Список літератури:

1. White T. *Hadoop: The Definitive Guide*. O'Reilly Media, 2020.
2. Dean J., Ghemawat S. MapReduce: Simplified data processing on large clusters. *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI)*. 2004.
3. Zaharia M., Chowdhury M., Das T., Dave A., et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2012.
4. Карпенко Ю. І., Харченко А. І. Принципи розроблення рекомендаційних систем з використанням метаевристичної оптимізації. *Радіоелектроніка та телекомунікації*. 2022.
5. Войтов В. А., Фененко К. А., Кравцов А. Г. Експериментальні дослідження інформативних амплітуд акустичної емісії трибосистем при зміні конструктивних, технологічних та експлуатаційних факторів. *Проблеми тертя та зношування*. 2021. № 4(93). С. 4–15. DOI: 10.18372/0370-2197.4(93).16248.
6. Левкін Д. А., Жерновникова О. А. Розробка математичних моделей прикладних задач геометричного проєктування технічних систем. *Вісник Хмельницького національного університету. Серія «Технічні науки»*. 2022. № 4(311). С. 133–136. DOI: 10.31891/2307-5732-2022-311-4-133-136.
7. Назаров І. Я. Вебзастосунок для формування ціни квитка у мультисегментному сполученні з використанням великих даних: кваліфікаційна робота бакалавра. Одеса : Одеська політехніка, 2025. 158 с.
8. Тройніна А. С., Кулік Л. О., Назаров І. Я. Розробка інтерактивної платформи для пошуку автобусних квитків з використанням технологій розподіленої обробки великих даних. *Матеріали ІКТ*. 2025.
9. Apache Spark Documentation. URL: https://spark.apache.org/docs (дата звернення: 15.11.2025).
10. Stonebraker M. The case for shared nothing. *IEEE Database Engineering Bulletin*. 1986.
11. Armbrust M., Xin R., Lian C., Huai Y., et al. Spark SQL: Relational data processing in Spark. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 2015.

Troinina A.S., Nazarov I.Ya. INFORMATION SYSTEM FOR BUS TICKET SEARCH USING DISTRIBUTED BIG DATA PROCESSING TECHNOLOGIES

This article presents a comprehensive study and development of an information system designed to support the search for bus tickets within multi-segment transportation networks using distributed big data processing technologies. The rapid growth of digital mobility services, combined with increasing data volumes and user demands for accurate and fast route search results, has highlighted the need for high-performance computational solutions. Existing ticket-search platforms are often limited to direct route selection, do not adequately support multi-segment combinations with transfers, and rely on static or simplified pricing models that fail to reflect real market dynamics. These challenges create a strong motivation for designing an adaptive, scalable, and computation-efficient information system capable of providing real-time processing of transport data. The proposed system integrates four core components: client interface, server-level middleware, relational database, and a distributed computing module based on Apache Spark. Spark's in-memory processing capabilities significantly accelerate analytical operations such as filtering, aggregation, and route combination generation, making it suitable for complex transportation tasks. The system's data model represents essential elements of the transportation domain, including cities, distances, routes, and temporal parameters, enabling consistent data manipulation and seamless integration with distributed computing workflows. A multi-stage routing algorithm is implemented to determine direct routes, evaluate possible transfer points, calculate waiting times, and assemble optimal multi-segment paths. Additionally, a dynamic pricing mechanism is introduced, incorporating distance, travel duration, demand coefficients, and other logistic factors, allowing prices to reflect realistic market behaviour.

Key words: *distributed data processing, Apache Spark, routing, transport logistics, dynamic pricing, bus transportation.*

Дата надходження статті: 25.11.2025
Дата прийняття статті: 11.12.2025
Опубліковано: 30.12.2025

УДК 004.421:004.5
DOI https://doi.org/10.32782/2663-5941/2025.6.2/50

Турчак В.Р.
Національний університет «Львівська політехніка»

МОДЕЛЬ ІНФОРМАЦІЙНОЇ СИСТЕМИ БАГАТОКРИТЕРІАЛЬНОГО СОРТУВАННЯ ДАНИХ З ЇХ ДИНАМІЧНО-НАСЛІДКОВИМ ОБМЕЖЕННЯМ ПЕРЕМІЩЕННЯ

Інформаційні системи сортування структурованих даних є ключовими для аналітики, логістики, освітнього моніторингу та технічної безпеки. Їх ефективність визначається не лише точністю алгоритмів, а й узгодженістю архітектури етапів обробки – від імпорту та нормалізації до контролю впливу критеріїв і пояснюваності результатів. Метою роботи є побудова моделі інформаційної системи багатокритеріального сортування, що зберігає частковий початковий порядок за рахунок керованого локального переміщення елементів і забезпечує відтворюваність кожного кроку зміни рангу.

Розроблена модель охоплює повний цикл опрацювання табличних даних: перехід від початкового стану T_1 до нормалізованого T_1', специфікацію множини критеріїв T_2, побудову компараторів із підтримкою нечіткого текстового збігу на основі нормалізованої відстані Левенштейна та формування впорядкованого результату T_3 із журналом стабільності. Архітектура реалізована як набір взаємодіючих модулів (імпорт/попередня обробка; менеджер критеріїв; модуль порівняння; рушій сортування; відображення й експорт; журнал і метрики), для яких чітко визначено входи/виходи та інваріанти поведінки.

Алгоритмічне ядро ґрунтується не на повному пересортуванні масиву записів, а на поетапному частковому "підтягуванні" елементів відповідно до пріоритету критерію. Для критерію з пріоритетом pr_j вводиться обмеження переміщення $R_j = N/pr_j$ (де N – кількість рядків), що лімітує максимальну дальність локального зсуву за одну ітерацію. На кожному кроці система обирає локально найкращий елемент у допустимому вікні та підтягує його вгору не далі ніж на R_j позицій, виконуючи лише послідовні суміжні обміни (adjacent-only), а у випадку рівності значень застосовується стабільне розв'язання рівностей (tie-break) за початковим порядком. Така інформаційна технологія забезпечує контрольовану зміну ранжування без руйнування глобального порядку і не дозволяє критеріям нижчого пріоритету повністю переважати критерії з вищим пріоритетом.

Система формує впорядкований список T_3, супроводжений детальним журналом кроків та узагальненими метриками стабільності (зокрема Kendall's τ та середній зсув позиції). Це підвищує пояснюваність і відтворюваність результатів у прикладних задачах, а також забезпечує можливість кількісного оцінювання масштабу втручання в початковий порядок і проводити аудит прийнятих рішень на рівні окремих операцій.

Наукова новизна полягає у створенні моделі інформаційної системи багатокритеріального сортування як послідовності взаємодіючих модулів із жорсткими інваріантами (обмеження на переміщення, режим покрокової заміни, стабільний початковий порядок, пороговий нечіткий збіг), що мають узгоджені відповідники в моделях, структурі даних і програмних інтерфейсах. На відміну від існуючих моделей, створена модель поєднує керовану локальність впливу критеріїв із формалізованим журналюванням процесу, що забезпечує як стабільність часткового порядку, так і прозорий аудит кроків сортування.

Практична значущість полягає в тому, що запропонована модель орієнтована на використання у вебзастосунках без серверної постопрацювання, підтримує змішані типи полів (числові й текстові), забезпечує прозорий контроль сили впливу кожного критерію, передбачає технологію експорту результатів і журналу для подальшого аналізу та може інтегруватися в робочі процеси електронної комерції, технічного моніторингу, освітньої аналітики й управління ресурсами.

Ключові слова: *інформаційна система, багатокритеріальне сортування, модульна архітектура, динамічно-наслідкове обмеження переміщення, нечіткий збіг, стабільність порядку, структуровані дані.*

Постановка проблеми. У прикладних системах роботи з таблицями багатокритеріальне впорядкування зазвичай реалізують як послідовні повні пересортування або як лексикографічне сортування без чітко заданої межі впливу кожного критерію. У такій постановці навіть незначні зміни ваг чи додавання нового критерію можуть повністю зруйнувати вже сформований частковий порядок, а кроки зміни рангу не фіксуються і не підлягають подальшому аудиту. Для гібридних таблиць з різнотипними полями (числовими, текстовими тощо) додатково потрібні уніфіковані правила порівняння з підтримкою порогового нечіткого збігу та стабільне розв'язання рівностей (tie-break) зі збереженням початкового відносного порядку рядків. В умовах сталого щорічного зростання обсягів даних такі проблеми призводять до відчутних втрат ефективності в освітній аналітиці, логістиці та суміжних сферах, де помилки й непрозорість сортування безпосередньо впливають на якість прийнятих рішень.

За цих умов потрібна клієнтська модель, яка зберігає частковий початковий порядок не за рахунок повної перебудови масиву, а через кероване обмеження локального впливу кожного критерію залежно від його пріоритету, виконує переміщення елементів лише як послідовні суміжні обміни, застосовує стабільний механізм tie-break за початковим індексом, уніфікує порівняння для числових і текстових полів із підтримкою порогового нечіткого збігу та веде формальний журнал кроків сортування разом із метриками стабільності для подальшого аудиту. Така модель має бути реалізована повністю на боці клієнта (у браузері), спиратися на чітко визначені інваріанти й забезпечувати можливість відтворення кожного кроку зміни рангу без залучення серверної постобробки.

Об'єкт дослідження – процес багатокритеріального впорядкування табличних даних. Предмет дослідження – архітектурна модель клієнтської інформаційної системи з локально обмеженим впливом критеріїв, яка забезпечує відтворюваність кроків сортування та пояснюваність отриманих результатів у прикладних сценаріях використання.

Аналіз останніх досліджень і публікацій. Клієнтська обробка табличних структур. За останні роки зросла роль безсерверних вебрішень, де імпорт, нормалізація, фільтрація та сортування великих таблиць виконуються безпосередньо у браузері. Практичні інструменти на кшталт SheetJS/XLSX, PapaParse та Handsontable забезпечують стабільний і протестований набір технологій для читання XLSX/CSV/JSON, відображення грідів і базових операцій сортування/експорту [1–3]. Водночас ці засоби, як правило, реалізують або повне пересортування, або лексикографічне впорядкування без формалізованого контролю локальної сили впливу критерію та без покрокового журналу зміни порядку; стабільність відносного порядку і пояснюваність кроків розв'язуються частково або лишаються поза моделлю користувацького інтерфейсу. Дослідження з досвіду взаємодії користувача (UX) з табличними інтерфейсами підтверджують потребу в адаптивних перетвореннях і чітких поясненнях, але не дають формалізованого механізму керованих локальних перестановок у клієнті [4, с. 2–3; 5, с. 5–6].

MCDA: робастність ранжувань і (не)компенсаторність. Сучасні роботи з багатокритеріального аналізу рішень (TOPSIS, варіанти вагових схем) демонструють істотну чутливість ранжувань до нормалізації, вибору ваг та порогів прийнятності; дедалі більшої уваги набувають аналіз чутливості, сценарії невизначеності та обмеження компенсаторності між критеріями [6, с. 33–34; 7, с. 3–6; 8, с. 2–3]. Фактично мета зсувається від пошуку «єдиного правильного» порядку до контролю амплітуди допустимих змін позиції за варіювання параметрів. Проте більшість цих моделей залишаються на рівні глобальних перетворень і не задають механізму локального підняття елемента з формальним радіусом впливу у виконуваному коді веб-клієнта.

Outranking-процедури. Сімейства ELECTRE/PROMETHEE надають апарати порогів переваги/індиферентності та контроль компенсаторності, а також сильну пояснюваність «чому A краще за B» [9, Art. 103116; 10, с. 1–2]. Однак їх параметризація і складність перенесення у змішані (числово-текстові) дані роблять інтеграцію в типові браузерні застосунки нетривіальною; відсутня проста операційна інтерпретація у вигляді покрокових суміжних перестановок із явним радіусом локального зсуву, придатним для аудиту в користувацькому інтерфейсі (UI).

Лексикографічні та алгоритмічні засади стабільності. Лексикографічні схеми природно пріоритезують критерії й сумісні зі стабільними алгоритмами сортування, що зберігають відносний порядок рівних елементів. Недавні роботи з лексикографічної оптимізації акцентують саме на стабільності та структурі обмежень, але не встановлюють кількісної межі локального зсуву запису в реалізації веб-клієнта [11, с. 1–4]. Класичні

алгоритмічні джерела забезпечують технічну базу для стабільних процедур, однак питання керованої локальності (радіуса підняття) і формального журналювання кроків лишається поза їхнім фокусом [12, с. 75–80].

У попередніх працях автора було запропоновано механізм контрольованого «підтягування» елемента в межах локального радіуса з виконанням лише послідовних суміжних обмінів та зі стабільним розв'язанням рівностей за початковим індексом, формалізовано структуру критеріїв, а також покрокову взаємодію користувача з імпортом, заданням критеріїв, впорядкуванням і експортом із прив'язкою до викликів у коді [13, с. 20–22; 14, с. 175–176; 15, с. 14–15]. Водночас у відкритих джерелах не було знайдено цілісної клієнтської специфікації, яка одночасно вводить кількісну межу локального впливу критерію, гарантує режим суміжних обмінів зі стабільним розв'язанням рівностей, уніфікує порівняння числових і текстових полів із пороговим нечітким збігом, а також фіксує формалізований журнал кроків і метрики стабільності в архітектурі, орієнтованій на користувацький інтерфейс. Саме заповнення цієї прогалини і становить предмет подальшого викладу.

Постановка завдання. Метою статті є побудова моделі клієнтської інформаційної системи багатокритеріального сортування структурованих даних з їх динамічно-наслідковим обмеженням переміщення елементів, яка виконує впорядкування лише послідовними суміжними обмінами, застосовує стабільне розв'язання рівностей, підтримує пороговий нечіткий збіг для текстових полів і веде формалізований журнал кроків.

Для досягнення поставленої мети необхідно розв'язати такі завдання:

1. Розробити інформаційну технологію керованого впорядкування, що реалізує *R*-обмежене локальне переміщення елементів у режимі лише послідовних суміжних обмінів (adjacent-only) зі стабільним розв'язанням рівностей за початковим індексом і підтримкою порогового нечіткого збігу для числових і текстових атрибутів без повного пересортування масиву.

2. Сформувати модульну архітектуру клієнтської системи, яка охоплює етапи імпорту даних, очищення, задання критеріїв, компарації, впорядкування, журналювання, експорту та пояснення результатів у браузерному середовищі без серверної постобробки.

3. Побудувати формалізований журнал виконання та систему метрик стабільності (коефіцієнт рангової кореляції Кендала, середній зсув позиції, частка значних зсувів), що забезпечують відтворюваність, пояснюваність і аудит результатів багатокритеріального сортування.

Виклад основного матеріалу. Модель клієнтської інформаційної системи багатокритеріального сортування структурованих даних з їх динамічно-наслідковим обмеженням переміщення розглядаємо як керований конвеєр перетворень показаний на рис. 1, де вхідна таблиця T_1 поступово доводиться до стану, придатного для надійного порівняння, а далі – впорядковується з контрольованою локальністю. Для цього ми спочатку уніфікуємо дані (отримуємо T_1'), потім специфікуємо критерії T_2, на їх основі будуємо компаратори $\mathrm{cmp}(T_2[j])$, і вже на завершальному етапі застосовуємо *R*-обмежене впорядкування з детальним журналюванням, формуючи T_3. Таким чином, кожен наступний модуль працює на чітко визначеному, стандартизованому вхідному потоці й повертає прозорий, відтворюваний результат.

Спочатку користувач через інтерфейс *B*0 (UI) запускає роботу системи, генеруючи події з множини Σ: *ev_import*, *ev_define_criteria*, *ev_sort_all* / *ev_next_step*, *ev_export*, *ev_error*. Тобто саме користувач своїми діями («імпортувати файл», «задати критерії», «відсортувати», «експортувати», «повідомити про помилку») послідовно переводить систему з одного кроку на інший. На цьому рівні ми свідомо розділяємо дві ролі. Інтерфейс *B*0 відповідає за прийом дій користувача, збір параметрів через форми, перевірку коректності введення (чи заповнені всі обов'язкові поля, чи коректно задані пріоритети, пороги тощо) і за показ повідомлень та результатів на екрані. Натомість обчислювальні модулі *B*1–*B*7 виконують «важку» роботу: імпорт із файлів, нормалізацію даних, побудову критеріїв, власне сортування, ведення журналу кроків, розрахунок метрик стабільності та експорт/збереження результатів. Інтерфейс сам нічого не сортує і не вирішує, кого поставити вище в таблиці: він лише передає дані у відповідні модулі й показує те, що вони обчислили. На екрані користувач бачить очищену таблицю T_1', відсортований результат T_3 та метрики стабільності, розраховані на основі журналу кроків. Такий поділ ролей не дозволяє «приховати» критичну логіку всередині UI і спрощує аудит: усі суттєві перетворення зосереджені в чітко визначених модулях, де їх можна окремо перевірити й відтворити.

Модуль *B*1 реалізує імпорт вхідних даних із файлів XLSX, CSV, JSON. Як показано на рисунку 1, *B*1 отримує команду від *B*0, читає файл, пере-

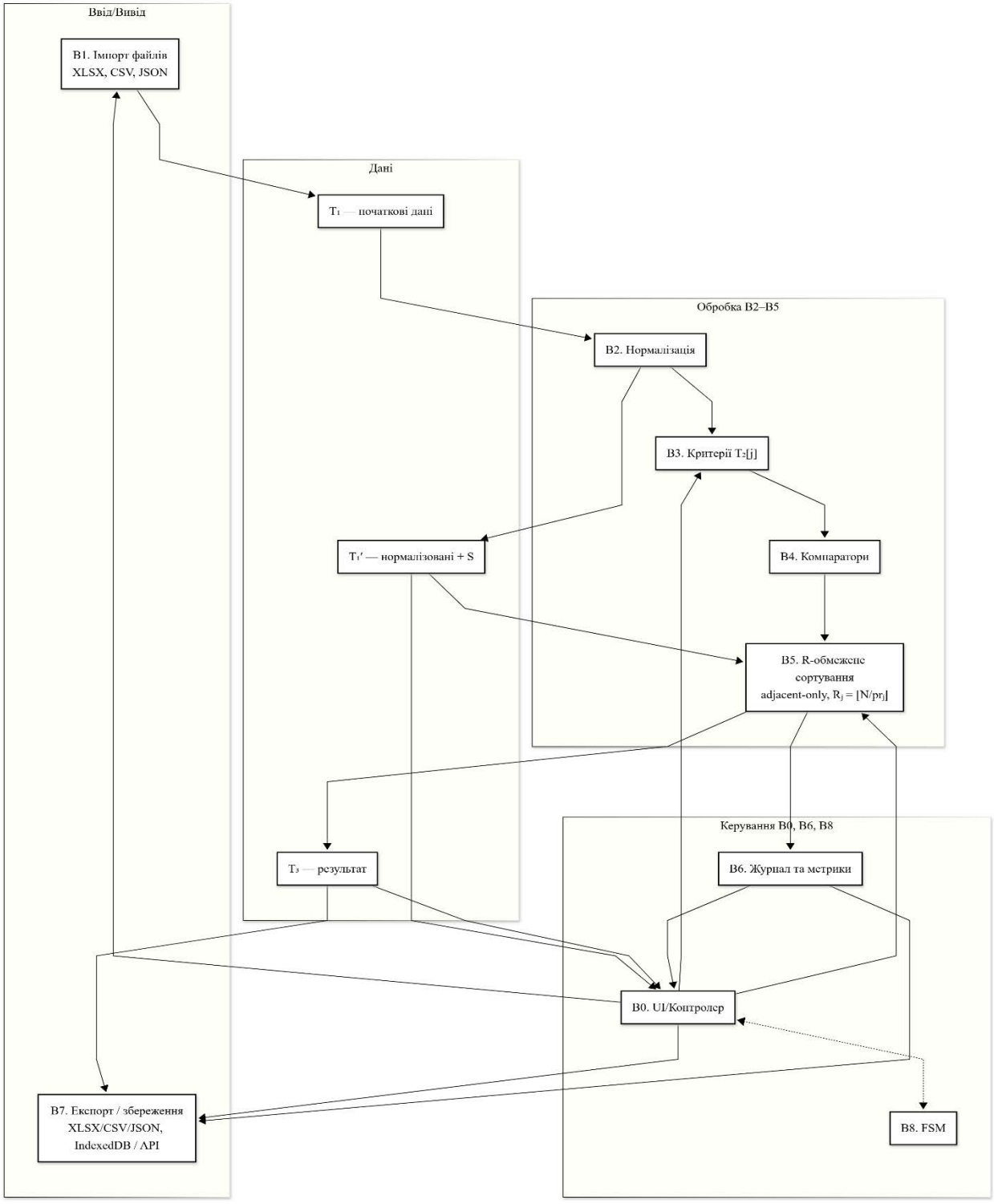


Рис. 1. Модель інформаційної системи багатокритеріального сортування даних з їх динамічно-наслідковим обмеженням переміщення

творює його до внутрішнього табличного представлення і формує початкову таблицю T_1, яка далі передається в модуль нормалізації $B2$. Отже, $B1$ відповідає за коректне та відтворюване завантаження сирих структурованих даних у систему.

Далі, оскільки будь-яке порівняння дуже чутливе до «сміття» у вхідних даних (зайві пробіли, різний регістр, неоднорідні типи, випадкові пропуски, дивні символи), першим кроком ми нормалізуємо вхідну таблицю в модулі $B2$. Для цього застосовується функція $f_1(T_1 \rightarrow T_1')$, яка по суті

є фільтром очищення. Вона послідовно прибирає зайві пробіли на початку та в кінці рядків (*trim*), щоб « Sensor » і «Sensor» не вважалися різними значеннями, зводить текст до єдиного регістру (нижній), щоб «Detector» і «detector» порівнювалися однаково, і за потреби усуває діакритики (наприклад, перетворює «é» на «e»), щоб незначні відмінності написання не ламали збіг. Далі f_1 приводить кожну колонку до одного з узгоджених типів із фіксованого набору {*number*, *text*, *date*, *bool*}, тобто перетворює те, що було рядком «10», у число 10, дати приводить до єдиного формату, логічні значення типу «так/ні» – до *bool*. Окремо фіксується політика обробки пропусків і *NaN*: за замовчуванням усі такі значення при порівнянні стабільно відносяться в кінець, щоб пропуски автоматично не «випереджали» заповнені рядки.

Паралельно з очищенням ми формуємо схему *S*, яку можна розуміти як «паспорт» таблиці. Для кожної колонки схема *S* зберігає її тип (число, текст, дата, *bool*), межі значень (*min*/*max*), а також параметри для нормування (масштаби, коефіцієнти, що дозволяють перевести значення у відносний інтервал, наприклад, [0;1]). Наприклад, для числової колонки «Ціна» ми запам'ятовуємо мінімальну та максимальну ціну, а для дати – найранішу і найпізнішу дату в цій колонці. Схема *S* формується автоматично, поки ми проходимо по рядках і очищуємо значення. Завдяки цьому T_1' стає канонічним, узгодженим за типами та форматами входом для наступних етапів, тобто всі подальші модулі працюють вже не з «як було в Excel», а з очищеною та однорідною таблицею. Схема *S* дає правдиві межі й типи колонок, на які ми далі спираємося при налаштуванні критеріїв та реалізації компараторів, не вгадуючи ці параметри щоразу заново.

Коли дані уніфіковано, у модулі *B*3 ми переходимо до задання критеріїв сортування. На цьому етапі формуємо множину $T_2 = \{T_2[j]\}$, $j = 1..m$, де кожен елемент $T_2[j]$ має структуру $T_2[j] = (c_j, w_j, d_j, f_j, pr_j, \tau_j)$.

Тут c_j – це цільовий атрибут, тобто колонка, на яку посилається критерій (індекс або ім'я колонки згідно зі схемою *S*). Параметр $w_j \geq 0$ – вага критерію (якщо використовується вагова схема), яка показує відносну важливість цього критерію у групових або агрегованих оцінках. Параметр $d_j \in \{min, max, asc, desc, match\}$ задає режим порівняння: *asc*/*desc* означають стандартне зростання/спадання, а *min*/*max* розглядаються як синоніми *asc*/*desc* після нормування (*min* – це «краще менше», *max* – «краще більше»). Режим *match* означає, що критерій працює не як звичайне «більше-менше», а як перевірка на збіг із деяким опорним значенням. Це опорне значення задається параметром f_j – шаблоном або еталоном, за яким ми шукаємо «схожі» рядки. Пріоритет pr_j визначає порядок важливості критеріїв: 1 – найвищий пріоритет; значення пріоритетів не повторюються й утворюють безперервну послідовність (тобто без пропусків). Нарешті, $\tau_j \in [0;1]$ – поріг нечіткого збігу, який використовується для текстових порівнянь: він визначає, яку мінімальну «схожість» треба досягти, щоб вважати рядок таким, що задовольняє критерію *match*.

На вході до *B*3 ми не просто зберігаємо ці параметри, а суворо перевіряємо їх на валідність. По-перше, цільові атрибути c_j мають бути унікальними, тобто кожен критерій працює з конкретною колонкою, і ми не дублюємо критерії для однієї й тієї ж колонки без потреби. По-друге, пріоритети pr_j мають бути без пропусків і дублювання, щоб утворювати чіткий порядок: 1, 2, 3, …, *m* без повторів і «дір». По-третє, режим d_j повинен бути узгоджений із типом колонки в схемі *S*: наприклад, режим *match* не застосовується до чисто числових полів, якщо для них не визначено текстове представлення, а режими *min*/*max* використовуються там, де є сенс говорити про «більше/менше». По-четверте, поріг τ_j має лежати в межах [0;1], тобто бути коректним коефіцієнтом, а не довільним числом. У результаті специфікація критеріїв T_2 перетворюється на формальний і перевірений вхідний артефакт для наступних етапів: саме на його основі будуть побудовані компаратори (реальні функції порівняння) і організований журнал кроків сортування.

У модулі *B*4 ми будуємо компаратори, тобто функції порівняння, які вже безпосередньо застосовуються до рядків у таблиці під час сортування. На цьому кроці абстрактні записи $T_2[j]$ перетворюються на операційні процедури $cmp(T_2[j])$. Для числових колонок ми спочатку нормуємо значення на основі параметрів із *S* (наприклад, приводимо їх до інтервалу [0;1], використовуючи *min*/*max* цієї колонки, щоб різні шкали – гривні, відсотки, метри – стали порівнюваними), а потім застосовуємо режими *asc*/*desc* (або їхні синоніми min/max). При цьому ми забезпечуємо, щоб *NaN* і порожні значення стабільно опинялися наприкінці, не змінюючи відносний порядок валідних елементів за початковим індексом *origIndex*. Це означає, що «порожні» рядки не будуть випадково «підстрибувати» вгору і не зламають логіку сортування.

Для текстових полів компаратор працює інакше. Ми обчислюємо нормалізовану відстань Левенш-

тейна $d\_\text{norm} \in [0;1]$ між поточним значенням у рядку та опорним шаблоном f_j. Відстань Левенштейна показує, скільки редагувань (вставок, видалень, замін) потрібно, щоб одне слово перетворити на інше; нормалізована версія приводить ці значення до інтервалу [0;1]. Потім ми визначаємо міру збігу $matchScore = 1 - d\_norm$: чим ближчі рядки, тим більший matchScore. Далі застосовуємо порогове правило $matchScore \geq \tau_j$: тобто ті рядки, де схожість із шаблоном перевищує або дорівнює порогу τ_j, вважаються такими, що «добре збігаються», і формують «клас переваги» за цим критерієм. Так ми отримуємо нечіткий, але контрольований збіг тексту: невеликі помилки чи варіанти написання не вибивають записи з вибірки.

Щоб не втрачати стабільність сортування, у випадку рівності значень за всіма параметрами компаратор щоразу застосовує *tie-break* за первісним індексом *origIndex*. Це означає, що якщо два рядки для критерію виявилися однаково «хорошими», їхній взаємний порядок залишається таким самим, як у вихідній таблиці. Така стабільність важлива, тому що на наступному етапі ми працюємо з локальними підняттями в межах «вікна» R_j, і будь-яка неточність або випадкова перестановка рівних елементів може породити зайві, погано пояснювані переміщення. З огляду на обчислювальні витрати, ми додатково кешуємо результати нормалізації рядка $normStr(x)$ (наприклад, уже обрізаний і приведений до нижнього регістру текст) та значення відстані Левенштейна $Lev(x, f_j)$ для фіксованого f_j. Якщо однаковий текст порівнюється з одним і тим самим шаблоном багато разів, ми не перерахуємо цю відстань щоразу, а візьмемо уже збережене значення. Це суттєво зменшує навантаження без зміни логіки порівняння.

Озброївшись компараторами, у модулі *B*5 ми запускаємо рушій сортування. Його ключова властивість – контрольована локальність впливу кожного критерію. Для критерію з пріоритетом pr_j ми задаємо радіус впливу $R_j = N / pr_j$, де $N = |T_1'|$ – кількість рядків у нормалізованій таблиці. Це означає, що для старших критеріїв (малий pr_j, наприклад 1) радіус R_j більший, і такі критерії можуть істотно змінювати порядок, «бачачи» майже весь список. Для молодших критеріїв (великий pr_j, наприклад 3, 4, 5) R_j менший, і їхній вплив обмежено локальними перестановками, які не руйнують структуру, сформовану старшими. Таким чином, пріоритети прямо пов'язані з «силою» втручання в початковий порядок.

Алгоритм роботи можна описати так. Для поточної позиції і ми розглядаємо підмасив $[i, \min(i + R_j, N - 1)]$ – це «вікно», в межах якого даний критерій має право шукати кращий кандидат для позиції і. У цьому вікні ми шукаємо «найкращий» елемент за компаратором $cmp(T_2[j])$ і визначаємо його індекс *bestIndex*. Далі обчислюємо $steps = bestIndex - i$, тобто відстань до цього елемента у вікні, $i\ move = min(steps, R_j)$, тобто фактичний дозволений зсув угору. Після цього зсуваємо обраний елемент вгору на *move* позицій за допомогою послідовних суміжних обмінів (adjacent-only), тобто міняємо його місцями з сусідніми рядками по одному кроку, поки він не досягне нової позиції. Якщо $steps > R_j$, ми не дозволяємо елементу стрибнути на всю відстань, а виконуємо лише часткове підтягування (на відстань R_j), завдяки чому й досягається бажана локальність: критерій не може «витягнути» елемент надто високо за один прохід.

Проходи виконуються в порядку зростання пріоритету: спочатку критерій із $pr_j = 1$, потім із $pr_j = 2$ і т. д. Таким чином, спочатку формується «каркас» порядку за найважливішими критеріями, які мають більший радіус впливу, а молодші критерії з меншими R_j можуть лише локально уточнювати цей порядок, не перекриваючи і не ламаючи результат старших. У поєднанні зі стабільним tie-break це забезпечує керовані, логічно пояснювані перестановки, де кожен крок має обмежений радіус дії й зрозуміле пояснення.

Щоб забезпечити відтворюваність усіх кроків, модуль *B*6 протоколює кожну операцію сортування. Для цього ми записуємо структуру (*id*, $P = pr_j$, *i*, R_j, *steps*, *move*, *swaps*), де *id* (*origIndex*) – незмінний ідентифікатор рядка, який не змінюється протягом усього процесу, P – активний пріоритет, за яким зараз виконується сортування, i – позиція опори, для якої ми шукаємо кращий елемент, R_j – поточний радіус впливу, *steps* – відстань до знайденого «найкращого» елемента у вікні, *move* – фактичний зсув угору, який ми дозволили, *swaps* – кількість виконаних суміжних обмінів (тобто скільки разів елемент поміняся місцями з сусідом). Після завершення проходів ми маємо повний трасувальний журнал: для кожного кроку чітко видно, хто, куди й чому змістився.

На основі цього журналу ми обчислюємо метрики стабільності. Коефіцієнт *Kendall's* τ показує, наскільки кінцевий порядок узгоджений із початковим: значення, близькі до 1, означають, що порядок мало змінився; малі або від'ємні значення означають суттєве перетасування. Середній модуль зсуву позиції *avgShift* обчислюється як середня абсолютна різниця між початковою

та фінальною позиціями рядків: він показує, на скільки позицій у середньому «переїхали» елементи. Частка великих зсувів *share*($\Delta > k$) відображає долю рядків, які змістилися більш ніж на заданий поріг k, тобто дає уявлення, чи були «радикальні» перестановки для окремих елементів. Таким чином, ми отримуємо не тільки впорядкований результат T_3, але й кількісну оцінку того, наскільки сильно система втручалася в початковий порядок, і можемо в будь-який момент відтворити траєкторію будь-якого запису, подивившись на його *id*/*origIndex* у журналі.

Коли результат готовий, модуль *B*7 забезпечує експорт і збереження даних. Як видно з рисунка 1, *B*7 приймає результат T_3 та журнал кроків із модулів *B*5 і *B*6, а також керувальні команди від *B*0. На основі цього він дозволяє вивантажити результат у файлові формати XLSX, CSV або JSON (наприклад, для передачі в іншу систему чи формування звіту) і зберегти T_3 разом із журналом у локальному сховищі браузера (IndexedDB, браузерна SQLite через sql.js) або на зовнішньому сервісі через REST API. Такий підхід, з одного боку, полегшує інтеграцію системи в реальні бізнес-процеси (дані можна забрати у звичному форматі або тримати локально), а з іншого – гарантує тривалу відтворюваність експериментів і можливість незалежної перевірки результатів: маючи T_3 і журнал, можна повторно проаналізувати, як саме був отриманий цей порядок.

Увесь рух між етапами координується скінченним автоматом у модулі *B*8. Ми фіксуємо дозволені стани (наприклад, *Idle* → *Loaded* → *CriteriaReady* → *Computing* → *Viewing* → *Viewing*/*Export*) і інваріанти, які забороняють нелегальні переходи. Це означає, що система не може «перескочити» через крок: наприклад, перехід із *Computing* назад у *CriteriaReady* неможливий без події *ev_finishSorting*, а запуск сортування (*ev_sort_all* або ш) блокується, поки не буде сформовано коректну множину критеріїв T_2 і система не перейде в стан *CriteriaReady*. Як відображено на рисунку 1, модуль ш8 отримує події Σ від *B*0 (стрілка *B*0 → *B*8) і повертає дозволені стани або повідомлення про помилки назад у *B*0 (стрілка *B*8 → *B*0). Якщо під час роботи стається збій, генерується подія *ev_error*, система переходить у стан *Error*, формує опис помилки в ш і повертає його в *B*0 у вигляді повідомлення (*toast* або *modal*), щоб користувач одразу бачив, що саме пішло не так і на якому етапі.

Оцінюючи властивості моделі в цілому, бачимо, що стабільність порядку забезпечується *tie-break* за *origIndex*, тобто рівні елементи зберігають свій початковий взаємний порядок. Обмеженість локального впливу критеріїв реалізується формулою $R_j = N / pr_j$, яка зв'язує пріоритет із максимальною «дальністю» переміщення. Пояснюваність гарантується наявністю детального журналу кроків і метрик стабільності, що дозволяють кількісно описати втручання в початковий порядок. За обчислювальною вартістю кожен прохід із вікном R_j має складність порядку $O(N \cdot R_j)$, а сумарна вартість є адитивною за пріоритетами, тобто загальна складність визначається сумою по всіх критеріях. При цьому зі збільшенням pr_j значення R_j зменшується, що природним чином обмежує витрати на критерії з нижчим пріоритетом: молодші критерії працюють із меншими вікнами й не перевантажують систему зайвими перестановками. Отже, досягається баланс між керованістю, прозорістю та продуктивністю.

Висновки. У роботі розроблено клієнтську модель системи багатокритеріального сортування табличних даних, яка поєднує локальне *R*-обмежене переміщення елементів ($R_j = N/pr_j$), режим adjacent-only зі стабільним tie-break за початковим індексом, уніфіковане порівняння числових і текстових атрибутів з пороговим нечітким збігом та формалізовану систему журналювання й метрик стабільності.

1. Розроблено керовану інформаційну технологію впорядкування структурованих даних із підтримкою порогового нечіткого порівняння, що забезпечує збереження часткового початкового порядку без повного пересортування масиву.

2. Побудовано модульну архітектуру *B*0–*B*8 із подієвим керуванням (FSM), що охоплює повний цикл: імпорт, очищення, специфікацію критеріїв, впорядкування, пояснення та експорт, зберігаючи контроль над локальним впливом критеріїв.

3. Запроваджено журнал кроків та систему метрик (*Kendall's τ,* середній зсув позиції, частка великих зсувів), які забезпечують відтворюваність, пояснюваність та аудит змін порядку на рівні окремих операцій.

Створена модель системи є інтегрованою у браузерні клієнтські середовища без потреби у серверній постопрацюванні, зберігає контроль над локальним впливом критеріїв і гарантує пояснюваність кожного кроку сортування.

Перспективи подальших досліджень полягають в оптимізації обчислювальної складності моделі для великих значень N, аналізі чутливості результатів до вибору параметрів pr_j та τ_j, а також у порівнянні *R*-обмеженого підходу з методами TOPSIS та outranking-процедурами в клієнтському (браузерному) середовищі.

Список літератури:

1. SheetJS (XLSX). Docs (Community Edition). 2025. URL: https://docs.sheetjs.com (дата звернення: 20.10.2025).
2. Papa Parse. Documentation. 2025. URL: https://www.papaparse.com/docs (дата звернення: 20.10.2025).
3. Handsontable. Multi-column sorting API / Row sorting guide. 2023. URL: https://handsontable.com/docs/javascript-data-grid/api/multi-column-sorting/ (дата звернення: 20.10.2025).
4. Díaz Ceñera M., Grigera J., Pascual Espada J. Enhancing data tables user experience via automated adaptations. *Expert Systems with Applications*. 2024. Vol. 255. Art. 124491. DOI: 10.1016/j.eswa.2024.124491.
5. Huang Y., Miao Y., Weng D., Perer A., Wu Y. StructVizor: Interactive profiling of semi-structured textual data. In: CHI '25: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems. 2025. Pp. 393:1–393:18. DOI: 10.1145/3706598.3713484.
6. Bánhidi Z., Dobos I. Sensitivity of TOPSIS ranks to data normalization and objective weights on the example of digital development. *Central European Journal of Operations Research.* 2024. Vol. 32, No. 1. Pp. 29–44. DOI: 10.1007/s10100-023-00876-y.
7. Więckowski J., Sałabun W. Sensitivity analysis approaches in multi-criteria decision-making: A systematic review. *Applied Soft Computing*. 2023. Vol. 148. Art. 110915. DOI: 10.1016/j.asoc.2023.110915.
8. Gaona À., Guisasola A., Baeza J. A. An integrated TOPSIS framework with full-range weight sensitivity analysis for robust decision analysis. *Decision Analytics Journal*. 2025. Vol. 17. Art. 100642. DOI: 10.1016/j.dajour.2025.100642.
9. Liu X., Liu Y. Sensitivity analysis of the parameters for preference functions and rank reversal analysis in the PROMETHEE II method. Omega. 2024. Vol. 128. Art. 103116. DOI: 10.1016/j.omega.2024.103116.
10. Jokar F., Jalali Varnamkhasti M., Hadi-Vencheh A. A new ELECTRE method based on left and right score for multicriteria decision-making. *Computational Intelligence and Neuroscience*. 2023. Article ID 6414686. DOI: 10.1155/2023/6414686.
11. Abernethy J., Schapire R. E., Syed U. Lexicographic optimization: Algorithms and stability. *arXiv preprint*. 2024. URL: https://arxiv.org/abs/2405.01387 (дата звернення: 20.10.2025).
12. Knuth D. E. The Art of Computer Programming. Vol. 3: Sorting and Searching. 2nd ed. Reading (MA): Addison-Wesley, 1998. 780 p.
13. Овсяк В. К., Турчак В. Р., Овсяк О. В. Модель вибору сповіщувачів системи безпеки. *Український журнал інформаційних технологій*. 2023. Т. 5, № 2. С. 17–24. DOI: 10.23939/ujit2023.02.017.
14. Турчак В. Р., Овсяк О. В. Модель алгоритму багатокритеріального сортування даних з їх динамічно-наслідковим обмеженням переміщень. *Науковий вісник НЛТУ України*. 2025. Т. 35, № 3. С. 175–182. DOI: 10.36930/40350319.
15. Овсяк В. К., Турчак В. Р. Модель користувацького інтерфейсу для сортування структурованих даних. *Комп'ютерно-інтегровані технології: освіта, наука, виробництво.* 2025. Вип. 60. С. 13–30. DOI: 10.36910/6775-2524-0560-2025-60-02.

Turchak V.R. MODEL OF AN INFORMATION SYSTEM FOR MULTI-CRITERIA DATA SORTING WITH DYNAMIC CONSEQUENCE-BASED MOVEMENT CONSTRAINT

Information systems for sorting structured data are crucial for analytics, logistics, educational monitoring and technical security. Their effectiveness is determined not only by algorithmic accuracy, but also by the consistency of the processing architecture – from import and normalization to the control of criteria influence and the explainability of results. The purpose of this work is to develop a model of an information system for multi-criteria sorting that preserves a partial initial order through controlled local movements of elements and ensures reproducibility of each ranking change step.

The proposed model covers the full cycle of processing tabular data: transition from the initial state T_1 to the normalized T_1', specification of the set of criteria T_2, construction of comparators with fuzzy text matching based on the normalized Levenshtein distance, and formation of the ordered result T_3 accompanied by a stability log. The architecture is implemented as a set of interacting modules (import/preprocessing; criteria manager; comparison module; sorting engine; visualization and export; logging and metrics) with clearly defined inputs/outputs and behavioral invariants.

The algorithmic core is based not on complete re-sorting of the record array but on stepwise partial "pulling up" of elements according to the priority of each criterion. For a criterion with priority pr_j a movement constraint $R_j = N / pr_j$ is introduced (where N is the number of rows), which limits the maximum distance of local displacement in a single iteration. At each step, the system selects the locally best element within the admissible window and pulls it upwards by no more than R_j positions using only consecutive adjacent exchanges (adjacent-only). In the case of equal values, a stable tie-breaking strategy is applied based on the

initial order. This information technology provides controlled changes in ranking without destroying the global order and prevents lower-priority criteria from completely dominating higher-priority ones.

The system produces an ordered list T_3 accompanied by a detailed step log and aggregated stability metrics (in particular Kendall's τ and the mean position shift). This increases the explainability and reproducibility of results in applied tasks and enables quantitative assessment of the extent of intervention in the initial order and auditing of decisions at the level of individual operations. The scientific novelty lies in the development of a model of an information system for multi-criteria sorting represented as a sequence of interacting modules with strict invariants (movement constraints, stepwise replacement mode, stable initial order, threshold-based fuzzy matching) that have consistent counterparts in the models, data structures and program interfaces. Unlike existing models, the proposed model combines controlled locality of criteria influence with formalized process logging, which ensures both stability of the partial order and transparent auditing of sorting steps. The practical significance is that the model is oriented toward use in web applications without server-side post-processing, supports mixed field types (numeric and textual), provides transparent control over the strength of each criterion's influence, includes a technology for exporting results and logs for further analysis, and can be integrated into workflows of e-commerce, technical monitoring, educational analytics and resource management.

Key words: *information system, multi-criteria sorting, modular architecture, dynamic consequence-based movement constraint, fuzzy matching, order stability, structured data.*

Дата надходження статті: 17.11.2025
Дата прийняття статті: 04.12.2025
Опубліковано: 30.12.2025

УДК 004.4:004.2]:004.6
DOI https://doi.org/10.32782/2663-5941/2025.6.2/51

Удод О.С.
Запорізький національний університет

Кривохата А.Г.
Запорізький національний університет

РОЗРОБКА ВИСОКОМАСШТАБОВАНОЇ БЕЗСЕРВЕРНОЇ ХМАРНОЇ АРХІТЕКТУРИ СИСТЕМИ УПРАВЛІННЯ ЗАВДАННЯМИ

У статті розглянуто актуальну науково-прикладну задачу підвищення ефективності розробки та експлуатації систем управління завданнями в умовах динамічних та нерівномірних навантажень. Проаналізовано недоліки традиційних монолітних та серверних архітектур, пов'язані з обмеженою масштабованістю та високими операційними витратами на утримання інфраструктури. Запропоновано використання концепції безсерверних обчислень (Serverless Computing) та моделі «функція як послуга» (FaaS). Даний підхід, на відміну від існуючих рішень, дозволяє оптимізувати використання ресурсів та забезпечити автоматичне горизонтальне масштабування.

Для визначення вимог здійснено порівняльний аналіз провідних комерційних (Wrike, JIRA) та відкритих (OpenProj) аналогів. Комплекс функціональних та нефункціональних вимог до системи сформульовано з акцентом на показники продуктивності, безпеки даних та відмовостійкості. Розроблено набір UML-діаграм (прецедентів, компонентів, розгортання), які ілюструють структуру розподіленого вебзастосунку.

Обґрунтовано вибір технологічного стека: фреймворк Angular для реалізації клієнтської частини (Single Page Application) та екосистеми Amazon Web Services (Lambda Node.js, API Gateway, RDS PostgreSQL) для бекенду. Особливу увагу приділено питанням інформаційної безпеки: впроваджено механізм захищеного управління обліковими даними через AWS Secrets Manager та stateless-автентифікацію.

Емпірична верифікація розробленого рішення проведена шляхом багаторівневого тестування. Застосування модульних тестів (Jest) та навантажувального тестування (Artillery) дозволило оцінити поведінку системи під стресовим навантаженням. Результати експерименту підтвердили, що запропонована архітектура забезпечує стабільну обробку до 20 запитів на секунду при збереженні стабільних часових характеристик. Отримані дані свідчать про те, що розроблена система є ефективним, економічно вигідним та масштабованим рішенням, готовим до впровадження в корпоративному середовищі для автоматизації проєктного менеджменту.

Ключові слова: *комп'ютерні інформаційні технології, система управління завданнями, безсерверна архітектура, Angular, AWS Lambda, PostgreSQL, API Gateway, JWT, навантажувальне тестування.*

Постановка проблеми. У сучасному цифровому середовищі системи управління завданнями є критично важливими для забезпечення ефективності командної роботи та організації бізнес-процесів. У сфері розробки програмного забезпечення та управління проєктами спостерігається значне розгалуження подібних систем. Це призвело до появи значної кількості різноманітних систем, кожна з яких пропонує свій унікальний набір функцій, підтримуваних методологій (Agile, Kanban, Waterfall) та власний підхід до організації робочого процесу (від простих одноріневих до комплексних та декомпозованих багаторівневих задач). Це розгалуження створює гостру необхідність детального аналізу існуючих варіантів, щоб зрозуміти їхні функціональні та нефункціональні можливості, особливості інтеграції, масштабованість та придатність до різних типів команд і проєктів, які суттєво відрізняються за своєю спрямованістю, складністю управління та специфікою бізнес-процесів.

Зі зростанням обсягів даних, збільшенням кількості користувачів та динамічністю робочих навантажень, ключовими вимогами до систем управління завданнями стають висока масштабованість, надійність та економічна ефективність експлуатації. Традиційні монолітні та серверні архітектури мають суттєві обмеження для систем

управління завданнями з динамічними робочими навантаженнями, що проявляється у таких ключових недоліках:

1. Неефективне масштабування. Складнощі у швидкому масштабуванні інфраструктури під час пікових навантажень та, навпаки, надлишкові ресурси та неефективні витрати у періоди низької активності, що є критичним для систем, які обслуговують різнорівневі робочі процеси.

2. Операційні витрати. Значні витрати часу та ресурсів на адміністрування, оновлення та обслуговування серверів.

Концепція безсерверних обчислень позиціонується як новий та сучасний підхід, що може забезпечити оптимальне рішення для підтримки різноманітних та динамічних робочих процесів систем управління завданнями, мінімізуючи операційні витрати [1].

Виходячи з існуючої проблеми, метою даного дослідження є розробка, імплементація та експериментальна оцінка ефективності високомасштабованої Serverless-архітектури системи управління завданнями, що забезпечить оптимальне співвідношення між продуктивністю, гнучкістю підтримки складних робочих процесів та економічною ефективністю експлуатації.

Аналіз останніх досліджень і публікацій. У межах дослідження здійснено порівняльний аналіз функціональних особливостей існуючих аналогів проєктованої системи. До переліку провідних рішень на ринку, які є об'єктом подальшого аналізу, включено: Wrike, Worksection, Asana, Trello, Microsoft Planner, JIRA, Microsoft Project та OpenProj [2, 3]. Детальний огляд деяких платформ представлено нижче.

Trello – це хмарна платформа, що імплементує методологію Kanban для візуалізації робочих процесів та відстеження завдань. Система використовує структуру дошок, списків та карток для організації завдань, забезпечуючи високу гнучкість та низький поріг входження. Архітектура Trello орієнтована на прості та гнучкі робочі процеси та ефективну командну співпрацю. Проте, аналіз функціональних можливостей виявляє суттєві обмеження для комплексного проєктного менеджменту: відсутність нативних підзадач та управління залежностями, слабкі інструменти звітності та недостатня масштабованість для великих, багатокомпонентних корпоративних проєктів [2, 4].

Wrike позиціонується як комплексна платформа для автоматизації повного життєвого циклу управління проєктами, орієнтована на оптимізацію робочих процесів та розподіл ресурсів. Архітектура системи забезпечує гнучку адаптацію під специфічні бізнес-вимоги завдяки широким можливостям кастомізації та розвиненим механізмам інтеграції із зовнішніми сервісами. До ключових переваг платформи належать універсальність, розвинений інструментарій для управління залежностями завдань та засоби узгодження етапів роботи. Водночас, аналіз експлуатаційних характеристик виявив низку суттєвих обмежень: висока вартість ліцензування (стартова ціна становить близько $10 за користувача/місяць), функціональна надлишковість, що ускладнює інтерфейс для малих команд, та суворі ліміти безкоштовної версії, що знижує доступність системи для стартапів та бюджетних проєктів [4, 5].

Jira (Atlassian) є спеціалізованим інструментом для автоматизації життєвого циклу розробки програмного забезпечення, орієнтованим на використання методологій Agile. Функціональна архітектура платформи забезпечує глибоку інтеграцію з системами контролю версій та CI/CD (Bitbucket, Jenkins), а також гнучке моделювання робочих процесів. Попри розвинені можливості трекінгу дефектів, впровадження системи ускладнюється високим порогом входження через складність адміністрування та значною вартістю масштабування. Крім того, функціональна надлишковість робить її використання неефективним для малих команд із обмеженими ресурсами [6].

OpenProj – це десктопне рішення з відкритим вихідним кодом, що позиціонується як безкоштовна альтернатива пропрієтарним системам (зокрема, MS Project). Функціонал системи охоплює класичні інструменти проєктного менеджменту: діаграми Ганта, мережеві графіки та структуру декомпозиції робіт. Ключовими перевагами є відсутність ліцензійних витрат та повна сумісність форматів даних із MS Project. Проте впровадження OpenProj у сучасних розподілених командах обмежене через відсутність механізмів хмарної синхронізації та інструментів для спільної роботи. До суттєвих недоліків також належать застарілий інтерфейс, відсутність технічної підтримки та проблеми стабільності при обробці великих масивів даних. Крім того, виявлено складнощі із кросплатформенністю в середовищах, відмінних від Windows [7].

Отже, висока насиченість ринку систем управління завданнями створює умови для гнучкого вибору інструментів залежно від специфіки проєктної діяльності. Проте, наявність значної кількості платформ із різними архітектурними та функціональними особливостями вимагає від

організацій ретельного попереднього аудиту для мінімізації ризиків та оптимізації співвідношення ціни та якості.

Постановка завдання. Метою статті є проєктування, програмна реалізація та експериментальна оцінка системи управління завданнями побудованої на базі безсерверної архітектури з використанням стека AWS Lambda та Angular, а також обґрунтування її переваг порівняно з традиційними монолітними або мікросервісними підходами.

Виклад основного матеріалу. При розробці фронтенд частини програмного забезпечення вибір фреймворку відіграє критичну роль у забезпеченні ефективності, масштабованості та зручності використання продукту. Після ретельного аналізу наявних рішень як основну технологію розробки клієнтської частини системи управління задачами було обрано Angular, через його комплексну архітектуру, вбудовані інструменти та TypeScript-інтеграції, які забезпечують необхідну структурованість, безпеку та масштабованість для реалізації функціональних вимог системи [8, 9, 10].

Для реалізації серверної частини системи управління задачами було проведено аналіз хмарних сервісів Amazon Web Services (AWS), які надають можливості для створення масштабованих, безпечних, та економічно ефективних рішень. При розробці системи управління завданнями, найактивніше було залучено такі сервіси: AWS Lambda; Amazon VPC; Amazon EC2; RDS PostgreSQL; AWS Secrets Manager; Amazon API Gateway [11-15].

Важливим етапом проєктування та імплементації програмного продукту є проведення детального аналізу вимог. Для системи управління завданнями «TaskManager» ключовий функціонал охоплює повний життєвий цикл завдання (від створення до завершення) та підтримку ефективної командної взаємодії. Основні функціональні вимоги включають:

1. Управління об’єктами. Створення, редагування та архівування завдань з підтримкою багаторівневої категоризації (за проєктами та типами) та можливістю встановлення пріоритетів та термінів виконання.

2. Спільна робота. Механізми призначення завдань конкретним виконавцям, система нагадувань та сповіщень про зміни статусу, коментування та обмін файлами у форматі чату.

3. Візуалізація та звітність. Відстеження прогресу виконання завдань у режимі реального часу, підтримка різних способів візуалізації даних (наприклад, Kanban-дошки або списковий формат), а також генерування деталізованих звітів з використанням параметричних фільтрів.

4. Інтеграція та безпека. Інтеграція з календарними сервісами для синхронізації термінів та реалізація механізмів контролю доступу (автентифікація та авторизація) для розмежування прав користувачів.

Особлива увага приділяється створенню інтуїтивного інтерфейсу для швидкого створення та модифікації завдань, що є критичним фактором для підвищення продуктивності користувачів.

Нефункціональні вимоги визначають якісні характеристики системи, які є визначальними для її архітектури, експлуатації та сприйняття користувачем. Категоріально вони систематизовані наступним чином:

1. Продуктивність. Система повинна забезпечувати час відгуку не більше 2 секунд для 95% операцій, навіть при піковому навантаженні, підтримуючи одночасну роботу не менше 20 користувачів.

2. Надійність та Доступність. Забезпечення доступності системи на рівні не менше 99.5% часу, з мінімальними плановими простоями.

3. Масштабованість. Архітектура повинна підтримувати горизонтальне масштабування для коректної обробки зростаючої кількості даних та користувачів без необхідності повного перепроєктування.

4. Безпека. Захист даних через шифрування, аудит дій користувачів та забезпечення захисту від типових кібератак.

5. Зручність та адаптивність. Інтерфейс повинен відповідати сучасним принципам UX/UI дизайну, бути інтуїтивно зрозумілим і забезпечувати коректне відображення на різних пристроях.

На рисунку 1 представлена діаграма варіантів використання, що визначає взаємодію двох ключових акторів: Адміністратора проєкту та Користувача. Адміністратор володіє розширеними дозволами, які охоплюють повний цикл управління проєктами (створення, зміна, видалення) та управління користувацькими завданнями. Авторизований користувач має доступ до повного життєвого циклу управління завданнями (CRUD-операції), включаючи встановлення ключових атрибутів (назва, пріоритет, дедлайн) та організацію завдань у папки. Система також забезпечує необхідні процеси автентифікації/авторизації та комунікаційний функціонал – додавання коментарів із підтримкою текстових та медіавкладень.

На рисунку 2 зображено IDEF1X діаграму бази даних, що складається з 11 сутностей. Центральна сутність Tasks пов'язана з проєктами (Projects), категоріями, статусами та пріоритетами. Управління користувачами (Users) та їхніми ролями (Roles) реалізовано через проміжну сутність Task_Assignees, що забезпечує зв'язок «багато до багатьох» між виконавцями та завданнями. Функціонал обговорення забезпечують сутності Comments та Media (з типами у Content_types). Така структура підтримує категоризацію, розподіл завдань та обмін файлами.

Представлена діаграма компонентів (рис. 3) фронтенд-частини системи, розробленої на базі Angular, відображає модульну архітектуру додатку. Ядром системи є компоненти Project та Task, взаємодія між якими забезпечується через Router. Компонент Task декомпозовано на під-

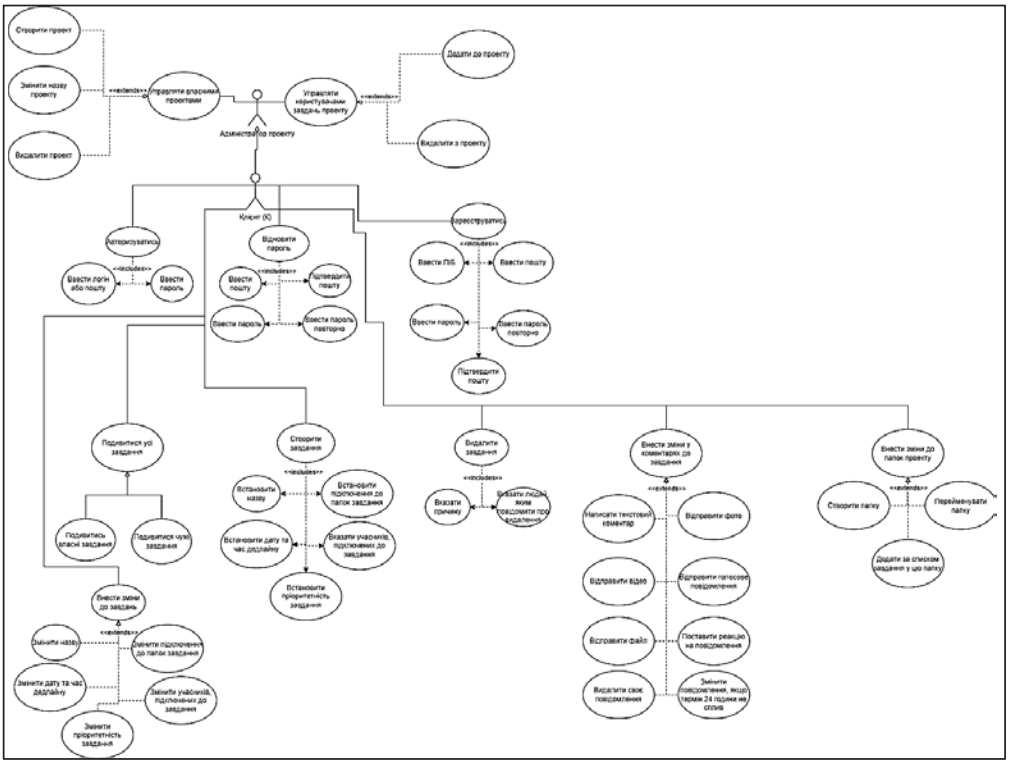


Рис. 1. Діаграма варіантів використання

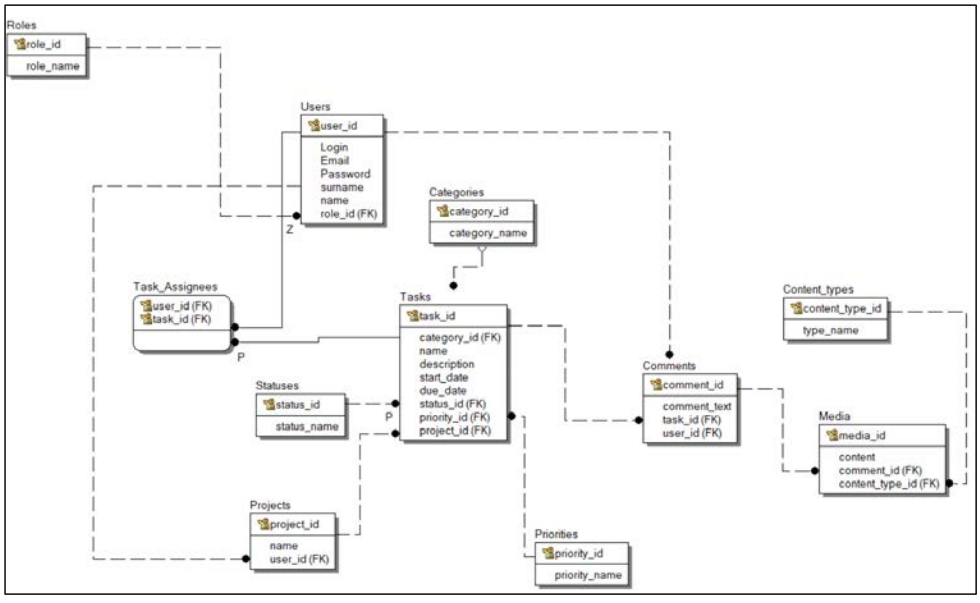


Рис. 2. IDEF1X діаграма системи управління завданнями

модулі: TaskBoard (візуалізація kanban-дошки) та TaskCreator (створення/редагування), що комунікують через систему подій.

Взаємодію з хмарною інфраструктурою AWS Lambda абстраговано через сервісний шар TaskService, що оперує типізованими моделями даних TaskModel, UserModel. Це дозволяє інкапсулювати логіку API-запитів та ефективно застосовувати патерн впровадження залежностей (Dependency Injection). Допоміжні модулі Shared, Admin, Chart забезпечують навігацію, адміністрування та аналітику. Така архітектурна організація сприяє слабкій зв'язаності коду, спрощує підтримку системи та забезпечує виконання вимог щодо продуктивності та масштабованості.

На рисунку 4 представлено реалізацію компонента app-login.ts, що забезпечує механізм авторизації на базі Angular Reactive Forms. Використання реактивних форм дозволяє реалізувати сувору валідацію вхідних даних (формат email, обов'язковість полів) на клієнтському боці. Логіка взаємодії з сервером інкапсульована в сервісі AuthService, який обробляє запити автентифікації, тоді як управління навігацією після успішного входу делеговано модулю Router. Такий підхід забезпечує модульність коду та централізовану обробку бізнес-логіки.

На рисунку 5 зображено TaskCreatorComponent, що забезпечує інтерфейс створення завдань засобами Angular Reactive Forms. Ініціалізація форми здійснюється через FormBuilder із встановленням правил валідації для полів title, priority та status. Архітектура компонента реалізована за принципом «Smart/Dumb components»: він не виконує збереження даних самостійно, а делегує цю операцію батьківському контролеру через механізм подій (@Output, EventEmitter).

Взаємодія з бекендом побудована на безсерверній архітектурі. Клієнтський додаток надсилає запити до Amazon API Gateway, який маршрутизує їх до відповідних функцій AWS Lambda. Така модель дозволяє інкапсулювати бізнес-логіку та операції з базою даних у масштабовані функції, що виконуються виключно за вимогою, забезпечуючи оптимізацію ресурсів.

На рисунку 6 представлено реалізацію Lambda-функції для отримання даних про зв'язки між користувачами та проєктами (project_assignees). У межах безсерверної архітектури функція є бекенд-обробником, що ініціює з'єднання з PostgreSQL. Безпека доступу до бази даних забезпечується інтеграцією з AWS Secrets Manager, який надає необхідні облікові дані без їх жорсткого кодування в тілі функції.

Алгоритм роботи передбачає виконання SQL-запиту на вибірку та серіалізацію отриманих результатів у формат HTTP-відповіді (JSON). Програмна реалізація включає механізми обробки

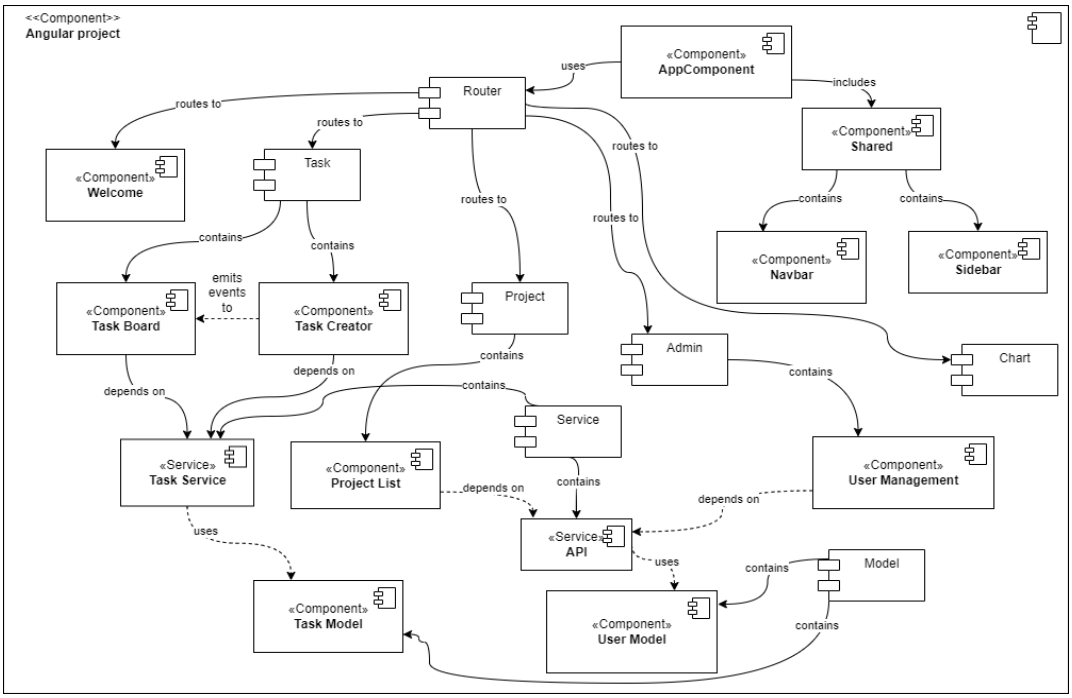


Рис. 3. Діаграма компонентів фронтенд-частини застосунку

login.component.ts

src > app > auth > login.component.ts > LoginComponent > onSubmit

```typescript
import { Component } from '@angular/core';
import { FormBuilder, FormGroup, Validators, ReactiveFormsModule } from '@angular/forms';
import { Router } from '@angular/router';
import { AuthService } from '../services/api.service';
import { catchError } from 'rxjs/operators';
import { of } from 'rxjs';

@Component({
  selector: 'app-login',
  standalone: true,
  imports: [ReactiveFormsModule],
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent {
  loginForm: FormGroup;
  errorMessage: string = '';

  constructor(private fb: FormBuilder, private authService: AuthService, private router: Router) {
    this.loginForm = this.fb.group({
      email: ['', [Validators.required, Validators.email]],
      password: ['', Validators.required],
    });
  }

  onSubmit() {
    if (this.loginForm.valid) {
      const { email, password } = this.loginForm.value;
      this.authService.login(email, password).pipe(
        catchError((err) => {
          this.errorMessage = err.message;
          return of(null);
        })
      ).subscribe(res => {
        if (res) this.router.navigate(['/tasks']);
      });
    }
  }
}
```

Рис. 4. Компонент app-login.ts

task-creator.component.ts

src > app > tasks > task-creator.component.ts > ...

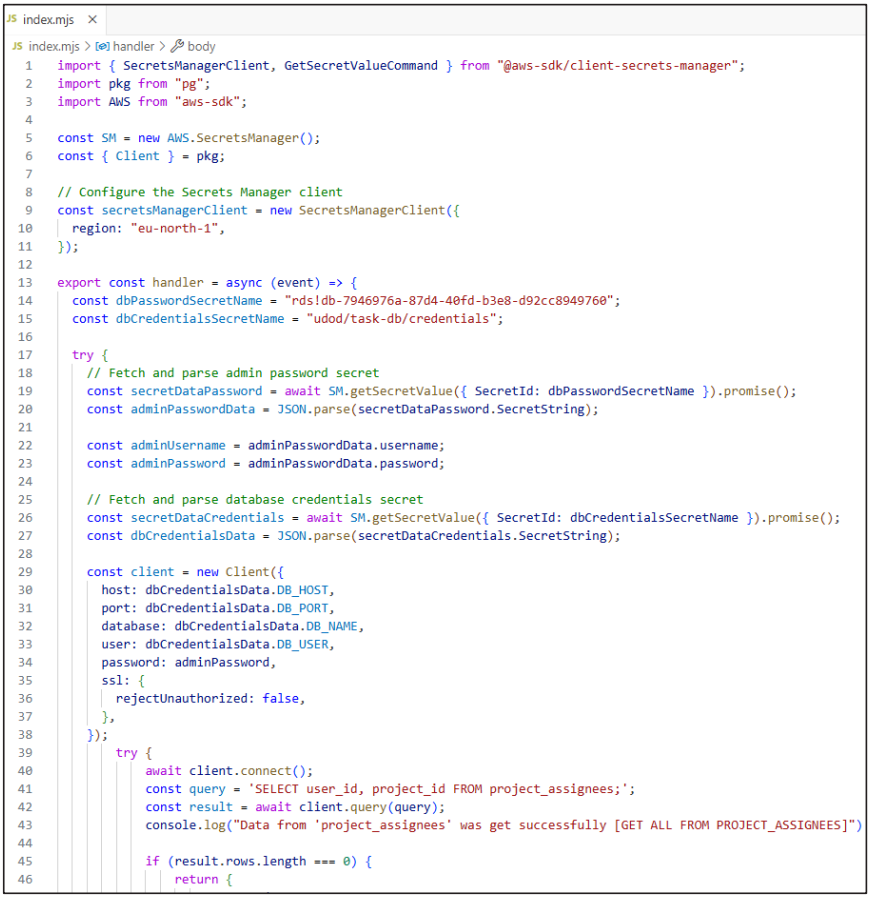
```typescript
import { Component, EventEmitter, Output } from '@angular/core';
import { FormBuilder, FormGroup, ReactiveFormsModule, Validators } from '@angular/forms';
import { Task } from '../model/task.model';

@Component({
  selector: 'app-task-creator',
  standalone: true,
  imports: [ReactiveFormsModule],
  templateUrl: './task-creator.component.html',
  styleUrls: ['./task-creator.component.css']
})
export class TaskCreatorComponent {
  @Output() taskCreated = new EventEmitter<Task>();
  form: FormGroup;

  constructor(private fb: FormBuilder) {
    this.form = this.fb.group({
      title: ['', Validators.required],
      priority: ['medium', Validators.required],
      status: ['design', Validators.required]
    });
  }

  onSubmit() {
    if (this.form.valid) {
      const task: Task = {
        id: `KAN-${Math.floor(Math.random() * 1000)}`,
        ...this.form.value
      };
      this.taskCreated.emit(task);
      this.form.reset({ priority: 'medium', status: 'design' });
    }
  }
}
```

Рис. 5. Компонент TaskCreatorComponent



```
index.mjs
index.mjs > handler > body
1  import { SecretsManagerClient, GetSecretValueCommand } from "@aws-sdk/client-secrets-manager";
2  import pkg from "pg";
3  import AWS from "aws-sdk";
4
5  const SM = new AWS.SecretsManager();
6  const { Client } = pkg;
7
8  // Configure the Secrets Manager client
9  const secretsManagerClient = new SecretsManagerClient({
10   region: "eu-north-1",
11 });
12
13 export const handler = async (event) => {
14   const dbPasswordSecretName = "rds!db-7946976a-87d4-40fd-b3e8-d92cc8949760";
15   const dbCredentialsSecretName = "udod/task-db/credentials";
16
17   try {
18     // Fetch and parse admin password secret
19     const secretDataPassword = await SM.getSecretValue({ SecretId: dbPasswordSecretName }).promise();
20     const adminPasswordData = JSON.parse(secretDataPassword.SecretString);
21
22     const adminUsername = adminPasswordData.username;
23     const adminPassword = adminPasswordData.password;
24
25     // Fetch and parse database credentials secret
26     const secretDataCredentials = await SM.getSecretValue({ SecretId: dbCredentialsSecretName }).promise();
27     const dbCredentialsData = JSON.parse(secretDataCredentials.SecretString);
28
29     const client = new Client({
30       host: dbCredentialsData.DB_HOST,
31       port: dbCredentialsData.DB_PORT,
32       database: dbCredentialsData.DB_NAME,
33       user: dbCredentialsData.DB_USER,
34       password: adminPassword,
35       ssl: {
36         rejectUnauthorized: false,
37       },
38     });
39         try {
40             await client.connect();
41             const query = 'SELECT user_id, project_id FROM project_assignees;';
42             const result = await client.query(query);
43             console.log("Data from 'project_assignees' was get successfully [GET ALL FROM PROJECT_ASSIGNEES]")
44
45             if (result.rows.length === 0) {
46                 return {
```

Рис. 6. Lambda-функція для отримання даних з PostgreSQL

```
index.mjs
index.mjs > ...
1  import { SecretsManagerClient, GetSecretValueCommand } from "@aws-sdk/client-secrets-manager";
2  import pkg from "pg";
3  import AWS from "aws-sdk";   Amazon Q Tip 1/3: Start typing to get suggestions ([ESC] to exit)
4  import jwt from "jsonwebtoken";
5  import bcrypt from "bcryptjs";
6
7  const SM = new AWS.SecretsManager();
8  const { Client } = pkg;
9
10 const secretsManagerClient = new SecretsManagerClient({ region: "eu-north-1" });
11
12 export const handler = async (event) => {
13   const dbPasswordSecretName = "rds!db-7946976a-87d4-40fd-b3e8-d92cc8949760";
14   const dbCredentialsSecretName = "udod/task-db/credentials";
15   const jwtSecretName = "udod/task-api/jwt-secret";
16
17   try {
18     const secretDataPassword = await SM.getSecretValue({ SecretId: dbPasswordSecretName }).promise();
19     const adminPasswordData = JSON.parse(secretDataPassword.SecretString);
20     const adminPassword = adminPasswordData.password;
21
22     const secretDataCredentials = await SM.getSecretValue({ SecretId: dbCredentialsSecretName }).promise();
23     const dbCredentialsData = JSON.parse(secretDataCredentials.SecretString);
24
25     const secretJwt = await SM.getSecretValue({ SecretId: jwtSecretName }).promise();
26     const jwtSecret = JSON.parse(secretJwt.SecretString).JWT_SECRET;
27
28     const { login, password } = JSON.parse(event.body);
29
30     const client = new Client({
31       host: dbCredentialsData.DB_HOST,
32       port: dbCredentialsData.DB_PORT,
33       database: dbCredentialsData.DB_NAME,
34       user: dbCredentialsData.DB_USER,
35       password: adminPassword,
36       ssl: { rejectUnauthorized: false },
37     });
38
39     await client.connect();
40
41     const query = 'SELECT user_id, login, password, role_id FROM users WHERE login = $1';
```

Рис. 7. Lambda-функція для авторизації і автентифікації користувачів

виключних ситуацій, що перехоплюють помилки підключення, збої служби секретів та внутрішні помилки виконання, забезпечуючи стабільність системи.

Система безпеки базується на використанні протоколу JWT (JSON Web Token), що забезпечує stateless-взаємодію між Angular-клієнтом та хмарним бекендом. На рисунку 7 представлено реалізацію Lambda-функції автентифікації, алгоритм роботи якої включає три ключові етапи.

Перший етап – ініціалізація, що передбачає отримання реквізитів доступу до PostgreSQL та ключів підпису JWT через захищене сховище AWS Secrets Manager. Другий етап – верифікація, під час якої здійснюється пошук користувача в базі даних та перевірка валідності пароля за допомогою алгоритму хешування bcrypt. Завершальним етапом є генерація підписаного токена, що містить рольові атрибути та термін дії сесії. Така архітектура гарантує ізоляцію конфіденційних даних та відповідність сучасним стандартам безпеки вебзастосунків.

Апробацію розробленого рішення здійснено за комплексною стратегією. Модульне та інтеграційне тестування компонентів Angular і AWS Lambda реалізовано з використанням фреймворків Jest та Mocha, а валідацію API-інтерфейсів – засобами Postman. Для оцінки нефункціональних вимог проведено навантажувальне тестування за допомогою інструменту Artillery.

Результати випробувань продуктивності наведено на рисунку 8. Аналіз графіків пропускної здатності та часу відгуку свідчить про стабільність системи під навантаженням: протягом тестового інтервалу (60 с) зафіксовано стійку обробку запитів з інтенсивністю 20 RPS при збереженні нормативних показників латентності. Отримані дані підтверджують ефективність обраної архітектури та її здатність до масштабування.

На рисунку 9 наведено реалізовану дошку задач проєкту. Ліва частина інтерфейсу містить навігаційне меню, яке дозволяє швидко переходити між основними розділами проєкту: головна, завдання, issue boards, код, тестування, аналітика тощо.

Центральна частина інтерфейсу поділена на три стовпці: Open, In Progress та Closed, які представляють поточний стан задач у проєкті. Кожне

Рис. 8. Тестування навантажень

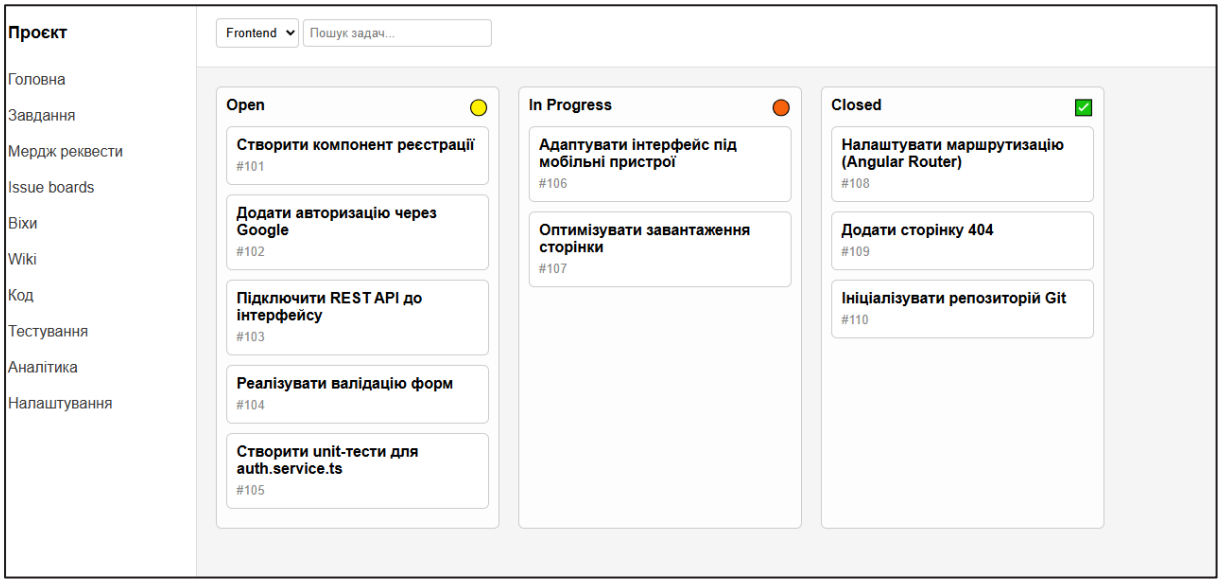


Рис. 9. Дошка завдань

завдання оформлено окремою карткою з коротким описом та унікальним номером. Наприклад, у стовпці «Open» є завдання «Створити компонент реєстрації», а в стовпці «Closed» – завершене завдання «Налаштувати маршрутизацію (Angular Router)». Такий формат дозволяє ефективно візуалізувати робочий процес команди та контролювати виконання задач на різних етапах розробки.

Висновки. У роботі представлено розробку системи управління завданнями на основі безсерверної архітектури. Аналіз аналогів та функціональних вимог дозволив обґрунтувати вибір стека технологій Angular та AWS Lambda як оптимального рішення для забезпечення масштабованості та мінімізації експлуатаційних витрат. У рамках дослідження спроєктовано та реалізовано надійну архітектуру із використанням сервісів AWS (RDS, API Gateway, Secrets Manager), що забезпечує високу доступність даних. Експериментальна перевірка, проведена шляхом навантажувального тестування, підтвердила стабільність роботи системи при інтенсивності 20 запитів за секунду при збереженні стабільних часових характеристик. Отримані результати свідчать про те, що запропонований підхід дозволяє створювати ефективні, відмовостійкі та економічно вигідні системи управління проєктами. Подальший розвиток системи передбачає інтеграцію алгоритмів машинного навчання для автоматизованого розподілу навантаження та прогнозування термінів виконання завдань. Крім того, перспективним напрямком є впровадження технології WebSockets для забезпечення синхронізації даних між учасниками проєкту в режимі реального часу.

Список літератури:

1. Kosur P. Optimizing Scalability and Performance in AWS Lambda: An InDepth Analysis of Best Practices, Case Studies, and Challenges in Serverless Architectures. *Journal of Engineering and Applied Sciences Technology*. 2023. Volume 5(3). P. 1–3. DOI: 10.47363/JEAST/2023(5)E115

2. Топ 10 систем управління проектами для України. URL: https://www.livebusiness.com.ua/ua/tools/pm/ (дата звернення: 11.11.2025).

3. 25 Best Task Management Software Reviewed For 2025. URL: https://thedigitalprojectmanager.com/tools/best-task-management-software/ (дата звернення: 10.11.2025).

4. Pawłowski P., Plechawska-Wójcik M. A comparative analysis of tools dedicated to project management. *Journal of Computer Sciences Institute*. 2022. №24. P. 258–264. DOI: 10.35784/jcsi.3001

5. Day B. Wrike Review: Features, Pros And Cons. *Forbes Advisor*. URL: https://www.forbes.com/advisor/business/software/wrike-review/ (дата звернення: 10.11.2025).

6. Tammy. Jira Pros and Cons in 2025: Features, Pricing & User Reviews. *ONES.com* | Advanced software development management platform for enterprise. URL: https://ones.com/blog/jira-pros-and-cons-in-2025 (дата звернення: 10.11.2025).

7. OpenProj – Project Management. *SourceForge*. URL: https://sourceforge.net/projects/openproj/ (дата звернення: 10.11.2025).

8. What is Angular? URL: https://angular.dev/overview (дата звернення: 10.11.2025).

9. Розуміння зв'язування даних в Angular: Ключ до динамічних веб-додатків – javascript.org.ua – JS Communities. *javascript.org.ua – JS Communities*. URL: https://javascript.org.ua/rozuminnya-zvyazuvannya-danih-v-angular-klyuch-do-dinamichnih-veb-dodatkiv/ (дата звернення: 10.11.2025).

10. Shields K. Angular Development Pros and Cons: Is It Right for Your Web Application. Mobile App & Web Development Greenville, SC | *Designli*. URL: https://designli.co/blog/angular-development-web-application (дата звернення: 10.11.2025).

11. AWS Lambda: Serverless Computing Explained – AWS. URL: https://aws.amazon.com/awstv/watch/64bb50cff05/ (дата звернення: 10.11.2025).

12. What is Amazon VPC? – Amazon Virtual Private Cloud. URL: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html (дата звернення: 10.11.2025).

13. What is Amazon EC2? – Amazon Elastic Compute Cloud. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html (дата звернення: 10.11.2025).

14. Hosted PostgreSQL – Amazon RDS for PostgreSQL – AWS. *Amazon Web Services, Inc*. URL: https://aws.amazon.com/rds/postgresql/ (дата звернення: 10.11.2025).

15. Welcome to AWS Documentation. URL: https://docs.aws.amazon.com/ (дата звернення: 10.11.2025).

Udod O.S., Kryvokhata A.G. DEVELOPMENT OF A HIGHLY SCALABLE SERVERLESS CLOUD ARCHITECTURE FOR A TASK MANAGEMENT SYSTEM

The article addresses the relevant scientific and applied problem of improving the development and operational efficiency of task management systems under conditions of dynamic and fluctuating workloads. The shortcomings of traditional monolithic and server-based architectures, associated with limited scalability and high infrastructure maintenance costs, are analyzed. The utilization of the Serverless Computing concept and the Function as a Service (FaaS) model is proposed. This approach, unlike existing solutions, enables resource optimization and automatic horizontal scaling.

To define the requirements, a comparative analysis of leading commercial (Wrike, JIRA) and open-source (OpenProj) analogs was conducted. A set of functional and non-functional requirements for the system was formulated, with a focus on performance, data security, and fault tolerance indicators. A set of UML diagrams (use case, component, and deployment diagrams) was developed to illustrate the structure of the distributed web application.

The choice of the technology stack is substantiated: the Angular framework for the client-side implementation (Single Page Application) and the Amazon Web Services ecosystem (Lambda Node.js, API Gateway, RDS PostgreSQL) for the backend. Particular attention is paid to information security issues: a mechanism for secure credential management via AWS Secrets Manager and stateless authentication has been implemented.

Empirical verification of the developed solution was conducted through multi-level testing. The application of unit tests (Jest) and load testing (Artillery) enabled the assessment of system behavior under stress loads. Experimental results confirmed that the proposed architecture ensures stable processing of up to 20 requests per second while maintaining stable timing characteristics. The obtained data indicate that the developed system is an effective, cost-efficient, and scalable solution, ready for implementation in a corporate environment for project management automation.

Key words: *computer information technologies, task management system, serverless architecture, Angular, AWS Lambda, PostgreSQL, API Gateway, JWT, load testing.*

Дата надходження статті: 28.11.2025
Дата прийняття статті: 17.12.2025
Опубліковано: 30.12.2025