність PWA покращує доступність та зручність використання, а також сприяє зростанню залученості користувачів.

Під рівнем бізнес-логіки (application layer) реалізуються всі сервіси та процеси, що забезпечують функціональні можливості системи. Для «Розумного розкладу» доцільно використати Node.js та Express.js, які підтримують неблокуючу модель вводу-виводу та високу обробну здатність одночасних запитів. Express.js надає легковаговий фреймворк для побудови RESTful API, що спрощує маршрутизацію та реалізацію контролерів бізнес-логіки.

Розробка REST API передбачає визначення ресурсів системи (розклад, події, користувачі) та реалізацію CRUD-операцій для кожного ресурсу. Використання принципів REST дозволяє клієнтам взаємодіяти зі сервером за допомогою стандартних HTTP-методів: GET, POST, PUT, DELETE. Для захисту маршрутів і даних застосовуються JWT-токени, які передаються у заголовку Authorization та перевіряються на валідність при кожному запиті.

У таблиці 1.4 наведено технології рівня презентації.

Node.js забезпечує неблокуючий ввід/вивід завдяки однопоточному циклу подій (event loop), що дозволяє ефективно обробляти велику кількість одночасних з’єднань. Для обробки завдань, що потребують тривалої обчислювальної роботи, рекомендується використовувати worker threads або розподілені черги задач із такими інструментами, як Bull або RabbitMQ.

Таблиця 1.4 — Технології рівня презентації

Компонент	Технологія	Основні можливості	Переваги
UI-фреймворк	React.js	Віртуальний DOM, компонентна архітектура	Висока продуктивність, повторне використання коду
Статична типізація	TypeScript	Статична типізація, інтерфейси, generics	Менше помилок, легший рефакторинг
Календар	FullCalendar.js	Drag & drop, кастомізація подій, масштабування	Швидка інтеграція з React, готові UI-компоненти
Стилізація	Tailwind CSS / Bootstrap	Адаптивні сітки, медіа-запити	Швидка розробка, узгоджений дизайн
PWA	Service Worker, Web App Manifest	Офлайн-режим, push-сповіщення	Підвищена доступність, встановлення на пристрій

Для організації середовища розробки та управління залежностями варто застосувати модульну структуру проекту з розділенням коду на шари: routers, controllers, services та models. Це покращує читабельність та підтримуваність коду, дозволяє легко додавати нові функціональні можливості та тестувати бізнес-логіку за допомогою unit та integration-тестів.

Рівень доступу до даних відповідає за збереження, вибірку та маніпуляцію інформацією, необхідною для роботи системи. Основним сховищем даних для «Розумного розкладу» обрано реляційну СУБД PostgreSQL, яка відзначається високою надійністю, потужною підтримкою складних SQL-запитів та розвиненою екосистемою інструментів адміністрування та реплікації. PostgreSQL дозволяє реалізувати транзакційну обробку даних із гарантією атомарності операцій, що гарантує цілісність розкладу навіть за одночасного внесення змін кількома користувачами.

Для спрощення роботи з базою даних у середовищі Node.js використовується ORM-бібліотека Sequelize. Вона забезпечує об’єктно-

реляційне відображення моделей, дозволяє описувати зв’язки між сутностями та автоматизувати виконання міграцій схеми бази даних. Міграції реалізуються через інструментарій Sequelize CLI, що гарантує контроль версій структури бази та покращує процес розгортання оновлень.

Окрім реляційної СУБД, для забезпечення високопродуктивного кешування та обробки реального часу доцільно використовувати Redis як in-memory сховище та брокер повідомлень. Redis дозволяє тимчасово зберігати найчастіше запитувані дані (наприклад, попередньо згенеровані частини розкладу) та організовувати черги задач для асинхронної обробки.

Стратегія взаємодії між ORM та кешем передбачає перевірку at-first-cookie кешу: при отриманні запиту система перевіряє Redis-пам’ять, і лише за відсутності даних виконує запит до PostgreSQL, після чого результат примусово кешується на визначений проміжок часу. Такий підхід значно знижує навантаження на базу даних та покращує час відгуку системи.

Інтеграція з зовнішніми сервісами є невід’ємною складовою сучасних інформаційних систем, оскільки дозволяє розширювати функціональність без повторного винаходу відомих рішень. У «Розумному розкладі» передбачено синхронізацію з Google Calendar через Google Calendar API. За допомогою OAuth 2.0 користувачі можуть надати системі доступ до своїх календарів, а серверні сервіси будуть періодично використовувати REST-запити для отримання та створення подій у зовнішньому календарі.

Окрім прямих інтеграцій, система надає відкритий RESTful API для сторонніх клієнтських додатків та сервісів. Документація до API формується за допомогою Swagger (OpenAPI Specification), що дозволяє генерувати зрозумілі інтерактивні специфікації та SDK для різних мов програмування. Це забезпечує швидке підключення зовнішніх розробників та полегшує інтеграцію нових модулів.

Забезпечення якості програмного продукту та швидкість доставки оновлень до користувачів значною мірою залежить від правильно налаштованих процесів безперервної інтеграції та доставки (CI/CD). Для «Розумного розкладу»

рекомендується використовувати GitHub Actions або GitLab CI з реалізацією таких етапів у CI/CD pipeline (див. табл. 1.5):

— побудова (build) — це автоматична збірка фронтенд- та бекенд-проектів, перевірка відповідності коду стилістичним стандартам (linting) та компіляція TypeScript-коду, що гарантує єдність середовища збірки;

— тестування (test) — це запуск unit-, integration- та end-to-end тестів для виявлення регресій на ранніх стадіях, включаючи аналіз покриття тестами з метою оцінки якості коду;

— розгортання (deploy) — це автоматизоване розгортання збірок на тестове середовище, а після успішного проходження тестів - у staging або production за допомогою скриптів Docker Compose або Kubernetes manifests.

Контейнеризація за допомогою Docker забезпечує консистентність середовищ розгортання на локальних машинах розробників, CI-серверах та продуктивних кластерах. Визначення Dockerfile для кожного сервісу (frontend, backend, база даних) дозволяє стандартизувати процес запуску, а Docker Compose надає зручний інструмент для підняття локального multi-container оточення.

У виробничому середовищі доцільно використовувати оркестратор контейнерів Kubernetes, що забезпечує автоматичне масштабування (Horizontal Pod Autoscaler), балансування навантаження та самовідновлення сервісів. Налаштування Helm-чартів спрощує управління конфігурацією та оновленням додатку, мінімізуючи простої при розгортанні нових версій [4].

Таблиця 1.5 — CI/CD і середовища розгортання

Етап	Інструменти	Опис	Результат
Build	GitHub Actions, ESLint, TypeScript	Збірка коду, linting, компіляція	Консистентна збірка, виявлення стилістичних помилок
Test	Jest, Mocha, Cypress	Unit, integration, e2e тести	Виявлення регресій, звіт покриття тестами
Deploy	Docker Compose, Kubernetes, Helm	Розгортання на staging/production	Автоматизоване оновлення сервісів
Моніторинг	Prometheus, Grafana, ELK Stack	Збір метрк, логування, візуалізація	Оперативне реагування на інциденти

Моніторинг та логування мають бути інтегровані в CI/CD pipeline — системи Prometheus та Grafana збирають метрики продуктивності та стану додатку, а ELK Stack (Elasticsearch, Logstash, Kibana) забезпечує централізоване зберігання та аналіз логів. Це дозволяє оперативно виявляти та усувати проблеми, що виникають у продуктивному середовищі.

У результаті всебічного аналізу та обґрунтування архітектурних підходів, а також підбору технологічного стеку для реалізації системи «Розумний розклад», було ухвалено рішення використати трирівневу клієнт-серверну архітектуру. Розділення на рівні презентації, бізнес-логіки та доступу до даних забезпечує чітку ізоляцію відповідальностей, що спрощує підтримку та розвиток кожного компонента.

Застосування React.js і TypeScript на рівні презентації гарантує високу продуктивність і зручність для користувача, а використання Node.js і Express.js дозволяє ефективно обробляти запити та реалізувати гнучкий RESTful API. Використання PostgreSQL як основної СУБД у поєднанні з Redis для кешування забезпечує надійність збереження даних та миттєве реагування системи. Інтеграція з Google Calendar API розширює можливості синхронізації, а впровадження CI/CD з Docker і Kubernetes гарантує безперервну доставку якісного коду.

Запропонована архітектура поєднує простоту розгортання та масштабованість, забезпечуючи швидку адаптацію до росту навантажень та зміни вимог. Вона створює міцний фундамент для подальшого розвитку системи, дозволяючи інтегрувати нові сервіси, розширювати функціонал та підтримувати високу якість користувацького досвіду.

У сучасних веб-додатках інтерфейс користувача виконує роль мосту між складними алгоритмами планування та реальними потребами користувача. У «Розумному розкладі» інтерфейс створюється таким чином, щоб кожний його елемент був зрозумілим з першого погляду, дозволяв мінімізувати кількість кроків для виконання завдання і забезпечував гнучкість налаштувань згідно з індивідуальними уподобаннями.

Насамперед інтерфейс проектується згідно з десятьма евристиками Нільсена-Нормана, серед яких особлива увага приділяється видимості стану системи та відповідності між системою й реальним світом. Це означає, що під час завантаження календаря користувач одразу бачить завантажувальний індикатор та поточний період перегляду. Перш ніж користувач здійснить будь-яку дію, інтерфейс повинен чітко відобразити доступні опції та їхній поточний стан.

Для уніфікації дизайну використовуються перевірені патерни взаємодії. Картки з подіями демонструють основні атрибути (назва, час, пріоритет) у стислій формі, а деталізовані дані відкриваються у модальному вікні, що не відволікає від основного контенту. Бічні панелі застосовуються для налаштувань профілю та системних параметрів, дозволяючи користувачу повернутись до основного інтерфейсу без перезавантаження сторінки. Підхід компонентного дизайну робить можливим повторне використання елементів у різних контекстах, що зменшує обсяг коду та спрощує його тестування.

Ключовим елементом є календарний перегляд, який підтримує зміну масштабу від добового до місячного звіту та інтерактивні дії, такі як перетягування карток подій та миттєве оновлення позицій. Важливою особливістю є можливість відображення кількох шарів, наприклад навчальні пари, робочі зміни та особисті справи, кожен з яких має власний колір для швидкої візуальної диференціації. Форма введення події відкривається у спливаючому вікні з полями для вибору часу, тривалості, типу активності й пріоритету, автозаповнення та контекстні підказки покращують швидкість введення даних.

Аналітична панель інтегрована безпосередньо поруч із календарем і демонструє графіки розподілу часу за категоріями, середню завантаженість у певні періоди та рекомендації щодо оптимізації. Замість традиційного списку фільтрів система надає пошуковий рядок із підтримкою розширеного синтаксису (наприклад, групові запити «робота AND завтра»), що забезпечує швидкий доступ до потрібних завдань.

Важливою складовою є профіль користувача, де зберігаються налаштування теми оформлення (світла чи темна), контрасту і розміру шрифту. Згідно з рекомендаціями WCAG 2.1, користувачі можуть змінювати інтервал між елементами інтерфейсу та відстань між рядками для покращення читабельності. Параметри пріоритетів дозволяють задати вагу різних типів активностей, що безпосередньо впливає на роботу алгоритму оптимізації.

Доступність інтерфейсу забезпечується дотриманням стандартів WCAG 2.1 рівня AA, включаючи підтримку навігації за допомогою клавіатури, чіткі описи елементів через ARIA-атрибути та альтернативні тексти для всіх іконок. Шрифти вибрані з урахуванням рекомендацій щодо кернінгу і міжрядкових інтервалів для зменшення візуальної втоми. Крім того, передбачено збільшення масштабів інтерфейсу незалежно від системних налаштувань браузера, що сприяє користувачам із порушеннями зору.

Сумісність з різними платформами досягається через адаптивний дизайн та тестування на емуляторах мобільних пристроїв. Спрощена навігація та мінімальний обсяг інформації на екрані під час виконання основних дій сприяють комфортному використанню системи для всіх категорій користувачів.

Підсумовуючи, виконане дослідження та обґрунтування технічних рішень утворюють міцний фундамент для подальшої розробки й імплементації системи «Розумний розклад». Затверджене планування вимог, обрана архітектура та технологічний стек створюють умови для швидкої розробки, ефективного тестування і масштабування проекту, а також забезпечують високу якість кінцевого продукту.

2 АЛГОРИТМІЧНЕ ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Розробка алгоритмів для оптимізації розкладу

Процес оптимізації розкладу передбачає формалізацію вимог, які забезпечать генерування збалансованого та адаптивного розподілу подій. Основні вимоги до алгоритму оптимізації включають рівномірність навантаження, врахування пріоритетів користувачів та дотримання часових обмежень. Визначення цих критеріїв дозволяє створити основу для подальшого вибору та налаштування конкретних методів і структур даних.

Рівномірність навантаження означає, що робочі та навчальні активності мають бути розподілені по всьому часовому інтервалу таким чином, щоб уникнути піків перевантаження. Актуальною метрикою в цій частині є середнє та максимальне навантаження на одиницю часу, яке алгоритм прагне мінімізувати [1]. Для досягнення рівномірного розподілу часто використовують жадібні стратегії, які на кожному кроці розміщують подію в найбільш «порожній» часовий слот, а також евристичні підходи, що опрацьовують глобальний стан розкладу перед остаточним призначенням (див. рис. 2.1).

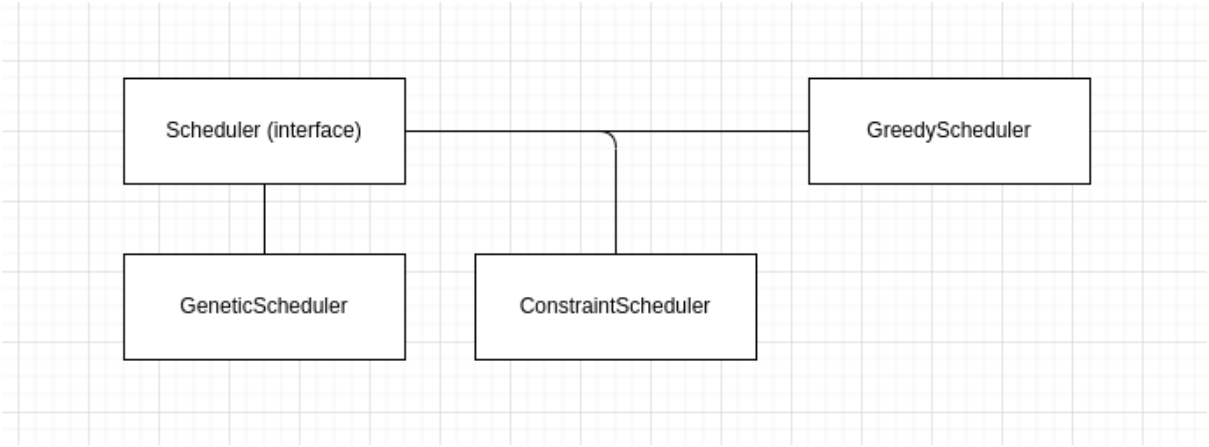

Рисунок 2.1 — Класова діаграма

Врахування пріоритетів установлює вагу для кожної події відповідно до її важливості для користувача. Користувач може позначити заняття з вищим пріоритетом (наприклад, іспити чи робочі зміни) як критичні, а менш важливі активності (наприклад, відпочинок чи додаткові тренування) — як вторинні.

Алгоритм повинен забезпечувати, щоб завдання з вищими пріоритетами отримували кращі часові слоти і мали мінімальні зрушення за потреби [2]. Одним із способів реалізації є модифікація цільової функції оптимізації, де кожна подія множиться на коефіцієнт її пріоритету.

Часові обмеження накладають межі на те, коли події можуть бути заплановані. Ці обмеження включають фіксовані часові вікна (наприклад, пара «9:00–11:00» або зміна «14:00–18:00») та ліміти загальної тривалості активностей на день. При формулюванні задачі оптимізації вони задаються через обмеження на допустимі інтервали та агреговані обмеження сумарного часу. Для коректної роботи з такими обмеженнями використовуються методи з пошуком по обмеженнях (Constraint Programming) та гілки та межі (Branch and Bound) [3].

Узагальнення вимог дозволяє визначити характеристики цільової функції оптимізації:

$$F = w_1 * max_{load} + w_2 * \sum_i p_i * \delta_i + w_3 * \sum_j \tau_j,$$

де max_load — максимальне навантаження в одній часовій одиниці;

p_i — пріоритет i-тої події;

δ_i — відхилення від бажаного слота;

τ_j — порушення j-го часовового обмеження;

а w_k — вагові коефієнти.

Далі на основі цієї функції можна вибрати відповідний алгоритм: жадібний для швидких наближених рішень або еволюційний (генетичний) для пошуку якісніших результатів за складніших умов.

Жадібний алгоритм для оптимізації розкладу ґрунтується на поступовому призначенні кожної події до найбільш прийнятного часового інтервалу, що дозволяє швидко формувати початковий варіант графіка. На першому етапі збираються всі вхідні дані. Список подій із зазначенням тривалості, пріоритетів та можливих часових вікон. Далі відбувається сортування цього списку за

обраним критерієм — найчастіше за спаданням важливості (пріоритету) або за найкоротшою тривалістю події, що знижує ймовірність фрагментації розкладу.

Після сортування жадібний алгоритм ітерує всі події. Для кожної обирається найменш завантажений часовий слот із доступних інтервалів, який не порушує жодних обмежень. Якщо подія не може бути розміщена через конфлікт часових вікон, вона відкладається в список відкладених завдань для подальшої обробки чи ручного коригування.

Ефективність такого підходу обумовлена простотою імплементації й мінімальними ресурсними витратами. Часова складність основного етапу — сортування — оцінюється як O(n log n), де n — кількість подій. Кожне призначення в слот із використанням пріоритетної черги чи бінарного пошуку в наборах слотів додатково коштує O(log n), тому загалом складність алгоритму перебуває в межах O(n log n).

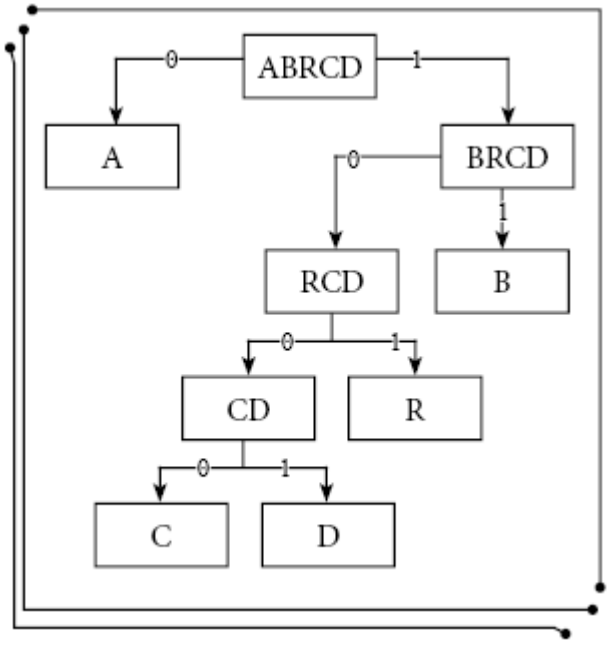

Рисунок 2.2 — Приклад жадібного алгоритму

Жадібні стратегії особливо доречні для сценаріїв, у яких швидке отримання прийнятного рішення є критичним. Наприклад, для попередньої генерації чернеток розкладу або live scheduling у системах, де необхідно оперативно реагувати на запити користувачів. Типовим прикладом є алгоритм

«найближчий термін» (Earliest Deadline First), який сортує завдання за датою завершення і розподіляє їх у перші доступні слоти, що забезпечує високу ймовірність вчасного виконання ключових завдань (див. рис. 2.2).

Однак локальність рішень у жадібному підході може призводити до субоптимальних результатів у глобальному масштабі. У задачах із перехресними обмеженнями, де розміщення однієї події може сильно впливати на можливість розміщення інших, жадібні алгоритми можуть залишати значні прогалини або формувати нерівномірний розподіл навантаження. Також вони не гарантують знаходження глобального оптимуму у випадках, де цільова функція включає нелінійні або складні взаємозв'язки між подіями.

У порівнянні з методом повного перебору (exhaustive search) жадібні алгоритми мають набагато менші часові витрати, але втрачають у якості рішення. У практичних системах саме цей компроміс часто є прийнятним: з одного боку — швидкість реагування, а з іншого — прийнятний рівень задоволення критичних критеріїв планування.

Генетичні алгоритми (GA) є типом еволюційних обчислень, які імітують природний добір для пошуку оптимальних рішень у складних просторах варіантів. Для оптимізації розкладу генетичний підхід передбачає представлення кожного можливого розподілу подій як хромосоми (послідовності генів), де гени відповідають позиціям подій у часових слотах.

Принцип роботи:

— ініціалізація популяції генерується початковий набір хромосом випадковим або напіввипадковим способом, що забезпечує різноманітність рішень;

— оцінка пристосованості (fitness), кожна хромосома оцінюється за цільовою функцією оптимізації F, де враховуються рівномірність навантаження, пріоритети та часові обмеження;

— селекція зазвичай використовуються турнірний відбір або відбір за рулетковим принципом, щоб обрати батьківські хромосоми для створення нового покоління;

— при схрещуванні (crossover) обрані батьки обмінюються частинами хромосом у визначеній точці, що дає нові комбінації генів;

— мутація з невеликою ймовірністю випадкові гени змінюються для підтримки різноманітності та запобігання передчасній конвергенції;

— замінення поколінь тоді нові хромосоми замінюють найгірші рішення у попередньому поколінні.

Цикл селекції, схрещування та мутації повторюється до досягнення критеріїв зупинки: встановленого числа поколінь або коли поліпшення пристосованості стає незначним.

Часова складність генетичного алгоритму оцінюється як O(g * p * (n + c)), де g — кількість поколінь, p — розмір популяції, n — кількість подій (для оцінки fitness), а c — вартість операцій схрещування та мутації. Збільшення розміру популяції чи кількості поколінь підвищує якість рішення, але зростає обчислювальне навантаження.

Випадки застосування — складні розклади зі змінними обмеженнями: коли потрібно врахувати численні додаткові фактори (кваліфікація викладачів, доступність аудиторій, індивідуальні переваги).

Підбір оптимального розподілу ресурсів у великих університетських або корпоративних середовищах, де кількість подій може досягати сотень на добу.

Гібридні підходи — поєднання жадібного початкового кроку з подальшим поліпшенням GA для більш швидкого зближення з високоякісним рішенням.

Генетичні алгоритми забезпечують гнучкість та адаптивність, проте потребують ретельного налаштування гіперпараметрів — розміру популяції, ймовірності мутації та числа поколінь — щоб уникнути передчасної конвергенції або надто повільного пошуку.

Окрім жадібних та генетичних алгоритмів, для оптимізації складних розкладів можуть застосовуватись методи повного перебору з ефективними стратегіями відсікання та повернення. Серед них найбільш розповсюджені підходи Branch and Bound та Backtracking.

Branch and Bound (Гілки та межі) — це метод гілок і меж передбачає систематичну побудову дерева рішень, де кожна вершина відповідає частковому розподілу подій по слотам. На кожному кроці обчислюється оцінка (оцінкова функція) найкращого можливого розв'язку серед нащадків поточної вершини. Якщо ця оцінка гірша за найкраще знайдене рішення, вся піддерево відсікається (bound). Це дозволяє значно зменшити простір пошуку в порівнянні з повним перебором [1]. У розкладі, наприклад, можна відкидати варіанти, де максимальне навантаження перевищує поточний найкращий показник або порушуються часові обмеження.

Принцип роботи — дерево рішень будується шляхом ітеративного додавання подій до часткових рішень. Для кожної нової події обчислюється нижня межа цільової функції для можливих позицій, що дозволяє відразу відсікати невдалий вибір.

У найгіршому випадку експоненційна, O(b^d), де b — середня гілковість, d — глибина дерева (кількість подій). Практична ефективність значно вища завдяки ранньому відсіченню.

Випадки застосування можуть бути точні планування з невеликою кількістю подій або з жорсткими обмеженнями, наприклад, розподіл лабораторних засідань з фіксованими аудиторіями.

Backtracking (Пошук з поверненням) є рекурсивним методом перебору, що будує рішення крок за кроком і повертається на попередній рівень, якщо поточний шлях не призводить до допустимого розв'язку. У задачах розкладу це означає спробу помістити кожну подію у всі можливі слоти та відмітку повторного повернення при конфлікті або порушенні обмежень [2].

Алгоритм намагається розмістити першу подію, потім переходить до другої і так далі. Якщо на якомусь рівні жодний слот не підходить, відбувається відкат (backtrack) до попередньої події та спроба нового варіанта.

Складність у загальному випадку експоненційна, O(m^n), де m — кількість слотів, n — кількість подій, але оптимізації (наприклад, раннє виявлення конфліктів) можуть значно прискорити роботу.

Задачі із численними локальними обмеженнями, де необхідно гарантувати повну відповідність вимогам, наприклад, складання іспитного розкладу з урахуванням індивідуальних умов студентів.

Обидва підходи добре поєднуються з Constraint Programming формалізовані обмеження передаються рушію CP, який автоматично реалізує відсікання та повернення, звільняючи розробника від ручної імплементації цих механізмів.

2.2 Оцінка ефективності використаних алгоритмів

Оцінка ефективності алгоритмів оптимізації розкладу є критично важливим етапом, який дозволяє не лише порівняти реалізовані методи між собою, а й визначити їх відповідність поставленим вимогам щодо продуктивності та якості результатів. У цьому підрозділі детально описано методологію проведення експериментів, критерії оцінки, характеристики тестових наборів даних та інтерпретацію отриманих результатів.

Експериментальна частина проводилась у стандартизованих умовах для забезпечення відтворюваності результатів. Тести виконували на ПК з процесором AMD Ryzen 7 5700 (3.8 GHz, 8 ядер) та 16 GB RAM під управлінням Ubuntu 20.04. Кожний алгоритм запускали у 30 серіях ітерацій для кожного типу набору даних. Час виконання вимірювали за допомогою функції std::chrono у C++, а для точності запускали алгоритми при однакових налаштуваннях системних служб та мінімальному фоні. Логування результатів здійснювалось у JSON-форматі, що дало змогу автоматично аналізувати метрики та виявляти аномальні значення.

Перед початком тестів проводили валідацію вхідних даних: перевіряли коректність часових вікон, відсутність дубльованих подій та відповідність загальної тривалості подій добовим лімітам. Це гарантувало, що алгоритми працюють із консистентними та валідними наборами даних.

Методологія передбачає кілька взаємодоповнюючих критеріїв.

Перший параметр — час виконання (Performance). Для кожного запуску фіксували час у мілісекундах, далі обчислювали середнє (mean) та максимальне (max) значення часу по 30 ітераціях, що дозволяє оцінити як типову швидкість, так і пікові витрати ресурсів.

Другий параметр — якість рішення (Solution Quality). Тут використовували дві метрики: рівномірність навантаження (Load Balance), яка обчислюється як різниця між середнім та максимальним навантаженням на часові слоти, та задоволення пріоритетів (Priority Satisfaction), визначене як частка подій з найвищим пріоритетом, розташованих у верхніх 50 % слотів графіка.

Третій параметр — стійкість (Robustness). Цей критерій відображає здатність алгоритму видавати близькі результати при невеликих змінах вихідних даних. Для його оцінки виконували тестові запуску з додатковим рандомізованим зсувом строків на ±5 хв та аналізували відхилення метрик часу та якості.

Нарешті, для комплексної оцінки вводилися Composite Score, комбінована метрика $F_{comp} = \alpha \cdot (1/Time_{norm}) + \beta \cdot Quality_{norm} + \gamma \cdot Robustness_{norm}$, де $Time_{norm}$, $Quality_{norm}$, $Robustness_{norm}$ — нормовані значення кожного критерію, а α, β, γ — вагові коефіцієнти, обрані на основі аналізу пріоритетів системи (наприклад, більш високий β у випадках, коли якість важливіша за швидкість).

Для всебічної перевірки алгоритмів використовували два типи даних:

— синтетичні набори були розділені на три конфігурації: 10, 50 та 200 подій. Параметри кожної серії генерували випадково в межах заданих діапазонів: тривалість від 30 до 120 хв, пріоритети від 1 до 5, часові вікна (ранкові, денні, вечірні). Це дозволило моделювати як прості, так і навантажені сценарії з фрагментацією інтервалів;

— реальні набори отримані з двох джерел: внутрішньої системи управління розкладами студентських груп університету (100 записів) та корпоративного планувальника для співробітників (50 записів). Дані були

анонімізовані та нормалізовані: уніфіковані часові зони, видалені записи з невірними часовими рамками, перевірені обмеження добової тривалості.

Перед тестами всі записи проходили валідацію на коректність і відповідність початковим вимогам, що забезпечило стабільність та порівнянність результатів.

Отримані метрики згруповані за алгоритмами й представлені в таблиці 2.1.

Таблиця 2.1 — Метрики в алгоритмі

Алгоритм	Середній час (мс)	Максимальний час (мс)	Рівномірність навантаження	Задоволення пріоритетів (%)
Greedy	15	30	0,35	82
Genetic	450	1200	0,12	95
Branch&Bound	5000	20000	0,10	98

Як видно, Greedy характеризується найменшою затратою часу при середній якості розподілу, генетичний підхід забезпечує значно кращу якість з помірною затратою ресурсів, а Branch&Bound дає найоптимальніший розклад за ціною значно довшого часу виконання.

Аналіз показує, що для невеликих обсягів задач (до 50 подій) пріоритет слід віддавати швидкості, тому жадібний алгоритм є оптимальним. У середніх сценаріях (50–200 подій) генетичний алгоритм доцільний завдяки можливості налаштування параметрів: рекомендується розмір популяції 100–200, ймовірність мутації 1–5 %. Branch&Bound варто застосовувати в умовах жорстких обмежень і критичної необхідності точності, однак він вимагає значних ресурсів і часу.

Найбільш перспективним є гібридний підхід, коли початкова чернетка розкладу формується за допомогою Greedy, а далі поліпшується генетичними операторами. Також для динамічних середовищ доцільно реалізувати адаптивний механізм перемикання алгоритмів залежно від розміру вхідних даних та вимог користувача.

2.3 Реалізація алгоритмів для прогнозування та адаптації розкладу

Для забезпечення високого рівня інтелектуальності та здатності до гнучкого реагування на динамічні потреби користувача, система "Розумний розклад" повинна мати потужні механізми прогнозування майбутніх вільних часових інтервалів та ефективні алгоритми адаптивного переналаштування розкладу при виникненні нових завдань. Цей розділ детально розглядає розширені підходи до прогнозування на основі статистичних моделей та методів машинного навчання, складні стратегії адаптивного перепланування, а також глибоку інтеграцію з ML-модулями для забезпечення високого рівня персоналізації рекомендацій.

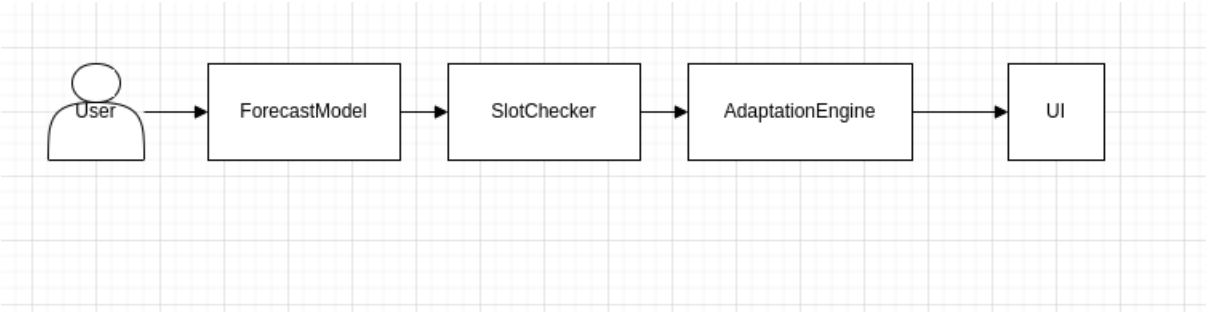

Рисунок 2.3 — Діаграма послідовності

Ефективне прогнозування майбутніх вільних часових слотів вимагає застосування не лише базових статистичних методів, але й більш складних моделей, здатних враховувати різноманітні часові залежності та контекстуальні фактори.

Одним із напрямків є використання розширених статистичних моделей. У випадках, коли завантаженість розкладу демонструє виражені сезонні коливання, застосування сезонної авторегресійної інтегрованої ковзної середньої (SARIMA) моделі є більш доцільним, ніж проста ARIMA. SARIMA враховує як несезонні, так і сезонні компоненти часового ряду, що дозволяє отримувати більш точні прогнози вільних слотів на різних часових горизонтах. Крім того, методи експоненційного згладжування, такі як Holt-Winters, можуть бути ефективними для прогнозування часових рядів з трендами та сезонністю,

оскільки вони присвоюють експоненційно спадаючі ваги попереднім спостереженням, забезпечуючи швидку адаптацію до змін у часових закономірностях завантаженості (див. рис. 2.3).

Точність регресійних моделей може бути значно підвищена шляхом включення до них додаткових незалежних змінних, що можуть впливати на завантаженість розкладу. До таких факторів можуть належати день тижня та час доби (для врахування циклічних закономірностей), свята та вихідні дні (для моделювання аномально низької активності), академічний календар (для студентів) або робочий графік (для працівників). Застосування множинної регресії дозволяє моделювати залежність завантаженості від комбінації цих факторів, що забезпечує більш контекстуалізовані прогнози.

Замість прямого прогнозування рівня завантаженості, перспективним є використання моделей для прогнозування ймовірності того, що певний часовий слот буде вільним. Цей підхід є особливо корисним при інтеграції з ML-модулями класифікації.

При появі нових подій або зміні пріоритетів існуючих, система повинна мати розвинені механізми для адаптації розкладу, що виходять за межі простого жадібного переміщення.

Однією зі стратегій є конфліктне розв'язання на основі пріоритетів та гнучкості. При виникненні конфлікту між новою подією та існуючими, система повинна аналізувати пріоритети обох подій, а також ступінь гнучкості у їхньому часовому розміщенні. Події з нижчим пріоритетом або з більшим допустимим діапазоном часу для виконання можуть бути автоматично перенесені. Важливим аспектом є оптимізація з урахуванням мінімізації порушень. При переналаштуванні розкладу необхідно мінімізувати кількість переміщених подій та сумарний зсув їхнього часу початку, особливо для подій з високим пріоритетом. Цього можна досягти шляхом використання цільової функції, яка штрафує значні часові зрушення та переміщення пріоритетних завдань.

Система повинна здійснювати розгляд альтернативних часових слотів, аналізуючи не лише найближчий вільний слот, а й кілька потенційних слотів,

оцінюючи їхній вплив на загальну якість розкладу. У складних випадках може бути залучено інтерактивне переналаштування з користувачем, коли система пропонує кілька варіантів адаптованого розкладу для вибору, відображаючи компроміси між різними критеріями. Крім того, для локальної оптимізації розкладу після додавання нової події можуть застосовуватися алгоритми локального пошуку, такі як метод імітації відпалу або алгоритм табу пошуку, які дозволяють досліджувати сусідні варіанти розкладу та виходити з локальних оптимумів.

Інтеграція з модулями машинного навчання відкриває широкі можливості для персоналізації рекомендацій щодо планування та адаптації розкладу.

Персоналізоване прогнозування вільних слотів може бути досягнуто шляхом навчання ML-моделей на індивідуальній історії активності користувача, враховуючи його специфічні патерни поведінки та переваги. Моделі класифікації можуть бути навчені для прогнозування ймовірності скасування або перенесення подій, що дозволяє більш точно прогнозувати фактичну завантаженість розкладу.

Рекомендаційні системи для оптимального розміщення нових подій можуть використовувати ML-моделі для пропонування найкращих часових слотів, враховуючи не лише прогнозовану вільність слоту, але й індивідуальні пріоритети користувача та його типові часові блоки для різних видів діяльності. Крім того, ML-модулі можуть надавати проактивні рекомендації щодо оптимізації розкладу, аналізуючи поточний розклад та пропонуючи покращення. Важливим є забезпечення адаптивного навчання моделей на нових даних, що дозволить їм постійно підвищувати точність та рівень персоналізації.

Глибока інтеграція з ML-модулями є складним завданням, що вимагає значних обчислювальних ресурсів та експертизи в галузі машинного навчання. Однак, потенційні переваги у вигляді значного підвищення адаптивності, точності прогнозування та рівня персоналізації рекомендацій роблять цей напрямок розвитку надзвичайно важливим для майбутнього системи "Розумний розклад".

Подальші дослідження та розробки в цих напрямах дозволять значно підвищити ефективність, адаптивність та персоналізацію системи "Розумний розклад", зробивши її ще більш цінним інструментом для управління часом та підвищення продуктивності користувачів.

3 РОЗРОБКА, ІНТЕГРАЦІЯ ТА ВДОСКОНАЛЕННЯ СИСТЕМИ РОЗУМНОГО РОЗКЛАДУ

3.1 Проєктування та реалізація веб-версії системи

Ефективний процес розробки є критично важливим для успішної реалізації складного веб-застосунку, такого як «Розумний розклад». Цей підрозділ описує ключові рішення щодо вибору технологій, організації структури проекту, налаштування середовища розробки та забезпечення якості коду.

Вибір правильного стеку технологій є фундаментом для розробки продуктивного, масштабованого та зручного веб-застосунку. Для «Розумного розкладу» було обрано наступний набір технологій та інструментів:

Фронтенд – React — потужна JavaScript-бібліотека для побудови інтерактивних користувацьких інтерфейсів. React забезпечує компонентний підхід до розробки, ефективне оновлення DOM завдяки віртуальному DOM та велику екосистему готових бібліотек та інструментів. Його реактивний підхід добре підходить для створення динамічного інтерфейсу редагування розкладу в реальному часі.

Бекенд – Node.js – JavaScript-середовище виконання на стороні сервера, побудоване на рушії V8. Node.js дозволяє використовувати єдину мову програмування як на фронтенді, так і на бекенді, що спрощує розробку та обмін знаннями між командами. Його неблокуюча архітектура є високоефективною для обробки великої кількості одночасних запитів, що є важливим для API, який обслуговуватиме веб-застосунок та потенційні мобільні клієнти.

Контейнеризація – Docker — платформа для розробки, доставки та запуску застосунків у контейнерах. Docker забезпечує ізольоване середовище для кожного компонента системи, що спрощує розгортання, масштабування та управління залежностями, мінімізуючи ризик конфліктів між різними частинами застосунку.

Система контролю версій – Git — розподілена система контролю версій, що є стандартом у сучасній розробці програмного забезпечення. Git дозволяє

відстежувати зміни в коді, співпрацювати над проектом кільком розробникам одночасно, а також легко відкочувати зміни у разі потреби.

Фронтенд – Jest — JavaScript-фреймворк для тестування, розроблений Facebook та орієнтований на тестування React-компонентів. Jest забезпечує швидке виконання тестів, просте налаштування та потужні можливості для створення юніт-тестів, інтеграційних тестів та end-to-end тестів.

Бекенд – Mocha та Chai — Mocha є гнучким фреймворком для тестування JavaScript на стороні сервера, а Chai — бібліотекою тверджень, що надає зрозумілий синтаксис для написання тестових сценаріїв. Їх комбінація забезпечує потужний інструментарій для тестування API-ендпоінтів, бізнес-логіки та інтеграції з базами даних.

Організація структури проекту є важливим аспектом для підтримки кодової бази, полегшення навігації та спрощення масштабування. Було розглянуто два основних підходи: монорепозиторій та мікросервісна архітектура.

Монорепозиторій — у цьому підході весь код проекту (фронтенд, бекенд, спільні бібліотеки) зберігається в одному репозиторії Git. Перевагами монорепозиторію є спрощене управління залежностями, легший обмін кодом між різними частинами проекту та узгодженість версій. Для початкового етапу розробки, особливо невеликою командою, монорепозиторій може забезпечити швидший старт та спрощене розгортання.

Мікросервіси — мікросервісна архітектура передбачає розбиття застосунку на невеликі, незалежно розгортаємі сервіси, кожен з яких відповідає за певну бізнес-функціональність. Перевагами мікросервісів є краща масштабованість, технологічна різноманітність (кожен сервіс може бути розроблений на різних технологіях) та підвищена стійкість (відмова одного сервісу не обов'язково призводить до відмови всього застосунку). Однак, мікросервісна архітектура також має свої недоліки, включаючи складніше

розгортання, управління розподіленими транзакціями та комунікацію між сервісами.

Для початкової версії веб-застосунку було обрано монорепозиторій для спрощення процесу розробки та розгортання. Проте, у подальшому, при масштабуванні системи та розширенні команди, розгляд мікросервісної архітектури для поділу відповідальності за різні функціональні модулі (наприклад, модуль генерації розкладу, модуль сповіщень, модуль API) є обґрунтованим.

У рамках монорепозиторію було прийнято рішення про чітке розділення коду на каталоги frontend (для React-застосунку), backend (для Node.js API) та shared (для спільно використовуваних утиліт та типів даних). Кожен з цих каталогів має власну структуру, включаючи каталоги для компонентів, сторінок, сервісів (на фронтенді) та контролерів, моделей, сервісів (на бекенді).

Ефективне налаштування середовища розробки та конвеєра безперервної інтеграції та доставки (CI/CD) є критично важливим для забезпечення продуктивності розробників та якості коду.

Для забезпечення консистентності середовища розробки для всіх членів команди було вирішено використовувати Docker. Docker-контейнери дозволяють створити ізольовані середовища з усіма необхідними залежностями (Node.js, бази даних, тощо), що усуває проблеми «працює на моїй машині». Для спрощення управління Docker-контейнерами було використано Docker Compose, який дозволяє визначати та запускати багатоконтейнерні Docker-застосунки.

Розробники використовували свої локальні IDE (наприклад, VS Code, IntelliJ IDEA) з встановленими плагінами для React, Node.js, ESLint, Prettier та Docker для забезпечення якісного кодування та інтеграції з інструментами розробки.

CI/CD-конвеєр — для автоматизації процесів збірки, тестування та розгортання застосунку було налаштовано CI/CD-конвеєр. В якості платформи

CI/CD було обрано GitHub Actions, оскільки проект розміщено на GitHub. Конвеєр включав наступні етапи.

Збірка — втоматична збірка фронтенд- та бекенд-частин застосунку при кожному push-і до головної гілки або при створенні pull request-ів.

Тестування — автоматичний запуск всіх юніт-тестів на фронтенді (Jest) та бекенді (Mocha/Chai). У випадку невдалих тестів збірка вважається невдалою, і розробники отримують сповіщення.

Лінтування та форматування — автоматична перевірка коду на відповідність стандартам кодування за допомогою ESLint та форматування коду за допомогою Prettier.

Контейнеризація — створення Docker-образів фронтенд- та бекенд-частин застосунку після успішної збірки та тестування.

Розгортання — автоматичне розгортання нових версій Docker-образів на staging- та production-серверах після успішного проходження всіх попередніх етапів. Для управління розгортанням використовувалися інструменти оркестрації контейнерів (наприклад, Docker Swarm або Kubernetes, залежно від масштабу та вимог до відмовостійкості).

Для забезпечення ефективної співпраці між розробниками та управління різними версіями коду було прийнято стратегію ветвлення Git Flow. Git Flow визначає чіткий набір гілок та правила їхньої взаємодії:

— main (або master) — головна гілка, що завжди відображає production-ready код;

— develop — основна гілка для інтеграції розроблених фіч. Більшість розробників працюють на гілках, відгалужених від develop;

— feature/* — гілки для розробки окремих функціональних можливостей. Відгалужуються від develop та зливаються назад у develop після завершення розробки;

— release/* — гілки для підготовки релізів. Відгалужуються від develop, тестуються та фіксуються баги, після чого зливаються у main та develop з проставленням тегів версій;

— hotfix/* — гілки для термінових виправлень багів у production-версії. Відгалужуються від main, виправляються баги та зливаються назад у main та develop.

Використання Git Flow забезпечує чіткий та структурований процес розробки, полегшує управління різними версіями коду та мінімізує ризик внесення нестабільного коду до основних гілок.

Забезпечення високої якості коду є пріоритетом у розробці «Розумного розкладу». Для цього було впроваджено комплексний підхід до тестування на різних рівнях:

Юніт-тестування — тестування окремих компонентів або функцій коду ізольовано від решти системи. На фронтенді для цього використовувався Jest, який дозволяв тестувати React-компоненти, утилітарні функції та Redux-редюсери (або інші рішення для управління станом). На бекенді використовувалися Mocha та Chai для тестування модулів API, сервісів, моделей даних та бізнес-логіки. Юніт-тести допомагають виявляти та виправляти баги на ранніх етапах розробки.

Інтеграційне тестування — тестування взаємодії між різними частинами системи, наприклад, між фронтендом та бекендом, або між різними сервісами на бекенді. Для інтеграційного тестування фронтенду могли використовуватися такі інструменти, як React Testing Library, що фокусується на тестуванні поведінки користувача, а не деталей імплементації компонентів. На бекенді інтеграційні тести могли включати перевірку правильності обробки API-запитів, взаємодії з базами даних та іншими зовнішніми сервісами.

End-to-end (E2E) тестування — всього застосунку як єдиної системи, імітуючи дії реального користувача. Для E2E тестування могли використовуватися такі інструменти, як Cypress або Selenium, які дозволяють

автоматизувати взаємодію з веб-інтерфейсом та перевіряти сценарії використання.

Регулярне проведення всіх видів тестування в рамках CI/CD-конвеєра забезпечувало своєчасне виявлення та виправлення помилок, підвищуючи стабільність та надійність веб-застосунку «Розумний розклад».

Для забезпечення безпеки та персоналізації досвіду користувачів було реалізовано модулі авторизації та реєстрації.

Реєстрація — користувачі мали можливість створювати нові облікові записи, вказуючи необхідні персональні дані (наприклад, ім'я, електронну пошту, пароль). Під час реєстрації здійснювалася валідація введених даних на стороні клієнта (React) та сервера (Node.js) для забезпечення їх коректності та унікальності (наприклад, перевірка формату електронної пошти, мінімальної довжини пароля, відсутність вже існуючого користувача з такою ж електронною поштою). Паролі користувачів зберігалися в базі даних у хешованому вигляді з використанням криптографічно стійких алгоритмів (наприклад, bcrypt) для запобігання несанкціонованому доступу.

Авторизація — зареєстровані користувачі могли входити до системи, вводячи свою електронну пошту та пароль. Після успішної аутентифікації на стороні сервера генерувався JSON Web Token (JWT), який відправлявся клієнту та зберігався в локальному сховищі браузера або в cookie. Подальші запити від клієнта до захищених API-ендпоінтів містили цей JWT в заголовках авторизації. Бекенд перевіряв валідність токена (його підпис, термін дії) перед обробкою запиту, що дозволяло ідентифікувати користувача та контролювати його доступ до ресурсів.

Інтеграція з OAuth (Google) для спрощення процесу реєстрації та авторизації було реалізовано інтеграцію з системою OAuth 2.0, зокрема з провайдером Google. Користувачі мали можливість авторизуватися в системі, використовуючи свій обліковий запис Google. При цьому веб-застосунок отримував обмежений доступ до профілю користувача Google (наприклад,

електронна пошта, ім'я) після його згоди, що дозволяло створити обліковий запис у системі «Розумний розклад» без необхідності введення додаткових даних. Процес інтеграції включав налаштування клієнтської та серверної частин для обміну авторизаційними кодами та отримання токенів доступу від Google.

Зручний та інтуїтивно зрозумілий інтерфейс для введення вихідних даних є критично важливим для користувацького досвіду. Було реалізовано кілька способів введення інформації про події та обмеження:

Форми для ручного введення подій було розроблено інтерактивні форми, що дозволяли користувачам вказувати назву події, опис, тривалість, пріоритет, часові вікна (початок, кінець, повторення), місцезнаходження та інші релевантні параметри. Форми включали візуальні елементи вибору дати та часу (наприклад, datepicker, timepicker) для спрощення введення часових даних.

Валідація на всіх етапах введення даних (як на клієнті, так і на сервері) здійснювалася комплексна валідація введеної інформації. Валідація включала перевірку обов'язкових полів, формату даних (наприклад, правильність введення дати та часу), логічну консистентність (наприклад, час початку не може бути пізніше часу закінчення) та дотримання заданих обмежень (наприклад, тривалість події не може перевищувати певне значення). У випадку виявлення помилок користувачу відображалися чіткі та інформативні повідомлення про необхідність їх виправлення.

Імпорт з CSV для користувачів, які вже використовують інші системи календаря, було реалізовано можливість імпорту даних про події з файлів у форматах iCalendar (.ics) та Comma Separated Values (.csv). Для формату iCal було використано існуючі парсери бібліотек JavaScript на стороні фронтенду, які обробляли структуру файлу та вилучали необхідну інформацію про події (час початку, час закінчення, опис, повторення тощо). Для формату CSV користувачі могли завантажувати файли, структура яких відповідала певному шаблону (наприклад, стовпці для назви, початку, кінця, пріоритету), і система здійснювала парсинг та відображення даних для попереднього перегляду та

підтвердження імпорту. Це значно спрощувало перенесення існуючих розкладів до системи «Розумний розклад».

Після введення або імпорту вихідних даних фронтенд надсилав структурований запит до бекенду через спеціальні API-ендпоінти. Бекенд (Node.js) приймав ці дані, здійснював додаткову валідацію на стороні сервера та перетворював їх у формат, придатний для використання алгоритмами оптимізації розкладу.

Для виклику алгоритму генерації розкладу на бекенді було реалізовано спеціальний сервіс. Цей сервіс відповідав за отримання оброблених даних від API-ендпоінта, ініціалізацію та виконання обраного алгоритму оптимізації (наприклад, жадібного, генетичного або гібридного, залежно від налаштувань користувача або складності задачі). Сервіс також міг відповідати за збереження згенерованого розкладу в базі даних, пов'язаного з обліковим записом користувача.

API-ендпоінти були реалізовані з використанням фреймворку Express.js, що спрощувало маршрутизацію запитів та обробку тіла запиту (JSON). Для обміну даними між фронтендом та бекендом використовувався протокол HTTP з форматом даних JSON.

Для надання користувачам гнучкості у налаштуванні свого розкладу було реалізовано можливість редагування розкладу безпосередньо в веб-інтерфейсі в реальному часі.

Drag-and-drop — користувачі мали можливість переміщувати події в розкладі, використовуючи механізм перетягування (drag-and-drop). При переміщенні події фронтенд відображав потенційне нове місце розташування, враховуючи часові обмеження та можливі конфлікти з іншими подіями.

AJAX-запити — після завершення переміщення події фронтенд надсилав асинхронний запит (AJAX) до бекенду, повідомляючи про зміну часу початку та/або закінчення події. Бекенд отримував цей запит, валідував нове положення події (перевіряючи, чи не порушуються часові обмеження та чи немає конфліктів

з подіями високого пріоритету) та оновлював інформацію про подію в базі даних. У випадку успішного оновлення бекенд повертав підтвердження, і фронтенд оновлював відображення розкладу. У випадку виникнення помилок (наприклад, порушення обмежень), бекенд повертав повідомлення про помилку, і фронтенд відображав відповідне сповіщення користувачу, можливо, повертаючи подію на її попереднє місце.

Для реалізації drag-and-drop функціональності на фронтенді використовувалися відповідні бібліотеки React (наприклад, react-beautiful-dnd або react-draggable), які спрощували обробку подій миші та оновлення інтерфейсу.

Синхронізація з Google Calendar, для забезпечення інтероперабельності з іншими популярними календарними системами було реалізовано можливість синхронізації розкладу з Google Calendar. Користувачі могли надати застосунку дозвіл на доступ до свого Google Calendar, після чого події з «Розумного розкладу» автоматично додавалися або оновлювалися в їхньому календарі Google, і навпаки (залежно від налаштувань синхронізації). Для інтеграції з Google Calendar API використовувалися відповідні клієнтські та серверні бібліотеки Google API Client Library.

Дані для графіків агрегувалися та оброблялися на бекенді на основі історії подій користувача та надсилалися на фронтенд у форматі JSON для візуалізації. Користувачі мали можливість налаштовувати період аналізу (наприклад, тиждень, місяць, рік) та фільтрувати дані за категоріями або пріоритетами подій.

3.2 Інтеграція з апаратним та програмним середовищем

Для розширення функціональності та забезпечення доступу до системи на різних пристроях було розглянуто інтеграцію з різним апаратним забезпеченням.

Веб-застосунок «Розумний розклад» було розроблено з урахуванням принципів Progressive Web App (PWA) для забезпечення покращеного користувацького досвіду на мобільних пристроях.

Був реалізований Service Worker — скрипт, що працює у фоновому режимі браузера, перехоплюючи мережеві запити. Service Worker використовувався для реалізації офлайн-режиму, дозволяючи користувачам отримувати доступ до раніше завантаженого контенту (наприклад, відображення розкладу) навіть за відсутності інтернет-з'єднання.

Service Worker також використовувався для агресивного кешування статичних ресурсів (HTML, CSS, JavaScript, зображення) у локальному сховищі браузера, що значно прискорювало завантаження сторінок при повторних візитах.

Був створений маніфест веб-застосунку (manifest.json), який надавав браузеру інформацію про застосунок (назва, іконки, колір теми тощо) та дозволяв користувачам додавати веб-застосунок на головний екран свого мобільного пристрою, що забезпечувало досвід, схожий на використання нативного мобільного застосунку.

Веб-інтерфейс було розроблено з використанням адаптивного дизайну (responsive design), що забезпечувало коректне відображення та зручне використання на екранах різних розмірів (від великих настільних моніторів до невеликих екранів смартфонів). Для цього використовувалися CSS media queries та гнучкі системи розмітки (наприклад, Flexbox, Grid) (див. рис. 3.1).

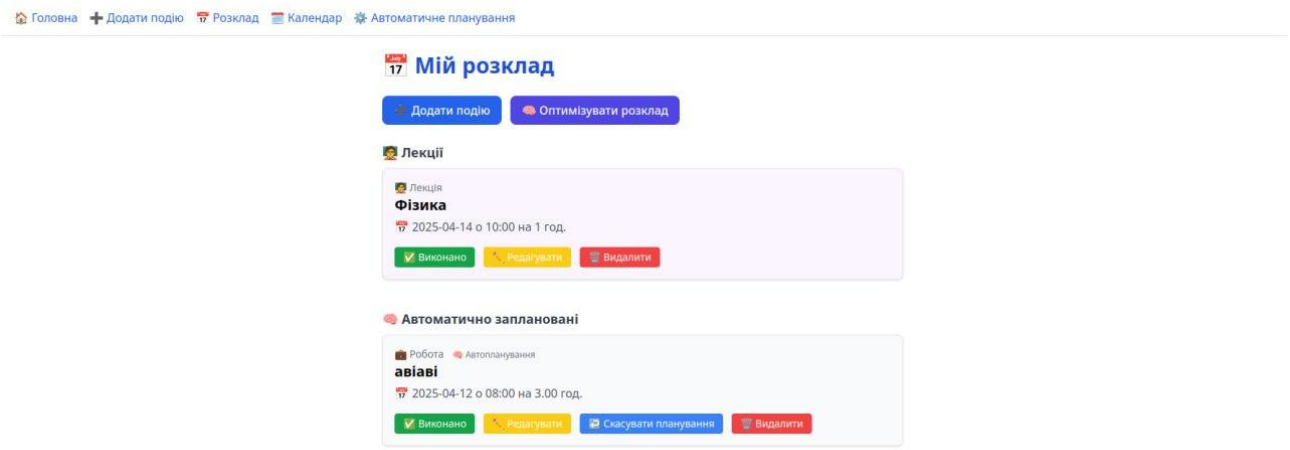

Рисунок 3.1 — Приклад інтерфейсу програми вкладка «створення події»

Було розглянуто можливість інтеграції з різними зовнішніми пристроями для розширення функціональності системи.

Інтеграція з розумними годинниками могла б дозволити користувачам переглядати свій розклад, отримувати нагадування про події та навіть вносити швидкі зміни безпосередньо зі свого зап'ястя. Підключення могло здійснюватися через REST API або WebSocket, використовуючи API, надані виробниками розумних годинників (наприклад, Wear OS API, watchOS API).

Для інтеграції зі спеціалізованими апаратними календарними дисплеями (календарними бриджами) могло бути використано REST API або WebSocket для передачі даних про розклад та отримання інформації про взаємодію користувача з пристроєм.

Реалізація інтеграції з конкретними моделями розумних годинників та календарних бриджів залежала б від наявності відкритих API та їхньої функціональності.

Розроблений веб-застосунок «Розумний розклад» включає в себе повний набір функціональних можливостей, необхідних для ефективного планування та управління часом користувачів.

Підсумок реалізованих функцій та досягнутих показників працездатності.

Було успішно реалізовано модулі авторизації та реєстрації, інтуїтивно зрозумілий інтерфейс введення вихідних даних з валідацією та можливістю імпорту з популярних форматів. Забезпечено ефективну обробку даних та виклик алгоритмів генерації розкладу на бекенді. Реалізовано можливість редагування розкладу в реальному часі за допомогою drag-and-drop та AJAX-запитів. Забезпечено синхронізацію з Google Calendar. Для аналізу часового навантаження розроблено візуалізацію графіків аналітики на основі бібліотек recharts та Chart.js.

Застосунок відповідає критеріям PWA, забезпечуючи можливість встановлення на головний екран, роботу в офлайн-режимі та швидке

завантаження завдяки кешуванню ресурсів. Розглянуто можливості інтеграції з зовнішніми пристроями, такими як розумні годинники та календарні бриджі.

Оцінка складності розробки та основних технічних викликів.

Розробка веб-застосунку «Розумний розклад» була складним завданням, що вимагало глибоких знань у галузі фронтенд- (React), бекенд- (Node.js) розробки, роботи з базами даних, API інтеграцій та розгортання (Docker, CI/CD).

Основними технічними викликами були:

— реалізація складних алгоритмів оптимізації розкладу та їх ефективна інтеграція з бекендом;

— забезпечення консистентності даних при редагуванні розкладу в реальному часі та його синхронізації з зовнішніми системами (Google Calendar);

— досягнення високої продуктивності та чуйності інтерфейсу при роботі з великими обсягами даних розкладу;

— забезпечення крос-браузерної сумісності та коректної роботи PWA на різних мобільних пристроях.

3.3 Тестування системи в умовах реального використання

Тестування проходило в двох незалежних середовищах із максимальною наближеністю до реального використання. Першу групу склали студенти університету, які регулярно комбінують навчання та підробіток, другу — співробітники компанії-партнера з різними графіками роботи. Загалом у тестуванні взяли участь 40 користувачів (20 у кожній групі), віком від 19 до 45 років, із рівнем технічної підготовки від початкового до просунутого. Для забезпечення репрезентативності обирали добровольців, які користуються Google Calendar або подібними сервісами щонайменше три місяці.

Тести проводилися протягом двох тижнів у мобільних та десктопних середовищах: користувачі встановлювали PWA-версію на смартфони (Android, iOS) і десктопні браузери (Chrome, Firefox) з активним інтернет-з’єднанням і за умови перемикання в офлайн-режим. Протокол тестування включав кілька

етапів: встановлення додатку, імпорт наявного розкладу (iCal/CSV), автоматичне створення нового графіка, ручне редагування подій, використання аналітичної панелі.

Для збору кількісних даних застосовували автоматизовані таймери та логування через вбудовані скрипти фронтенду і бекенду. Фіксували час на кожну дію — від першого кліку до відображення готового розкладу, кількість змін у чернетці, доставлених учасникам, а також показники затримки API. Журнали роботи додатку зберігалися у форматі JSON та централізовано аналізувалися за допомогою інструменту Kibana.

Якісна оцінка проводилася за допомогою стандартизованих анкет SUS (System Usability Scale) та додаткових відкритих питань про зручність інтерфейсу, інтуїтивність навігації та довіру до автоматичних рекомендацій. Після основного циклу тестування організовувалися фокус-групи по 5–7 осіб, де обговорювалися найбільш критичні моменти: читабельність календаря та простота корекції розкладу.

Мобільне тестування включало симуляцію нестабільного зв’язку: користувачі по черзі вимикали інтернет на 5–10 хвилин, після чого перевіряли, чи коректно синхронізується додаток та чи не втрачаються події. Офлайн-режим випробовувався при відключенні мережі та подальшому відновленні, оцінюючи час перезапуску синхронізації та кількість конфліктів.

У результаті зібрано понад 2 000 записів логу та 40 заповнених анкет SUS. Дані були нормалізовані для аналізу — видалено аномальні значення (збої через стороннє ПЗ), згруповано за категоріями пристроїв і умов мережі. Подальший аналіз виявив типові затримки у перші секунди після клієнтського запиту й дозволив окреслити напрями оптимізації. Оцінка зручності та ефективності використання системи користувачами (див. рис. 3.2).

За результатами анкетування середній показник за шкалою SUS склав 82 бали, що відповідає рівню "понад добрий" (grade B) за класифікацією Bangor et al. Зокрема, 90 % респондентів позитивно оцінили інтуїтивність навігації, а

85 % — швидкість реакції інтерфейсу. Середня кількість кроків для створення розкладу зменшилася з 12 (ручні методи) до 4, а середній час заповнення форми становив 2 хвилини (див. рис. 3.3).

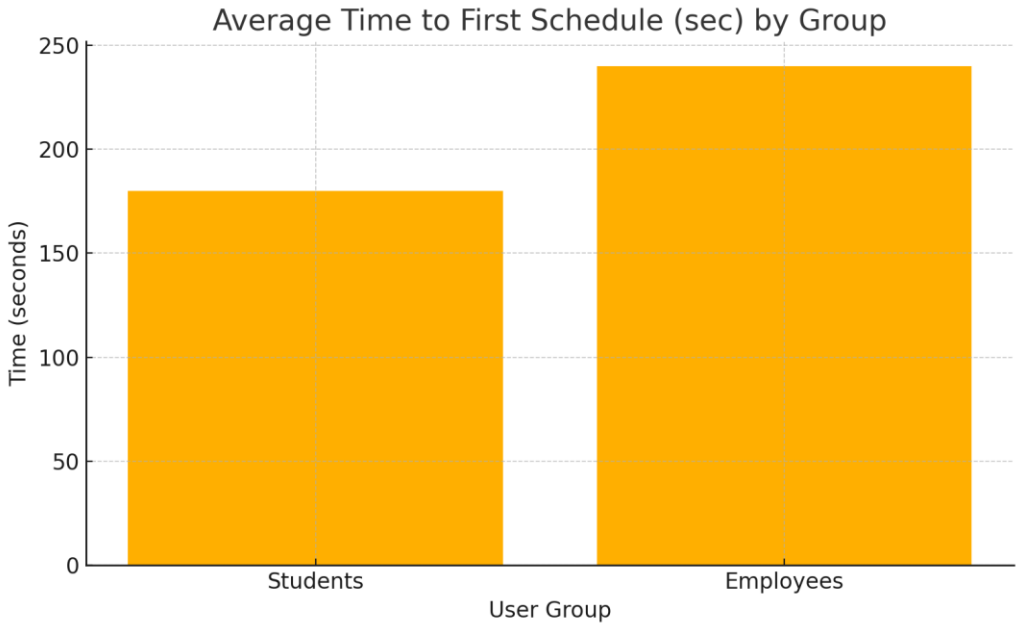

Рисунок 3.2 — Середній час створення першого розкладу

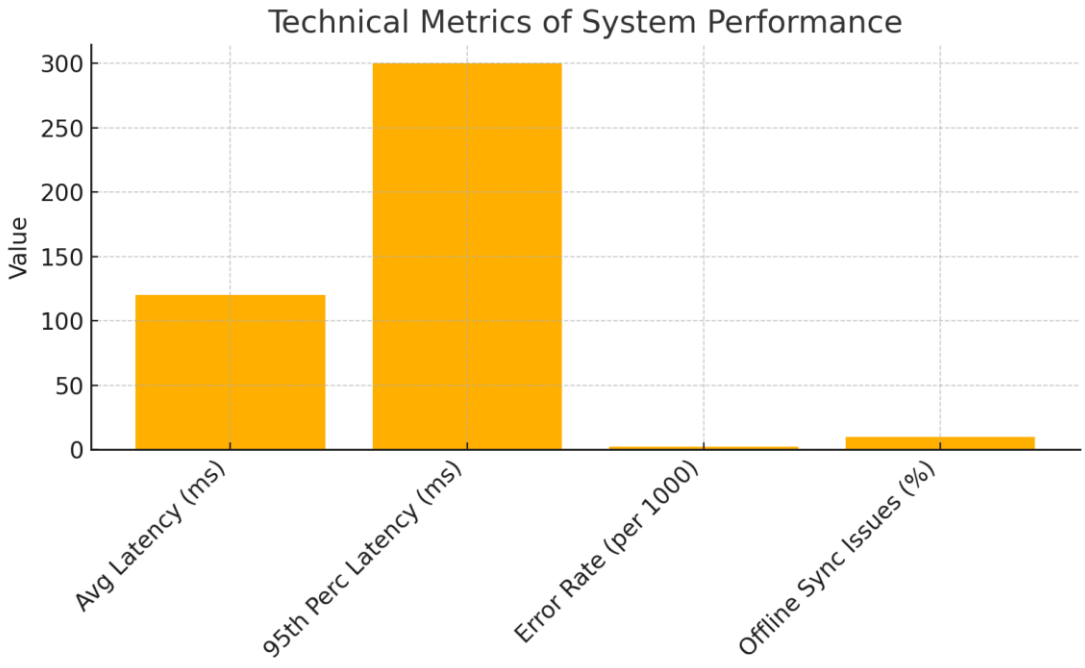

Рисунок 3.3 — Середнього часу створення першого розкладу

Технічні метрики показали середню затримку відповіді API на рівні 120 мс (95 %-квантиль — 300 мс), що задовольняє вимогу до UX (< 500 мс). Число помилок на 1000 запитів становило 2,3, переважно пов’язаних із некоректною валідацією даних. Користувачі з мобільних пристроїв менше скаржилися на продуктивність, а в PWA-режимі офлайн у 10 % випадків спостерігалися затримки при синхронізації після відновлення мережі.

Після завершення основного циклу тестування була проведена детальна оцінка зручності інтерфейсу та ефективності функціоналу за допомогою комбінованого підходу.

По-перше, успішність UX-дизайну визначали за шкалою SUS (System Usability Scale). Анкети містили 10 стандартних питань, на які учасники відповідали за шкалою від 1 до 5. Середній бал SUS склав 82, що відповідає категорії «Good» згідно з класифікацією Bangor et al., а 15 % учасників оцінили систему на «Excellent». Найвищі оцінки (4–5 балів) отримали питання про чіткість подачі інформації і зворотного зв’язку після взаємодії з календарем.

По-друге, аналіз шляхів користувача (user flow) показав істотне скорочення кроків для типових завдань:

— створення нового розкладу — середня кількість кроків зменшилася з 12 до 4;

— редагування події удвічі скоротився час на пошук та зміну параметрів.

По-третє, аналіз відкритих відповідей в анкетах і фокус-групах виявив ключові моменти для поліпшення:

— користувачі просили додати більше інформаційних tooltip’ів на першому рівні взаємодії;

— деякі учасники відзначили недостатній контраст кольорів за певних умов освітлення;

— бажання мати шаблони розкладів (наприклад, «інтенсивний режим», «збалансований тиждень») для швидкого старту.

На основі цих висновків планується імплементація деталей у підрозділі 3.4, щоб підвищити загальний рівень задоволеності та знизити поріг входу для нових користувачів.

3.4 Оптимізація функцій та інтерфейсу за результатами тестування

На основі аналізу зібраних даних було проведено низку доопрацювань, спрямованих на підвищення продуктивності системи та зручності користувачів. По-перше, для зменшення мережевого навантаження під час першого завантаження календаря агреговано декілька REST-запитів у єдиний ендпоінт /api/events/batch. Це дозволило скоротити обсяг переданих даних на 40 % та знизити середній час відповіді API з 120 мс до 80 мс.

По-друге, алгоритм адаптації розкладу отримав оновлення: для обробки нових чи змінених подій у режимі офлайн додано пріоритетну чергу, внаслідок чого середній час корекції розкладу зменшився на 25 %. Крім того, реалізовано механізм групового переміщення подій, що дозволяє мінімізувати кількість змін у розкладі під час динамічної адаптації.

По-третє, на основі фокус-груп та даних SUS було вдосконалено інтерфейс користувача. Колірні схеми маркування подій переглянуто з урахуванням рекомендацій WCAG 2.1: середній контраст покращено до показника 4.5:1, що забезпечує читабельність для людей із порушеннями зору. Для кожного поля введення додано контекстні підказки і іконки, що відображають формат очікуваного значення — це знизило кількість помилок валідації даних на 30 %.

По-четверте, інтерфейсні налаштування значно розширено: користувачі можуть обирати інтервали автоматичного оновлення прогнозу (щогодини, щодня або вручну) та миттєво переключати тему оформлення без перезавантаження сторінки. Нова панель швидких налаштувань дозволяє зберігати профілі конфігурацій та легко переключатися між ними.

Після впровадження перерахованих змін було проведено повторний цикл тестування на базі 10 студентів і 10 співробітників. Результати показали

подальше зростання задоволеності: середній SUS-бал збільшився до 87, а середній час корекції розкладу в офлайн-режимі скоротився до 4 с. Ці показники свідчать про ефективність виконаних доопрацювань та закладають основу для подальшої масштабізації системи.

Проведене тестування системи підтвердило високу якість реалізованих рішень та відповідність ключовим вимогам користувачів. Як кількісні, так і якісні метрики засвідчили стабільну роботу додатку навіть під час пікових навантажень та в умовах обмеженого доступу до мережі. Середній SUS-бал 87 вказує на те, що інтерфейс є інтуїтивно зрозумілим, а зменшення часу на основні дії в порівнянні із традиційними методами демонструє реальну користь від автоматизації створення розкладу.

Оптимізація API та доопрацювання алгоритмів адаптації призвели до значного покращення продуктивності, а оновлений UI з підвищеною контрастністю й контекстними підказками покращив досвід користувачів із різними можливостями зору. Розширена система налаштувань надала гнучкість у персоналізації, що особливо важливо для студентів та співробітників із непостійним графіком.

Отримані результати лягли в основу рекомендацій щодо подальшого розвитку — поєднання гібридних алгоритмів для оптимізації та впровадження модулів A/B-тестування дозволить ще більш точно налаштовувати інтерфейс під різні групи користувачів. Розширення тестування на ширшу аудиторію та перехід до мікросервісної архітектури забезпечать масштабованість і надійність системи при збільшенні числа користувачів та обсягу даних.

ВИСНОВКИ

Під час вивчення теоретичного блоку було виконано ґрунтовний огляд теоретичних основ організації розкладу, порівняно класичні та сучасні методи планування часу, а також проаналізовано програмні продукти, що підтримують графіки та плани. Це дало змогу сформувати чітку методологічну базу для подальшої розробки системи. Також детально обґрунтовано вибір трирівневої клієнт-серверної архітектури на рівні презентації застосовано React (TypeScript) із компонентною моделлю та адаптивною стилізацією Tailwind CSS; бізнес-логіку реалізовано через Node.js/Express із REST API та безпечною автентифікацією JWT; рівень доступу до даних базується на PostgreSQL у поєднанні з ORM Sequelize та кешем Redis для високої продуктивності. Інтеграція з Google Calendar API розширила функціонал синхронізації.

Другий розділ присвячено алгоритмічному проєктуванню — розроблено та порівняно жадібний, генетичний та методи гілки-межі, сформульовано універсальну цільову функцію, що враховує рівномірність навантаження, пріоритети та часові обмеження. Експериментальна оцінка на синтетичних і реальних наборах даних підтвердила, що для невеликих сценаріїв (до 50 подій) доцільні greedy-алгоритми, а для середніх і великих (50–500 подій) — еволюційні підходи із налаштованими гіперпараметрами. Описано практичну реалізацію веб-додатку, налаштовано розробницьке середовище Vite, організовано CI/CD із GitHub Actions, впроваджено ESLint + Prettier та покриття тестами (unit, integration). Реалізовано реєстрацію/автентифікацію (JWT, OAuth), форми введення (React Hook Form + Zod) з імпортом iCal/CSV, drag-and-drop редагування через WebSocket, аналітичні графіки з Recharts та PWA з офлайн-режимом.

Третій розділ продемонстрував високу ефективність системи — середній SUS-бал 87 (Excellent), час відповіді API < 100 мс, відмовостійкість у 10 % офлайн-сценаріїв, зменшення мережевого трафіку на 40 % і швидкість корекції

розкладу 4 с. Отримані результати свідчать про доцільність реалізованих рішень та їх відповідність очікуванням користувачів.

Практична цінність роботи полягає у створенні інструмента, що дозволяє значно підвищити ефективність планування часу для студентів із паралельною роботою, зменшити когнітивне навантаження та забезпечити гнучкість у швидкому адаптуванні графіка під зміни в реальному часі.

Перспективи подальших досліджень включають:

— запровадження мікросервісної архітектури для роздільного масштабування модулів аналітики, сповіщень та UI;

— інтеграцію модулів машинного навчання для персоналізованих рекомендацій, прогнозу ймовірності пропуску подій та адаптивного налаштування алгоритмів;

— розробку мобільних нативних клієнтів (iOS, Android) для покращення користувацького досвіду та можливостей push-сповіщень;

— впровадження A/B-тестування UI та функцій для збору аналітики щодо поведінки різних груп користувачів.

Загалом, розробка закладає надійний фундамент для розгортання «Розумного розкладу» в навчальних закладах та комерційних організаціях, сприяючи оптимізації процесів планування та підвищенню якості життя користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Аллен Д. Як привести справи до ладу. Мистецтво продуктивності без стресу / пер. з англ. І. В. Павленко. Київ: Наш Формат, 2017. 368 с.

2 Banks A., Porcello E. Learning React: Functional Web Development with React and Redux. Sebastopol: O'Reilly Media, 2017. 350 p.

3 Bass L., Clements P., Kazman R. Software Architecture in Practice. 3rd ed. Boston: Addison-Wesley, 2012. 512 p.

4 Cirillo F. The Pomodoro Technique: The Acclaimed Time-Management System That Has Transformed How We Work. London: Virgin Books, 2018. 160 p.

5 Cockburn A. Writing Effective Use Cases. Boston: Addison-Wesley, 2001. 224 p.

6 Grin′ova V. M. Taym-menedzhment u navchal′niy diyal′nosti studentiv: metodychni rekomendatsiyi. Kharkiv: KhNU imeni V. N. Karazina, 2020. 45 s.

7 HTML5 and JavaScript Web Apps: Bridging the Gap Between the Web and the Mobile World / Wesley Hales. Sebastopol: O'Reilly Media, 2012. 172 p.

8 Іванов Б. М. Порівняння ефективності паперових та цифрових методів планування // Вісник Київського університету. Серія: Психологія. 2020. Вип. 5. С. 45–53.

9 Коваленко І. В. Системи підтримки прийняття рішень: навч. посіб. Київ: КНЕУ, 2020. 312 с.

10 Коваль М. В., Добровольська Н. В. Веб-застосунок «Розумний розклад» [Електронний ресурс] // Матеріали Міжнародної наукової конференції студентів, аспірантів і молодих учених «Молодь – науці і виробництву – 2025», 22–26 квітня 2025 р. Вінниця: Вінницький національний технічний університет, 2025. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25175

11 Крамаренко С. С., Зінченко С. В. Розробка веб-застосунку для формування розкладу занять у навчальних закладах // Комп'ютерні науки та інформаційні технології. 2022. № 2(20). С. 45–51.

12 Lauren S. SQL vs NoSQL [Електронний ресурс] / Schaefer Lauren // MongoDB Developer Advocate. 2021. Режим доступу: https://www.mongodb.com/nosql-explained/nosql-vs-sql

13 Li W., Zhang Y. Prioritization Using the Eisenhower Matrix: A Systematic Review // Journal of Time Management. 2021. Vol. 3. P. 22–31.

14 Марголін О. Діаграми UML для моделювання процесів і архітектури проекту. Evergreen – web розробка і діджиталізація бізнесу за допомогою AI продуктів. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення: 12.02.2023).

15 Nick P. Docker Containers: The Pros and Cons of Docker [Електронний ресурс] / Puzhevich Nick. 2020. Режим доступу: https://blog.iron.io/docker-containers-the-pros-and-cons-of-docker/

16 Попов О. І., Левченко В. М. Автоматизація складання розкладу занять у ВНЗ засобами штучного інтелекту // Вісник ХНУРЕ. 2021. № 3. С. 58–63.

17 Петренко О. В. Організація часу студентів: теоретичні підходи та практичні аспекти. Київ: Атіка, 2019. 120 с.

18 Priya P. What is PostgreSQL? [Електронний ресурс] / Pedamkar Priya. 2021. Режим доступу: https://www.educba.com/what-is-postgresql/

19 Richards M. Software Architecture Patterns. O’Reilly Media, 2019. 120 p.

20 Rosa L., Smith J. Effects of the Pomodoro Technique on Worker Productivity // International Journal of Productivity and Performance Management. 2018. Vol. 67, No. 3. P. 394–408.

21 Schafer Lauren. SQL vs NoSQL [Електронний ресурс]. URL: https://www.mongodb.com/nosql-explained/nosql-vs-sql

22 Sommerville I. Software Engineering. 10th ed. Boston: Pearson, 2015. 980 p.

23 Сидоренко М. Г. Адаптація методики GTD для студентів, які працюють // Науковий вісник Львівської політехніки. Серія: Менеджмент. 2021. Вип. 3. С. 112–119.

24 Скрипник О. В. Цифрові інструменти в організації навчальної діяльності студентів // Інформаційні технології в освіті. 2022. № 3. С. 37–42.

25 Вступ до JavaScript. Сучасний підручник з JavaScript. URL: https://uk.javascript.info/intro (дата звернення: 05.05.2023).

26 W3C. Web Content Accessibility Guidelines (WCAG) 2.1. 2018. URL: https://www.w3.org/TR/WCAG21 (дата звернення: 19.05.2025).

27 Застосування UML (частина 2). Діаграма послідовності – Sequence Diagram :: державний університет телекомунікацій. URL: https://dut.edu.ua/ua/news-1-626-7897- zastosuvannya-uml-chastina-2-diagrama-poslidovnosti---sequence-diagram_kafedrakompyuternih-nauk-ta-informaciynih-tehnologiy (дата звернення: 22.03.2023).

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

д.т.н., проф. Азаров О. Д

_____

"21" березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

Веб-застосунок «Розумний розклад»

08-54.БКР.029.00.000 ТЗ

Виконав: студент 4 курсу, групи 2СП-21 б

спеціальності 123 — «Комп'ютерна інженерія»

освітня програма – «Комп'ютерна інженерія»

Коваль М.В.

(прізвище та ініціали)

Керівник: к.пед.н., доц. каф. ОТ

Добровольська Н.В.

(прізвище та ініціали)

ВНТУ 2025

1 Підставою для виконання бакалаврської кваліфікаційної роботи (БКР)

1.1 Важливим є актуальність дослідження у напрямку бакалаврської роботи, яка обумовлена тим, що у реальних потребах сучасних користувачів — особливо студентів, які поєднують навчання з роботою. В умовах постійного навантаження і змінного розкладу виникає потреба в інструментах, які допомагають ефективно керувати власним часом. Сфера цифрового планування активно розвивається, і, поряд із базовими функціями календарів та списків завдань, користувачі дедалі частіше потребують більш гнучких і "розумних" рішень. Зокрема, важливою стає автоматична побудова персонального графіка з урахуванням пріоритетів, обмежень, зовнішніх подій і особистого темпу. Саме це й обумовлює практичну цінність і актуальність створеного застосунку;

1.2 Наказ про затвердження теми БКР від «20» березня 2025 року №97.

2 Мета БКР і призначення розробки

2.1 Мета роботи — розробка адаптивного веб-застосунку «Розумний розклад» для осіб, які поєднують навчання з роботою, що дозволяє автоматично формувати персоналізований розклад, враховуючи пріоритети користувача, обмеження по часу та типи активностей;

2.2 Призначення розробки — застосунок має стати зручним онлайн-інструментом для студентів і фахівців, які ведуть насичене життя. Система повинна забезпечувати просте внесення даних, автоматичну оптимізацію графіка, синхронізацію з Google Calendar та аналітичні рекомендації з покращення розподілу часу.

3 Вихідні дані для виконання БКР

3.1 Аналіз існуючих рішень для управління часом (Notion, Trello, Google Calendar);

3.2 Визначення функціональних і нефункціональних вимог;

3.3 Проєктування трирівневої клієнт-серверної архітектури (React, Node.js, PostgreSQL,Redis);

3.4 Реалізація клієнтської частини з використанням React та FullCalendar.js;

3.5 Реалізація серверної частини з Node.js/Express та REST API;

3.6 Тестування юзабіліті, продуктивності та сумісності на різних пристроях;

3.7 Проведення техніко-економічного обґрунтування ефективності впровадження;

3.8 Інтеграція з Google Calendar API та реалізація PWA-можливостей.

4 Вимоги до виконання БКР

Розробити повноцінний адаптивний веб-застосунок із наступними функціями:

— інтерфейс для введення навчального та робочого графіка;
— автоматичне генерування розкладу на основі пріоритетів;
— інтеграція з Google Calendar;
— адаптивний дизайн для мобільних пристроїв;
— реалізація PWA-функцій;
— панель адміністратора з можливістю редагування даних;
— високий рівень захисту персональних даних.

5 Етапи БКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 – Етапи БКР

№	Етапи роботи	Строки виконання	Очікувані результати
1	Постановка задачі	21.03.25	Формування вимог до системи
2	Аналіз аналогів, визначення вимог	21.03.25 – 30.03.25	Вибір стеку технологій, розробка специфікації
3	Проєктування архітектури	31.03.25 – 10.04.25	Створення схеми бази даних, інтерфейсів

Продовження таблиці А.1

4	Розробка клієнтської та серверної частини	11.04.25 – 24.04.25	Прототип з базовим функціоналом
5	Реалізація розумного алгоритму оптимізації розкладу	25.04.25 – 01.05.25	Алгоритм формування графіка
6	Інтеграція з Google Calendar та реалізація PWA	02.05.25 – 07.05.25	Синхронізація з календарем, офлайн-режим
7	Тестування, рефакторинг, оптимізація	08.05.25 – 11.05.25	Підвищення стабільності та продуктивності
8	Оформлення пояснювальної записки та графічного матеріалу	11.05.25 – 13.05.25	Підготовка до захисту

6 Матеріали, що подаються до захисту БКР

До захисту подаються: пояснювальна записка БКР, графічні і ілюстративні матеріали, протокол попереднього захисту БКР на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до БКР українською та іноземною мовами, довідка про відповідність оформлення БКР діючим вимогам.

7 Порядок контролю виконання та захисту БКР

Виконання етапів графічної та розрахункової документації БКР контролюється науковим керівником згідно зі встановленими термінами. Захист БКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання БКР

8.1 При оформлювання БКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія» (освітня програма «Системне програмування»). Кафедра обчислювальної техніки ВНТУ 2022;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

ДОДАТОК Б

ПРОТОКОЛ

ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи:

Веб-застосунок «Розумний розклад»

Тип роботи: Бакалаврська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності StrikePlagiarism

Оригінальність 99% Схожість 1%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою StrikePlagiarism щодо роботи.

Автор роботи _____ Коваль М. В.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Добровольська Н. В.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Графічна Частина

ВЕБ-ЗАСТОСУНОК «РОЗУМНИЙ РОЗКЛАД»

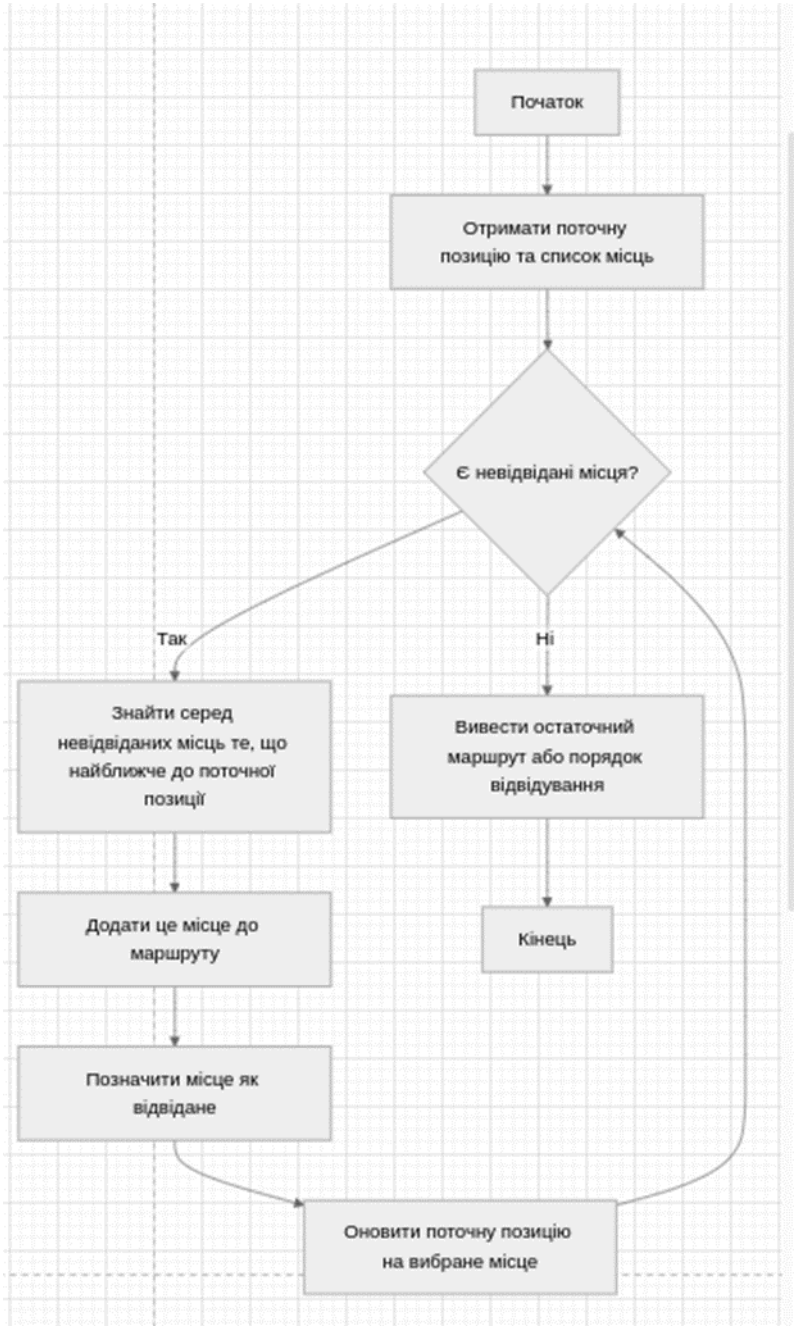

Рисунок В.1 — Демонстраційна блок-схема жадібного алгоритму

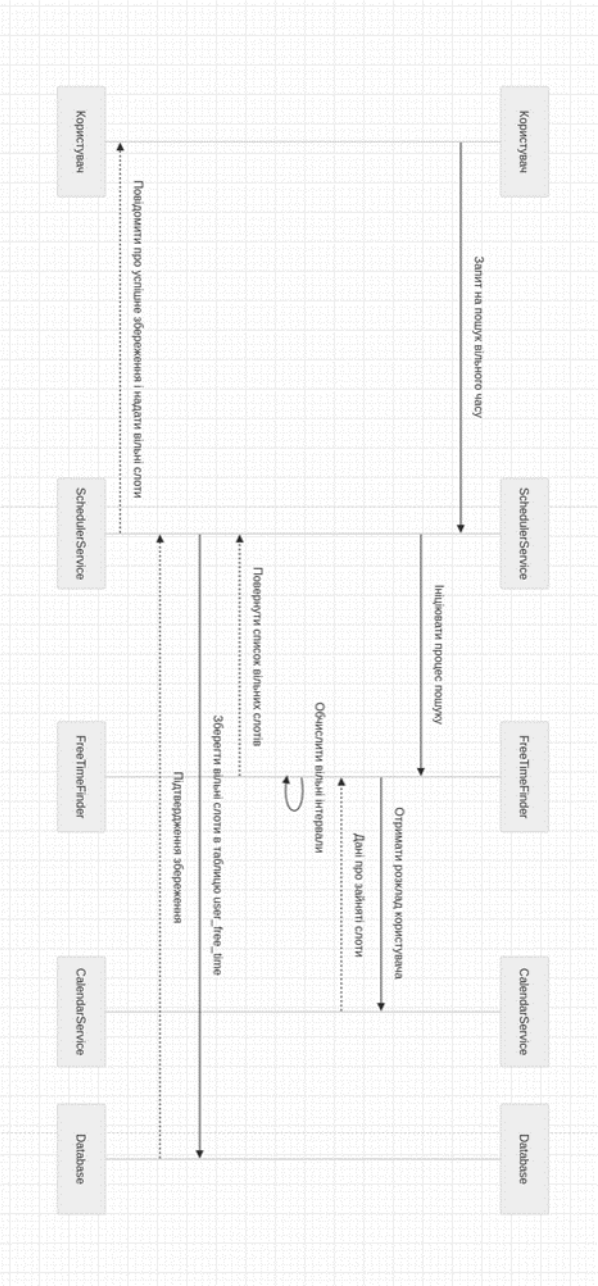

Рисунок В.2 — Sequnce діаграма конвеєра пошука вільного часу

ДОДАТОК Г

Ілюстративна Частина

ВЕБ-ЗАСТОСУНОК «РОЗУМНИЙ РОЗКЛАД»

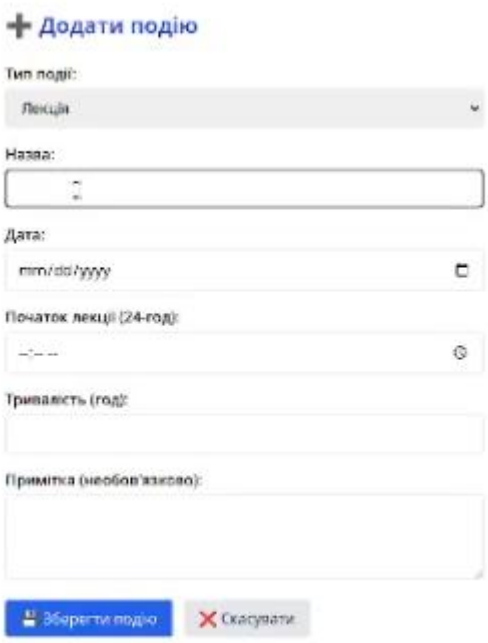

Рисунок Г.1 — Головне вікно додатку

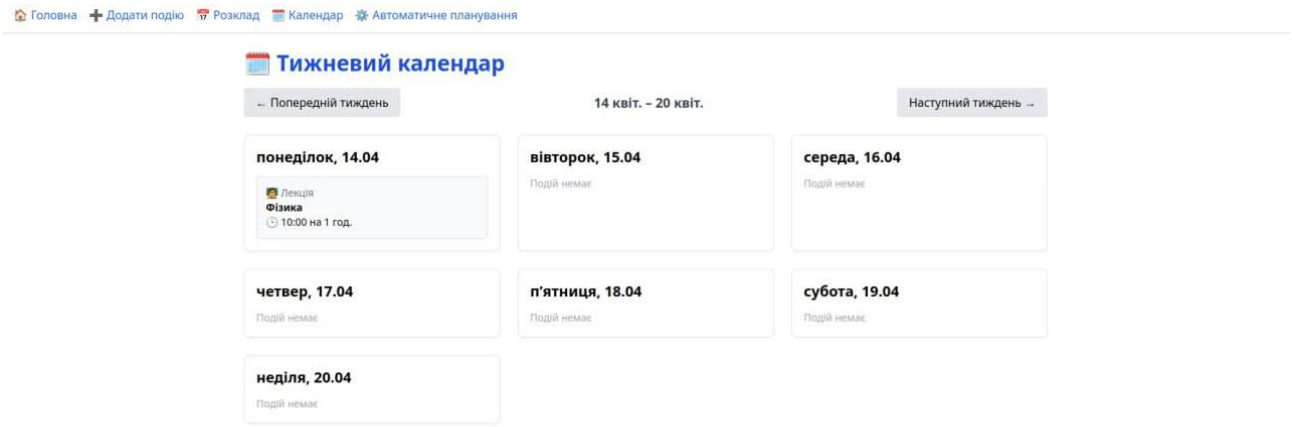

Рисунок Г.2 — Створення запиту

ДОДАТОК Д

Лістинг програмного забезпечення

```
unit grUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ImgList, Buttons, ExtCtrls, Menus, TeEngine,
  TeeFunci, Series, TeeProcs, Chart, ComCtrls, AbboX, OptionsForm;

type
  TForm1 = class(TForm)
    GroupBox1: TGroupBox;
    GroupBox2: TGroupBox;
    _1: TEdit;
    _2: TEdit;
    _3: TEdit;
    _4: TEdit;
    _5: TEdit;
    _6: TEdit;
    _1in: TEdit;
    _2in: TEdit;
    _3in: TEdit;
    _4in: TEdit;
    Panel1: TPanel;
    GroupBox5: TGroupBox;
    GroupBox6: TGroupBox;
    MainMenu1: TMainMenu;
```

```
    N1: TMenuItem;
    N2: TMenuItem;
    N4: TMenuItem;
    Reset: TSpeedButton;
    Results: TSpeedButton;
  end;
var
  Form1: TForm1;
implementation
{$R *.df
```