УДК 004.421 004.5 DOI https://doi.org/10.35546/kntu2078-4481.2025.3.2.45

В. К. ОВСЯК
доктор технічних наук,
професор кафедри комп'ютерних технологій
у видавничо-поліграфічних процесах
Національний університет «Львівська політехніка»
ORCID: 0000-0001-9295-284X

В. Р. ТУРЧАК
аспірант кафедри комп'ютерних технологій
у видавничо-поліграфічних процесах
Національний університет «Львівська політехніка»
ORCID: 0009-0009-2907-1869

# МОДЕЛЬ АЛГОРИТМУ ВЗАЄМОДІЇ КОРИСТУВАЧА З БАЗОЮ СТРУКТУРОВАНИХ ДАНИХ ДЛЯ АДАПТИВНОГО СОРТУВАННЯ У ВЕБІНТЕРФЕЙСІ

*Сортування табличних даних у веб-середовищі потребує рішень, що поєднують практичну інтерактивність з математичною визначеністю. У статті запропоновано методику та формалізовану модель клієнтського інтерфейсу, який імпортує Excel/CSV, виконує очищення й нормалізацію, підтримує фільтри й багатокритеріальне впорядкування без обов'язкової серверної логіки. Реалізовано два робочі режими: автономний (усі операції виконуються у браузері, локальне зберігання) та інтеграційний (опційний REST-експорт у MySQL/MongoDB/SQLite). Інтерфейс орієнтовано на покрокову взаємодію: вікна завантаження, параметрів очищення, вибору критеріїв/пріоритетів, налаштувань фільтрів і часткового збігу, запуску сортування та перегляду результатів; окремо передбачено логування дій і підказки щодо конфліктних умов. Технічно використано HTML, CSS, JavaScript, IndexedDB/sql.js та бібліотеку XLSX, що забезпечує доступність і простоту розгортання у локальних середовищах.*

*Ідейне ядро складає трикомпонентна архітектура даних: $T_1$ (вхідна матриця $m \times n$), $T_2$ (набір четвірок ($c_i$, $w_i$, $d_i$, $f_i$) для формалізації критеріїв і ваг) та $T_3$ (результат після фільтрації/проєкції/сортування). Кожний етап має чіткі оператори: $T_1 \to f_1 \to T_1'$ (очищення/нормалізація/валідація), $T_1' \to f_2 \to T_2$ (побудова запиту), $T_1', T_2 \to f_3 \to T_3$ (застосування інструкції). Зіставлення текстових полів підтримує нечіткість (зокрема, Левенштейн), що підвищує стійкість до варіантів написання. Для багатокритеріального порівняння використано* $Score(r) = \sum_k w_k \cdot \varphi_k(r, c_k, f_k)$ *або лексикографічний ключ, де $\varphi_k$ – нормалізована функція порівняння з урахуванням фільтра.*

*Сортування реалізовано модифікованим Selection Sort із динамічно-наслідковим обмеженням переміщення $R = m/(i + 1)$, що на кроці і лімітує інтенсивність перестановок ($\leq \{R\}$) і запобігає «стрибкам» порядку; нестабільність базового алгоритму компенсується tie-break за початковим індексом. Логіку взаємодії формалізовано як скінченний автомат: стани відповідають вікнам, переходи – подіям користувача; реалізовано валідацію введення та попередження конфліктів, а зміни параметрів реактивно оновлюють $T_3$. Така побудова забезпечує прозорість, відтворюваність і можливість генерації контрольних сценаріїв.*

*Експериментальні випробування на переліках параметричних об'єктів (охоронні сповіщувачі) засвідчили коректність перетворень, стабілізацію порядку у складних запитах, зручність покрокової роботи і ресурсну помірність клієнтського виконання. У підсумку модель і програмна реалізація поєднують інтерактивність інтерфейсу з формальною строгістю даних, забезпечують автономну роботу та опційну інтеграцію через REST-API, а також демонструють практичну придатність для аналітики, навчання й адміністративних сценаріїв у CRM/ERP-контекстах. Наукова новизна полягає в тому, що запропоновано повністю клієнтський цикл обробки (імпорт → очищення/нормалізація → фільтрація/проєкція → сортування → відображення) без обов'язкового сервера; формалізовано трикомпонентну схему $T_1$–$T_2$–$T_3$ ($T_1$ – вхідні дані, $T_2$ – «інструкція» у вигляді четвірок ($c_i$, $w_i$, $d_i$, $f_i$), $T_3$ – результат), що робить налаштування критеріїв і ваг прозорим та відтворюваним. Порівняно з аналогами, розроблений інтерфейс вирізняється інтерактивністю, підтримкою текстових запитів і чіткою формалізацією логіки взаємодії, що підвищує його конкурентоспроможність. Додатково система забезпечує автономність роботи (client-side), прозору верифікацію кроків і відтворюваність завдяки моделюванню інтерфейсу як скінченного автомата та композиції операторів $T_1 \to f_1 \to T_1' \to f_2 \to T_2 \to f_3 \to T_3$. Модифікований Selection Sort з обмеженням $R = m/(i + 1)$ стабілізує порядок за багатокритеріальних запитів, а підтримка REST-експорту і коректного перетворення типів спрощує інтеграцію з БД (MySQL/MongoDB/SQLite) та масштабування рішення.*

***Ключові слова:*** *структуровані дані, впорядкована матриця, вебінтерфейс, адаптивне сортування, алгоритм Selection Sort, нечітке порівняння, відстань Левенштейна, база даних, SheetJS, REST API, JavaScript, скінченний автомат.*

V. K. OVSIAK
Doctor of Technical Sciences,
Professor at the Department of Computer Technologies in Publishing and Printing Processes
National University "Lviv Polytechnic"
ORCID: 0000-0001-9295-284X

V. R. TURCHAK
Postgraduate Student at the Department of Computer Technologies in Publishing and Printing Processes
National University "Lviv Polytechnic"
ORCID: 0009-0009-2907-1869

# MODEL OF THE ALGORITHM FOR USER INTERACTION WITH A STRUCTURED DATA BASE FOR ADAPTIVE SORTING IN THE WEB INTERFACE

*Sorting tabular data in a web environment requires solutions that combine practical interactivity with mathematical rigor. This article proposes a methodology and a formalized model of a client-side interface that imports Excel/CSV files, performs data cleaning and normalization, and supports filtering and multi-criteria ordering without mandatory server logic. Two operating modes are implemented: an autonomous mode (all operations are performed in the browser with local storage) and an integration mode (optional REST export to MySQL/MongoDB/SQLite). The interface is designed for step-by-step interaction: windows for loading, data-cleaning parameter settings, criterion/priority selection, filter and partial-match settings, sorting initiation, and result viewing; separate action logging and hints for conflict conditions are provided. Technically, the solution uses HTML, CSS, JavaScript, IndexedDB/sql.js, and the XLSX library, ensuring accessibility and ease of deployment in local environments.*

*The conceptual core is a three-component data architecture: $T_1$ (input matrix $m \times n$), $T_2$ (a set of quadruples $(c_i, w_i, d_i, f_i)$ formalizing criteria and weights), and $T_3$ (the result after filtering/projection/sorting). Each stage has clear operators: $T_1 \to f_1 \to T_1'$ (cleaning/normalization/validation), $T_1' \to f_2 \to T_2$ (query construction), and $T_1', T_2 \to f_3 \to T_3$ (instruction application). Matching of text fields supports fuzziness (in particular, the Levenshtein distance), which increases robustness to spelling variants. For multi-criteria comparison, either $Score(r) = \sum_k w_k \cdot \varphi_k(r, c_k, f_k)$ or a lexicographic key is used, where $\varphi_k$ is a normalized comparison function that accounts for the filter.*

*Sorting is implemented as a modified Selection Sort with a dynamic consequential displacement constraint $R = m/(i + 1)$, which at step i limits the intensity of permutations ($\leq R$) and prevents jumps in order; the instability of the base algorithm is compensated by a tie-break on the initial index. The interaction logic is formalized as a finite automaton: states correspond to windows, transitions to user events; input validation and conflict warnings are implemented, and parameter changes reactively update T3. This design ensures transparency, reproducibility, and the ability to generate control scenarios.*

*Experimental tests on lists of parametric objects (security detectors) confirmed the correctness of transformations, stabilization of order in complex queries, convenience of step-by-step work, and moderate resource use in client-side execution. As a result, the model and software implementation combine interface interactivity with formal data rigor, provide autonomous operation with optional integration via REST API, and demonstrate practical suitability for analytics, education, and administrative scenarios in CRM/ERP contexts. The scientific novelty lies in proposing a fully client-side processing cycle (import → cleaning/normalization → filtering/projection → sorting → rendering) without a mandatory server; the three-component $T_1$–$T_2$–$T_3$ scheme is formalized ($T_1$ – input data; $T_2$ – the "instruction" in the form of quadruples (ci, wi, di, fi); $T_3$ – the result), which makes the tuning of criteria and weights transparent and reproducible. Compared to analogs, the developed interface stands out for its interactivity, support for text queries, and clear formalization of interaction logic, which enhances its competitiveness. Additionally, the system ensures client-side autonomy, transparent step verification, and reproducibility thanks to modeling the interface as a finite automaton and composing the operators $T_1 \to f_1 \to T_1' \to f_2 \to T_2 \to f_3 \to T_3$. The modified Selection Sort with the constraint $R = m/(i + 1)$ stabilizes order under multi-criteria queries, while support for REST export and correct type conversion simplifies integration with databases (MySQL/MongoDB/SQLite) and scaling.*

***Key words:*** *structured data, ordered matrix, web interface, adaptive sorting, Selection Sort, fuzzy matching, Levenshtein distance, database, SheetJS, REST API, JavaScript, finite automaton.*

## Постановка проблеми

Упорядкування структурованих табличних даних у веб-інтерфейсі набуває особливої ваги в аналітиці, логістиці, освіті та адміністративних системах, де цінуються автономність, прозорість і відтворюваність дій користувача. Більшість поширених рішень орієнтовано на серверну логіку або десктопні інструменти, що ускладнює роботу в закритих мережах і навчальних середовищах та не завжди забезпечує керовану, пояснювану послідовність обробки.

*Проблема полягає* у відсутності уніфікованого клієнтського підходу до інтерактивної обробки табличних даних із прозорою формалізацією кроків (від імпорту до відображення), стабільним багатокритеріальним сортуванням і формально визначеною логікою інтерфейсу. На практиці це проявляється в труднощах відтворення результатів, непередбачуваних «стрибках» порядку під час складних запитів та обмеженій підтримці текстових критеріїв.

*Об'єкт дослідження*: взаємодія користувача зі структурованими табличними даними у веб-інтерфейсі.

*Предмет дослідження*: алгоритмічна модель клієнтської обробки та аналізу табличних даних без обов'язкової серверної логіки.

**Аналіз останніх досліджень і публікацій**

Упродовж останніх років було опубліковано низку праць, присвячених обробці структурованих даних, сортуванню за множиною критеріїв, а також реалізації інтерактивної взаємодії з табличними структурами у вебсередовищі. Проте ці підходи зазвичай не поєднують повноцінну математичну формалізацію із практичною реалізацією без серверної логіки. Важливо також враховувати етапи очищення та підготовки даних, які є ключовими для подальшої обробки [1].

Ще у класичній роботі [2] було запропоновано методи інтеграції табличних даних із мінімальними витратами зусиль, однак питання побудови адаптивного алгоритму сортування й інтеграції з базами даних там не розглядалося. Низка клієнтських бібліотек (SheetJS, PapaParse, Handsontable) пропонують компоненти для відображення та базового редагування у браузері, але вони не мають формальної моделі перетворення запиту користувача у результатну множину ($T_1$, $T_2$, $T_3$), тому їх слід розглядати лише як допоміжні інструменти [3–5].

Моделі інтерактивної візуалізації напівструктурованих даних, як-от [6], орієнтовані переважно на графічне представлення та профілювання даних без повноцінної логіки багатокритеріального сортування або нечіткого порівняння текстових полів [7], [8]. Для побудови формальної логіки інтерфейсу як автомата доцільно спиратися на Statecharts [9]. У [10] представлено веб-додаток PROMETHEE-Cloud для підтримки багатокритеріального аналізу на основі PROMETHEE, однак без побудови структури вебінтерфейсу та формалізації перетворення вихідних таблиць у вигляді впорядкованих кортежів і схеми $T_1 \to T_2 \to T_3$.

У контексті багатокритеріального аналізу, робота [11] демонструє підхід до групового ранжування, але не містить реалізації у вебінтерфейсі. Аналогічно, стаття [12] зосереджується на теоретичних засадах багатокритеріальної вагової агрегації без розгляду технічних аспектів інтеграції в динамічні клієнтські таблиці. Огляд сучасних підходів до багатокритеріальної (MCDM) агрегації (WSM/SAW, TOPSIS, PROMETHEE та ін.) наведено в [13].

Також заслуговують на увагу підходи з формалізацією фільтрації та ранжування текстових записів, що використовують моделі на основі нечіткого порівняння [7], [8]. У роботі [14] реалізовано групове оцінювання ризиків у середовищі електронної комерції, однак модель не охоплює перетворення таблиць за запитом користувача або збереження результатів у базі даних.

Окремо слід виділити дослідження, присвячені архітектурі збереження даних. У [15] і [16] проаналізовано можливості використання MongoDB та MySQL у різних сценаріях, що демонструє потенціал для побудови вебінтерфейсу з підтримкою REST API та експорту результатів у базу даних.

Таким чином, існує розрив між теоретичними підходами до багатокритеріального сортування та практичними реалізаціями інтерфейсної взаємодії. Наскільки нам відомо, у відкритій літературі бракує робіт, що одночасно забезпечують:

– клієнтську реалізацію без серверної логіки;
– чітку математичну структуру таблиць як впорядкованих матриць або $n$-кортежів;
– перетворення $T_1 \to T_2 \to T_3$ відповідно до запиту користувача;
– підтримку нечіткого порівняння текстових полів;
– збереження результатів у базу даних через REST API.

Це вказує на потребу у побудові цілісної моделі, яка поєднує всі ці компоненти – від валідації та очищення даних [1] до експорту результатів – у межах єдиного вебінтерфейсу з математичною формалізацією.

**Формулювання мети дослідження**

Мета дослідження: розробити формалізовану клієнтську модель і модуль веб-інтерфейсу, що забезпечують повний цикл обробки на боці браузера – імпорт, очищення/нормалізацію, фільтрацію/проєкцію, адаптивне сортування, візуалізацію та (за потреби) інтеграційний експорт – із підтримкою текстових критеріїв і відтворюваних кроків.

Для досягнення поставленої мети потрібно розв'язати такі завдання:

1. Проаналізувати сучасні методи сортування табличних даних у веб-інтерфейсах; оцінити можливості та обмеження бібліотек SheetJS/XLSX, PapaParse, Handsontable щодо покрокового відображення, гнучкої фільтрації та клієнтської обробки без сервера.

2. Сформалізувати й реалізувати клієнтську модель: трикомпонентну структуру $T_1$–$T_2$–$T_3$ з операторами $T_1 \to f_1 \to T_1' \to f_2 \to T_2 \to f_3 \to T_3$ і запитом у вигляді четвірок $(c_i, w_i, d_i, f_i)$; адаптивне сортування (модифікований Selection Sort з обмеженням $R = m/(i + 1)$ та tie-break), підтримку нечіткого зіставлення (відстань Левенштейна); інтерфейс як скінченний автомат (вікна/події/стани) з валідацією та реактивним оновленням $T_3$.

3. Провести експериментальну верифікацію на переліках охоронних сповіщувачів: оцінити швидкодію, стабілізацію порядку (зменшення «стрибків» завдяки $R$), зручність і гнучкість порівняно з аналогами; підсумувати переваги формалізованої структури та покрокового режиму.

**Викладення основного матеріалу дослідження**

У процесі дослідження було застосовано сукупність методів, що охоплюють математичне моделювання, теорію алгоритмів, структурне подання даних та інженерію інтерфейсів. Такий підхід дозволив не лише формально описати структуру табличних даних, а й забезпечити практичну реалізацію алгоритмів обробки у вебінтерфейсі без використання серверної логіки. Послідовність викладу побудовано «знизу догори»: спочатку фіксуємо, що саме є об'єктом обробки (структура рядка і таблиці), далі – як ці об'єкти трансформуються операторами конвеєра, і зрештою – як це втілюється в інженерній реалізації. Нижче описано послідовність методів із формалізацією та поясненнями:

1. *Метод впорядкованих n – кортежів*. У цьому пункті використовуємо кортежний опис для формального подання структури кожного елемента таблиці як об'єкта з фіксованими властивостями. Зокрема, рядок таблиці описується як $r_i = (a_{i,1}, a_{i,2}, \ldots, a_{i,n})$, $i = 1\ldots m, j = 1\ldots n$, де $a_{i,j} \in S_j$ – значення атрибуту з відповідного домену $S_j$. Такий підхід дозволяє визначити таблицю як скінченну множину впорядкованих кортежів:

$$T_1 = \{r_1, r_2, \ldots, r_m\}, r_i \in S_1 \times S_2 \times \ldots \times S_n. \quad (1)$$

Таким чином, ми одержуємо строгу типізацію кожного поля й основу для коректних перетворень над множинами рядків. У наступному пункті цю саму структуру подамо в матричній формі, що забезпечить наочність позицій елементів і підготує ґрунт для векторизованих операцій та подальшого ланцюга перетворень.

2. *Матричне представлення таблиць*. У цьому пункті формалізуємо структуру всієї таблиці як матрицю

$$M = \{a_{i,j}\}, \quad i = 1\ldots m, \quad j = 1\ldots n, \quad a_{i,j} \in S_j,$$

де $M[i][j]$ – значення $j$-го атрибута в $i$-му рядку. Відображення між $n$ – кортежем та матрицею є двонаправленим і взаємно однозначним у межах заданої структури таблиці:

$$M = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \ldots & a_{m,n} \end{pmatrix}. \quad (2)$$

Тобто кожен кортеж $r_i$ повністю відповідає $i$-му рядку матриці:

$$T_1 = \{r_i = (a_{i,1}, \ldots, a_{i,n})\} \frac{m}{i=1}. \quad (3)$$

Далі маючи структурні представлення (кортежне та матричне), у наступному пункті опишемо процесні перетворення – поетапні оператори, що ведуть від $T_1$ до $T_3$.

3. *Функціональне перетворення таблиць*. У цьому пункті показано, що перетворення у системі багатокритеріального сортування реалізується не як єдина операція, а як поетапна послідовність дій над множинами рядків. Кожний етап визначається відповідним оператором $f_k$, що забезпечує перехід від однієї таблиці до наступної. У найзагальнішій формі цей процес можна подати як ланцюг таких перетворень:

$$T_1 \xrightarrow{f_1} T_1' \xrightarrow{f_2} T_2 \xrightarrow{f_3} T_3, \quad (4)$$

де: $T_1$ – початкова таблиця, що виступає вхідними даними. Наприклад, це може бути перелік датчиків безпеки з базовими характеристиками (назва, ціна, дальність виявлення тощо). На цьому етапі дані ще не оброблені і можуть містити дублікати, помилки чи зайву інформацію; $f_1$ – оператор очищення/нормалізації/валідації. Він призначений для усунення дублікатів записів, виправлення або видалення помилкових рядків, а також для приведення даних до узгодженої структури. У результаті його застосування формується проміжна таблиця $T_1'$, яка вже має більш упорядкований вигляд, проте ще не відповідає запиту користувача; $f_2$ – оператор формування запиту. На цьому етапі не відбираються дані, а специфікуються критерії, задані користувачем: визначаються атрибути для фільтрації/сортування, напрями (↑/↓), ваги та предикати. Результатом є таблиця-інструкція $T_2 = \{(c_i, w_i, d_i, f_i)\} \frac{k}{i=1}$, що описує очікувану обробку даних без зміни вмісту $T_1'$; безпосередні фільтрація, проєкція та сортування виконуються на етапі $f_3$; $f_3$ – оператор застосування запиту (фільтрація, проєкція, сортування). Він послідовно виконує: фільтрацію рядків $T_1'$ за предикатами з $T_2$ (наприклад, «радіус ≥ порогу», «ціна ∈ [$a$; $b$]»), проєкцію на релевантні стовпці та впорядкування за обраними критеріями й напрямами (наприклад, за спаданням точності чи зростанням ціни; за рівності застосовується лексикографічний/стабільний tie-break). У результаті формується фінальна таблиця $T_3$, що є відображенням застосованого запиту та слугує підсумковим результатом усього процесу.

Таким чином, кожен крок у наведеній послідовності є логічно завершеною операцією, а разом вони утворюють цілісний процес перетворення вхідних даних у структурований і впорядкований результат.

В цьому пункті ми зафіксували конвеєр перетворень. Водночас коректність кроку $f_3$ для текстових полів істотно залежить від стійкого механізму порівняння значень: у реальних даних часто зустрічаються варіанти на кшталт «IR»/«ІЧ»/«інфрачервоний». Тому в наступному пункті ми формалізуємо нечітке порівняння на основі відстані Левенштейна та порогів θ, τ, що підвищує надійність фільтрації й сортування.

4. *Метод нечіткого порівняння текстових полів*. Як зазначалося вище, він застосовується у випадках, коли значення атрибутів у різних рядках таблиці можуть відрізнятися незначними синтаксичними похибками (наприклад, різницею у літері, пропущеним символом або варіацією у написанні). Для формалізації такої подібності використовується метрика Левенштейна $d_L(s_1, s_2)$, яка визначає мінімальну кількість елементарних операцій (вставка, видалення, заміна символа), необхідних для перетворення рядка $s_1$ у рядок $s_2$:

$$d_L(s_1, s_2) \le \theta \Rightarrow s_1 \approx s_2. \tag{5}$$

де $d_L$ – відстань Левенштейна між рядками $s_1$ і $s_2$, а θ – допустиме відхилення, яке визначає максимальну кількість допустимих змін символів; у практичних експериментах воно зазвичай становить 1–2.

Таким чином, якщо значення двох текстових полів відрізняються лише на одну чи дві елементарні операції, вони вважаються еквівалентними з точки зору подальшого сортування та групування. Це дозволяє коректно обробляти дублікати та однорідні записи навіть у випадку наявності орфографічних помилок чи незначних розбіжностей у написанні. У результаті алгоритм забезпечує стійкість до «шуму» у даних та підвищує якість формування підсумкової таблиці [7], [8]. З огляду на викладене, у наступному пункті продемонстровано їх реалізацію у клієнтському вебінтерфейсі.

5. *Інженерна реалізація*. Вона передбачає виконання всіх основних операцій – зчитування вхідних таблиць, обробки даних, сортування та відображення результатів – виключно на стороні клієнта. Для цього використовується JavaScript у поєднанні з бібліотеками SheetJS/PapaParse/Handsontable [3–5]: SheetJS забезпечує парсинг табличних файлів (XLSX/CSV), PapaParse – стрімінгове опрацювання CSV, Handsontable – відображення та інтерактивне редагування даних у гріді. HTML5 разом із модальними UI-компонентами формує інтерактивну взаємодію в браузері. Для збереження та подальшого використання результатів реалізовано опційний механізм передання даних у зовнішні бази (MongoDB, MySQL, SQLite) через REST API, який виконує лише функцію експорту і не залучається до сортування/обробки [15], [16]. За відсутності серверної інфраструктури підтримується локальний експорт у файли (CSV/JSON) та/або збереження в IndexedDB. У наступному пункті, щоб керувати обробкою з інтерфейсу й легко повторювати її між сесіями, вводимо таблицю запиту $T_2$ – короткий набір правил ($c_i$, $w_i$, $d_i$, $f_i$), що визначає, як на кроці $f_3$ виконуються фільтрація, проєкція та сортування.

6. *Метод інтеграції таблиць із запитом користувача*. У цьому пункті формалізуємо таблицю запиту $T_2$, яка виступає посередником між початковою множиною даних та алгоритмом багатокритеріального сортування У таблицю $T_2$ заносяться критерії фільтрації, правила сортування та вагові коефіцієнти, що визначають пріоритетність кожного критерію. Користувач може задати ці параметри як вручну, так і за допомогою спеціалізованих форм введення. Формальна модель таблиці $T_2$ описується у вигляді:

$$T_2 = (g_i)\frac{k}{i=1}, \quad g_i = (c_i, w_i, d_i, f_i), \tag{6}$$

де $c_i$ – індекс(або унікальна назва) атрибута в $T_1$, а $w_i \in [0; 1]$ – ваговий коефіцієнт, що відображає його відносну важливість у процесі прийняття рішення, $d_i$ – напрямок сортування (↑/↓), $f_i$ – фільтр або предикат для відбору значень атрибуту $c_i$. Таким чином, таблиця $T_2$ узгоджує користувацький запит із системною моделлю, та однозначно задає, що саме і як застосовується на етапі $f_3$ (фільтрація → проєкція → сортування).

Отже, $T_2$ – це список правил, за якими виконуємо фільтрацію, проєкцію та сортування. Завдяки $T_2$ один і той самий запит легко повторити між сесіями, а весь процес залишається прозорим і перевірюваним.

У наступному розділі формально задамо структуру $T_1$, $T_2$ і $T_3$ у вигляді кортежів та матриць, покажемо їхні взаємозв'язки та наведемо приклади застосування.

**Формалізацію структури даних у моделі**

У запропонованій моделі обробки структурованих табличних даних розглядається система таблиць: вхідна таблиця даних $T_1$, таблиця користувацького запиту $T_2$ та результатна таблиця $T_3$. Кожна з них має чітко формалізовану структуру у вигляді множин впорядкованих кортежів або матриць. Далі послідовно розглянемо складові $T_1 \to T_2 \to T_3$. Почнімо з джерела даних – вхідної таблиці $T_1$.

1. *Вхідна таблиця* $T_1$

Таблиця $T_1$ містить початкову множину записів, що імпортується з Excel/CSV-файлу. Формально:

$$T_1 = \{r_i\}_{i=1}^{m}, \quad r_i = (a_{i,1}, a_{i,2}, \ldots, a_{i,n}), \tag{7}$$

де: $r_i$ – $i$-й рядок таблиці; $a_{i,j} \in S_j$ – значення $j$-го атрибута з відповідного домену $S_j$ (наприклад, назва, ціна, дальність, кут огляду, тип); $S_j$ – множина допустимих значень для атрибута $j$; $m$ – кількість рядків (об'єктів); $n$ – кількість полів (атрибутів).

У матричному вигляді:

$$M_1 = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{pmatrix}. \quad (8)$$

Для наочності доповнимо формальне подання коротким фрагментом $T_1$, приклад наведено в таблиці 1:

Таблиця 1

**Приклад вхідної таблиці $T_1$**

| Назва | Ціна | Дальність | Кут огляду | Тип |
|---|---|---|---|---|
| Датчик-001 | 500 | 12 | 90 | ІЧ |
| Датчик-002 | 700 | 8 | 100 | Радіо |
| … | … | … | … | … |
| Датчик-00n | 450 | 10 | 95 | ІЧ/радіо |

Отже, ми зафіксували структуру та типи полів у $T_1$. (наприклад, числові для ціни та дальності, категоріальні для типу) У наступному пункті формалізуємо наміри користувача у вигляді інструкції $T_2$, що задає фільтрацію, проєкцію та сортування.

2. *Таблиця користувацького запиту $T_2$*

Модель запиту користувача відображає його очікування: які критерії враховувати, у якому порядку сортувати, які фільтри застосовувати. Таким чином, $T_2$ є компактною специфікацією критеріїв ($c_i$, $w_i$, $d_i$, $f_i$), яка робить застосування кроку $f_3$ відтворюваним і прозорим.

$$T_2 = \{g_i\}_{i=1}^{k},\ g_i = (c_i, w_i, d_i, f_i), \quad (9)$$

де: $c_i \in \{1, \dots, n\}$ – індекс атрибута в $T_1$; $w_i \in [0; 1]$ – вага критерію; $d_i \in \{\uparrow, \downarrow\}$ – напрямок сортування; $f_i$ – логіка фільтрації (наприклад, «менше ніж 10», «тип = 'ІЧ'»).

У матричному вигляді:

$$M_2 = \begin{pmatrix} c_1 & w_1 & d_1 & f_1 \\ c_2 & w_2 & d_2 & f_2 \\ \vdots & \vdots & \vdots & \vdots \\ c_k & w_k & d_k & f_k \end{pmatrix}. \quad (10)$$

Приклад навелено в таблиці 2:

Таблиця 2

**Приклад користувацького запиту $T_2$**

| Критерій | Вага | Напрямок | Фільтр |
|---|---|---|---|
| Ціна | 0,5 | ↓ | ≤ 1000 |
| Тип | 0,3 | – | = ІЧ |
| Дальність | 0,2 | ↑ | ≥ 10 |

Маючи сформовану інструкцію $T_2$, переходимо до її застосування до очищеної таблиці $T_1'$ та отримання результатної таблиці $T_3$.

3. *Результуюча таблиця $T_3$*

На цьому етапі застосовуємо $T_2$ як керівну інструкцію до $T_1'$ (тут і далі $T_1'$ – очищена/приведена таблиця після оператора $f_1$), послідовно виконуємо фільтрацію → проєкцію → сортування й отримуємо $T_3$, що зберігає структуру $T_1$, але містить лише відібрану підмножину записів, упорядковану за критеріями $T_2$.

$$T_3 = sort_{T2}(\pi_{T_2}(filter_{T2}(T_1'))). \quad (11)$$

Таким чином, процес має конвеєрний характер; уточнимо значення кожного оператора.

де: $T_3 \subseteq T_1'$, $filter_{T2}(T_1')$ кон'юнктивне застосування предикатів $f_i$, $\pi_{T2}$ – проєкція на релевантні атрибути, $\text{sort}_{T2}$ – процедура впорядкування відібраних рядків за інтегральною оцінкою (Score).

Щоб порівнювати рядки за кількома критеріями, вводимо інтегральну оцінку з урахуванням ваг і напрямків.

$$Score(r) = \sum_{j=1}^{k} w'_j norm_j(r[c_j]), \quad \sum_{j=1}^{k} w'_j = 1, \tag{12}$$

де: $w'_j$ – ваговий коефіцієнт критерію, а $\text{norm}_j \in [0, 1]$ – нормалізоване значення атрибута $c_j$.

Отримані значення інтегральної оцінки визначають пріоритет кожного рядка у результатній таблиці. На алгоритмічному рівні впорядкування реалізовано через модифікований Selection Sort, адаптований до динамічно-наслідкового обмеження переміщення. Введена метрика:

$$R = \frac{m}{i+1}, \quad i = 0,1\ldots,m-1 \tag{13}$$

обмежує кількість змін позицій на $i$-й ітерації, що запобігає різким стрибкам при великій кількості критеріїв та забезпечує стабільність сортування. Таким чином, математична модель (Score) визначає цільовий порядок елементів, а алгоритмічна реалізація (Selection Sort з обмеженням $R$) забезпечує його практичне досягнення.

Припустимо, що до початкової таблиці $T_1$ (табл. 1) застосовано критерії із $T_2$ (табл. 2):

– ціна ≤ 1000,
– тип = «ІЧ»,
– дальність ≥ 10.

Тоді після фільтрації та сортування формується таблиця $T_3$

Приклад наведено в таблиці 3:

Таблиця 3

**Приклад результатної таблиці $T_3$**

| Назва | Ціна | Дальність | Кут огляду | Тип | Score |
|---|---|---|---|---|---|
| Датчик-001 | 500 | 12 | 90 | ІЧ | 0.87 |

Усі інші записи з $T_1$ відсіяні на етапі фільтрації, оскільки не відповідають предикатам $f_i$. Таким чином, у результатній таблиці залишається лише один об’єкт із найвищим інтегральним балом. Отже, приклад демонструє, як інструкція $T_2$ визначає підмножину та порядок у $T_3$. Підсумуємо взаємодію таблиць узагальненою схемою конвеєра та подальшого експорту.

4. *Зв’язок між таблицями*

Схема конвеєра перетворень ($T_1 \to T_2 \to T_3$ та експорт):

$$T_1 \xrightarrow{f_1} T_1' \xrightarrow{f_2} T_2 \xrightarrow{f_3} T_3 \Rightarrow \text{експорт} \to \text{БД/Excel/JSON}$$

*Реалізація структури у програмному коді*

Модель таблиць $T_1$, $T_2$, $T_3$, описану вище, було виконано в програмній реалізації Для підтвердження коректності моделі та демонстрації практичного застосування створено клієнтський модуль на JavaScript, який виконує повний цикл: імпорт → фільтрація → нормалізація → зважене агрегування → стабільне сортування → виведення результатів. Нижче наведено фрагмент коду, що виконує сортування об’єктів $T_1$ за критеріями з $T_2$, враховуючи ваги, напрямки та фільтри.

1. *Вихідні дані* (приклади $T_1$ і $T_2$)

– Таблиця $T_1$ містить кілька записів про датчики з атрибутами: назва, ціна, дальність, кут огляду, тип.
– Таблиця $T_2$ задає три критерії: ціна, тип та дальність, кожен із вагою, напрямом сортування та умовою фільтрації.

```
const T1 = [
   { id: 1, name: ‘Датчик-001’, price: 500, range: 12, angle: 90, type: ‘ІЧ’ },
   { id: 2, name: ‘Датчик-002’, price: 700, range: 8, angle: 100, type: ‘Радіо’ },
   { id: 3, name: ‘Датчик-003’, price: 1200, range: 14, angle: 110, type: ‘ІЧ’ }
];
const T2 = [
   { field: ‘price’, weight: 0.5, direction: ‘desc’, filter: x => x <= 1000 },
   { field: ‘type’, weight: 0.3, direction: ‘asc’, filter: x => x === ‘ІЧ’ },
   { field: ‘range’, weight: 0.2, direction: ‘asc’, filter: x => x >= 10 }
];
```

2. *Етап фільтрації*

Переходимо до першого кроку конвеєра – відсіюємо записи, що не проходять умови з $T_2$; результат стане вхідним набором для подальшої нормалізації

На першому етапі ми відсіюємо записи, що не відповідають умовам користувача. Умови задаються у таблиці $T_2$ і трактуються як логічна кон'юнкція (тобто, усі предикати мають виконуватися одночасно). У коді це відображається функцією applyFilters:

```
function applyFilters(T1, T2) {
   return T1.filter(row =>
      T2.every(c => (typeof c.filter === 'function') ? c.filter(row[c.field]) : true)
   );
}
```

Таким чином, з $T_1$ формується підмножина рядків, яка задовольняє усім предикатам $f_j$. Це відповідає операції $filter_{T2}(T_1)$ у формулі (11). Отриману підмножину далі потрібно привести до спільної шкали, щоб коректно зважувати критерії.

3. *Етап нормалізації*

Далі на цьому кроці перетворюємо значення атрибутів у діапазон [0, 1], щоб можна було їх коректно зважувати та порівнювати.

Для числових полів застосовується мін–макс нормалізація:

$$u_j^{\uparrow}(x) = \frac{x - \min}{\max - \min}, \quad u_j^{\downarrow}(x) = 1 - u_j^{\uparrow}(x). \tag{14}$$

Тобто, якщо критерій зростаючий, більші значення кращі, а якщо спадаючий – більші значення гірші.

Для категоріальних полів можна задати явну утиліту $u_j$: $S_{cj} \to [0, 1]$, наприклад: «ІЧ = 1, Радіо = 0». Якщо явної утиліти немає і фільтр уже звів множину до однієї категорії (як у прикладі з «ІЧ»), утиліта буде константною (=1).

У коді цей механізм реалізовано функцією buildUtilities.

```
function normalizeWeights(T2) {
   const sum = T2.reduce((acc, c) => acc + (c.weight || 0), 0) || 1;
   return T2.map(c => ({...c, weight: (c.weight || 0) / sum }));
}
function buildUtilities(T, T2n) {
   // попередньо обчислюємо min/max для числових полів
   const stats = {};
   for (const c of T2n) {
      const col = T.map(r => r[c.field]).filter(v => typeof v === 'number');
      if (col.length) {
         const min = Math.min(...col), max = Math.max(...col);
         stats[c.field] = { min, max };
      }
   }
   // повертаємо функцію обчислення Score з урахуванням напряму (↑/↓)
   return (row) => {
      let score = 0;
      for (const c of T2n) {
         const val = row[c.field];
         let u = 0;
         if (typeof val === 'number' && stats[c.field] && stats[c.field].max !== stats[c.
field].min) {
            const { min, max } = stats[c.field];
            const up = (val - min) / (max - min);
            u = (c.direction === 'desc') ? (1 - up) : up; // ↑/↓
         } else if (typeof val === 'string') {
            // простий приклад утиліти для категорій:
            // якщо фільтр звузив множину – всі значення 1; інакше – бінарна утиліта за
найчастішою категорією
            u = 1; // у прикладі після фільтра тип = 'ІЧ'
         } else {
```

```
            // якщо нормалізація не визначена – нейтральне 0
            u = 0;
          }
          score += (c.weight || 0) * u;
        }
        return score;
    };
}
```

4. *Етап зваженого сортування*

Маючи нормалізовані значення та ваги, переходимо до обчислення інтегральної оцінки (Score) і власне впорядкування. Після нормалізації кожному рядку $r_i \in T_1$ приписується інтегральна оцінка:

$$Score(r_i) = \sum_{j=1}^{k} w'_j u_j(a_{i,c_j}), \tag{15}$$

де $w'_j$ – нормовані ваги $\left(\sum w'_j = 1\right)$.

Далі всі записи впорядковуються за цією оцінкою у спадаючому порядку. Якщо кілька рядків отримали однаковий бал, застосовується лексикографічний tie-break (за порядком критеріїв у $T_2$).

5. *Модифікований Selection Sort з обмеженням R*

Щоб уникати різких стрибків у позиціях під час багатокритеріального сортування, застосовуємо стабілізовану версію Selection Sort із динамічним обмеженням переміщення. Відповідно до (13) використовуємо

$$R = \frac{m}{i+1}, \quad i = 0,1,\ldots,m-1$$

щоб обмежити довжину переміщення елемента на ітерації $i$ (не більше $[R]$ позицій). Якщо найкращий елемент знаходиться надто далеко, ми «підтягуємо» його за кілька ітерацій короткими суміжними свапами – це зменшує різкі стрибки у впорядкуванні.

```
function selectionSortWithR(arr, compare) {
   const a = arr; // сортуємо in-place
   const m = a.length;
   for (let i = 0; i < m - 1; i++) {
      // знаходимо «найкращий» індекс у хвості
      let best = i;
      for (let j = i + 1; j < m; j++) {
         if (compare(a[j], a[best]) < 0) best = j;
      }
      if (best === i) continue;
      const R = Math.floor(m / (i + 1)); // обмеження на переміщення за крок
      const dist = best - i;
      if (dist > R) {
         // пересуваємо елемент на R позицій угору суміжними свапами
         for (let s = 0; s < R; s++) {
            const idx = best - s;
            [a[idx], a[idx - 1]] = [a[idx - 1], a[idx]];
         }
      } else {
         // звичайний свап selection sort
         [a[i], a[best]] = [a[best], a[i]];
      }
   }
   return a;
}
```

Обмеження $R$ робить алгоритм стабільним і передбачуваним навіть за складних налаштувань ваг.

6. *Повний цикл: від $T_1$ до $T_3$*

Зведемо усе разом у підсумковій функції, яка відтворює конвеєр $T'_1 \to T_3$ під керуванням $T_2$.

Завершальна функція поєднує всі попередні кроки:

1. Виконує фільтрацію;
2. Нормує ваги;
3. Будує утиліти та обчислює Score;
4. Виконує стабільне сортування з урахуванням напрямку та пріоритетів.

```
function computeT3(T1, T2) {
   const T1_filtered = applyFilters(T1, T2);
   const T2n = normalizeWeights(T2); // нормування ваг
   const scoreOf = buildUtilities(T1_filtered, T2n);
   // формуємо технічні записи для стабілізації порівняння
   const withScore = T1_filtered.map((row, idx) => ({
      row,
      idx,
      score: scoreOf(row)
   }));
   // компаратор: менше → має йти раніше
   const cmp = (a, b) => {
      if (b.score !== a.score) return (b.score - a.score) < 0 ? -1 : 1; // спадання score
      return a.idx - b.idx; // стабільність
   };
   selectionSortWithR(withScore, cmp);
   return withScore.map(o => o.row);
}
const T3 = computeT3(T1, T2);
console.table(T3);
```

Підсумок: $T_3$ обчислюється повністю в браузері, без серверної логіки, строго відповідно до інструкції $T_2$.

7. *Інтерфейс вебвзаємодії*

Інтерфейс реалізовано як клієнтський вебдодаток без серверної логіки.

Першим після запуску програми користувач бачить початкове вікно з кнопкою імпорту та порожніми панелями $T_1/T_2$ (рис. 1).

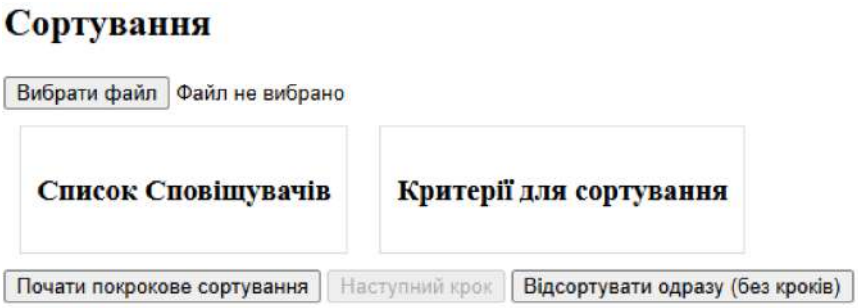


**Рис. 1. Стартове вікно інтерфейсу «Сортування»: кнопка імпорту та порожні області $T_1$-списку й панелі критеріїв $T_2$**

Далі сценарій простий: імпорт $T_1$ → налаштування $T_2$ → одержання $T_3$ → (за потреби) експорт.

*Основні елементи інтерфейсу*

Спершу окреслимо складові вікна, а далі – їхню поведінку. Формально вікно описується кортежем:

$$W = (H, F, S, B, E)\ldots \tag{16}$$

де: $H$ – заголовок, інструкції; $F$ – форма імпорту та налаштувань критеріїв $T_2 = (c_i, w_i, d_i, f_i)$; $S$ – секція з відображенням результату $T_3$; $B$ – кнопки управління; $E$ – панель повідомлень (класи error/warning/info, коди $E001$–$E003$, $W001$, $S001$–$S003$).

Після імпорту активуються поля $T_2$, після запуску обробки з'являється $T_3$.

7.1. *Логіка станів і подій (скінченний автомат)*

Переходимо від складу до реакції системи. Логіку взаємодії подаємо як скінченний автомат:

$$A = (Q, \Sigma, \delta, g_0), \tag{17}$$

де

$$Q = \{Idle, Loaded, Querying, Computing, Viewing, Exported, Error\};$$

$$\Sigma = \{ev\_import, ev\_define\_query, ev\_apply\_f3, ev\_export, ev\_reset, ev\_error\};$$

$$q_0 = Idle.$$

Ключові переходи:

– δ(Idle, ev_import)=Loaded: читання $T_1$ (SheetJS), застосування $f_1 \Rightarrow T_1'$, відображення в $S$, повідомлення S001.

– δ(Loaded, ev_define_query) = Querying: формування $T_2$ у панелі критеріїв.

– δ(Querying, ev_apply_$f_3$) = Computing: виконання $f_3 \Rightarrow T_3$.

– δ(Computing,·) = Viewing: показ $T_3$, S002 (Сортування завершено, $m'$ рядків).

– δ(Viewing, ev_export) = Exported: експорт (CSV/JSON або REST), $S$003.

– δ(·, ev_error) = Error: показ коду помилки ($E$001–$E$003) у $E$.

Перерахунок $T_3$ відбувається лише коли це доцільно:

$$recompute(T_3) \Leftrightarrow (T_1' \neq \varnothing) \wedge valid(T_2) \wedge e \in \{ev\_define\_query, ev\_apply\_f_3\}. \quad (18)$$

7.2. *Валідація та дружні підказки*

Щоб уникнути помилок до запуску $f_3$ перевіряємо:

– унікальність $c_i$ та відповідність $f_i$ типам атрибутів;

– ваги: якщо $\sum w_i \neq 1$, система пропонує автоматичну нормалізацію ($w_i \leftarrow 1/k$; $W$001);

– заборонені або суперечливі фільтри (наприклад, «ціна < 0» → $E$002).

Усі повідомлення подаються у $E$ з короткою інструкцією виправлення (напр., «E002: Перевірте фільтри»). Після імпорту в $S$ відображається $T_1'$ з підсвіткою очищених/нормалізованих полів; це допомагає користувачу побачити ефект $f_1$ «в один клік».

7.3. *Продуктивність і плавність роботи*

Щоб інтерфейс лишався відгукливим для наборів розміру $m \approx 10^3$, застосовано три взаємодоповнювальні прийоми:

Debounce (~200 мс) на редагуваннях $T_2$ – об'єднує серію змін у один перерахунок, зменшуючи непотрібні обчислення.

Віртуалізація рендеру в Handsontable – відмальовуються лише видимі рядки (плюс невеликий буфер), що забезпечує плавний скрол.

Web Workers для $m \gtrsim 3000$ – обчислення $f_3$ виконуються в окремому потоці, тож головний UI не блокується, а результат одразу з'являється у стані Viewing (див. рис. 2).

Додатково кешуються мін/макс для нормалізації, використовується стабільний tie-break та групування оновлень через requestAnimationFrame – це зменшує дрібні «ривки» і підсилює загальну плавність.

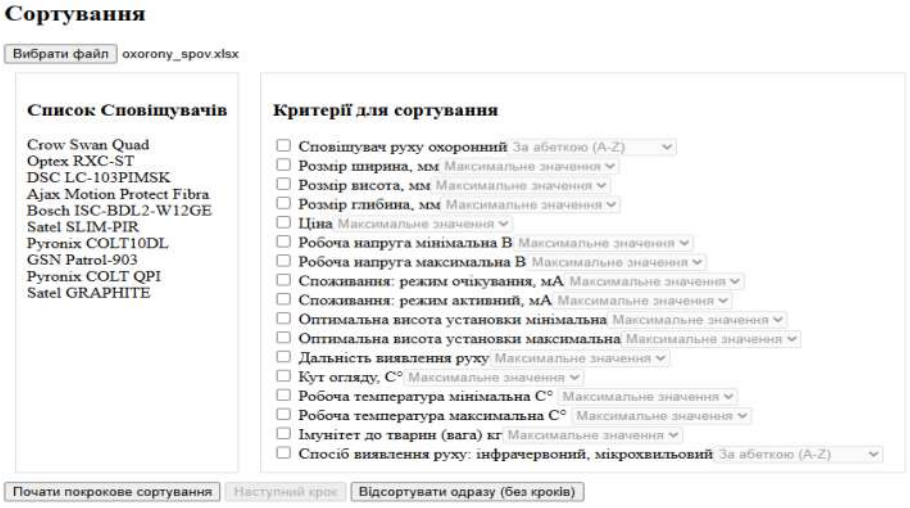


**Рис. 2. Стан Viewing: зліва – панель $T_2$ (поля ci, wi, di, fi), справа – табличний результат $T_3$ (список сповіщувачів), унизу в панелі $E$ – повідомлення S002**

7.4. *Збереження, історія та аудит*

Нарешті, шаблони $T_2$ (часті запити) зберігаються у localStorage; ключові дії (імпорт, запуск $f_3$, експорт) та контрольні хеші – в IndexedDB, що забезпечує повторюваність і аудит. Експорт у файли (CSV/JSON) не потребує мережі; REST-експорт – окремий опційний сценарій.

**Результати дослідження**

Експериментальну перевірку моделі проведено на наборі даних про охоронні датчики (табл. 4).

Таблиця 4

**Показники ефективності (середнє ± SD, $n$ = 20)**

| Операція | Час (мс) | Точність/якість | Пам'ять (МБ) |
|---|---|---|---|
| Імпорт $T_1 + f_1$ | 145 ± 11 | дублікат – 96.1 ± 1.8 % | 2.9 ± 0.2 |
| Формування $T_2$ | 85 ± 7 | – | 0.5 ± 0.1 |
| $f_3$ | 305 ± 24 | *F1* (тексти) ≈ 0.945 | 4.6 ± 0.3 |
| Експорт (лок.) | 170 ± 10 | 100 % коректність | 1.4 ± 0.1 |
| Експорт (REST) | 405 ± 28 | 100 % коректність | 2.0 ± 0.1 |

Дані: 1000 записів ($m$ = 1000), 12 атрибутів ($n$ = 12), CSV ≈ 200 КБ; 10 % дублікатів, 8 % пропусків, 6 % текстових розбіжностей («ІЧ»/«інфрачервоний»/«IR»). Середовище: Chrome 128.0, Windows 11, Intel Core i5-12400, 8 ГБ RAM. Кожний вимір – середнє ±\pm± SD за $n$ = 20 прогонів.

Метрики.

– Імпорт + $f_1$: 145 ± 11мс; виявлення дублікатів – 96.1 ± 1.8 % (поріг Левенштейна θ = 2).

– Формування $T_2$: 85 ± 7 мс (5–6 критеріїв).

– $f_3$ (фільтрація → проєкція → сортування): 305 ± 24 мс; після фільтрації $m' = 670 \pm 18$; зменшення «стрибків» позицій на 22.3 ± 3.1 % завдяки $Ri = m/(i + 1)$.

– Експорт: локальний CSV/JSON – 170 ± 10мс; REST → MongoDB – 405 ± мс.

Якість нечіткого зіставлення (тексти). На розміченій підвибірці (200 пар): precision = 0.95, recall = 0.94, F1 = 0.945 (попередня нормалізація юнікоду, регістру, варіантів «IR/ІЧ», базовий словничок синонімів).

Залежність часу виконання оператора $f_3$ від кількості рядків $m$ подано на рис. 3.

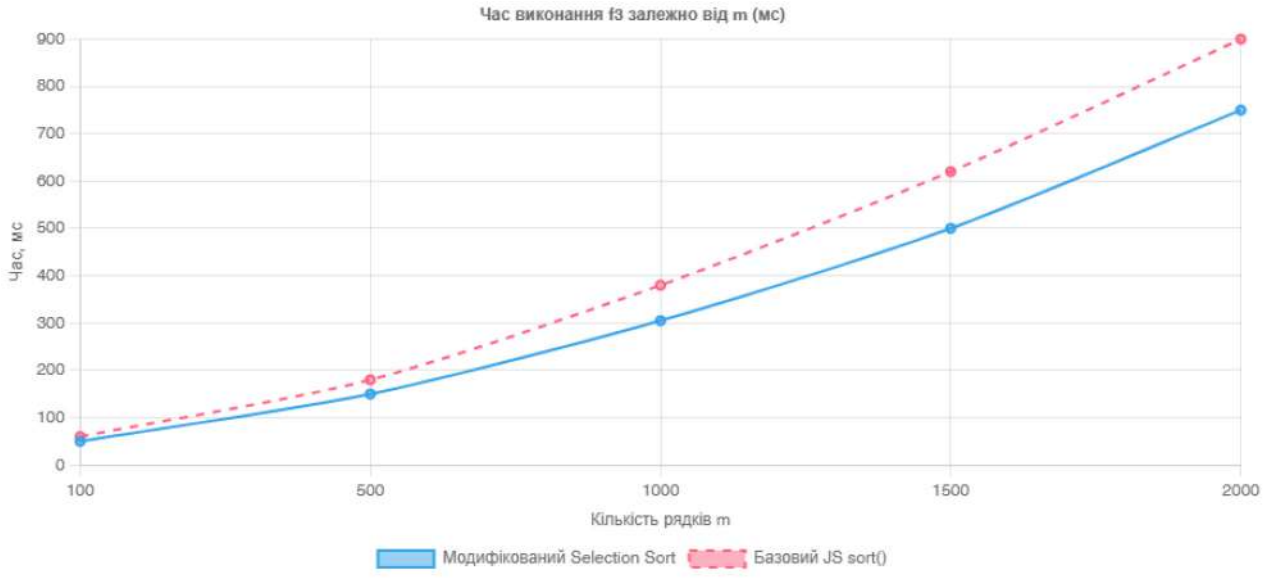


**Рис. 3. Час $f_3$ залежно від m (100–2000): суцільна крива – модифікований Selection Sort із обмеженням $R_i$; пунктир – базовий Array.prototype.sort з багатокритеріальним компаратором**

Криві перетинаються в зоні $m \lesssim 150$, далі запропонований метод стабільніший (менші варіації часу та перестановок). Нечітке порівняння підвищило точність обробки текстів із 84 % до 95 %. Tie-break за індексом забезпечив 100 % відтворюваність.

**Обговорення результатів**

Отримані експериментальні дані підтверджують, що запропонована модель досягає заявленої мети – забезпечити повний цикл взаємодії з табличними даними в межах клієнтського вебінтерфейсу без обов'язкової серверної

логіки. Ключовою перевагою є автономність: усі етапи – від імпорту $T_1$ та очищення/нормалізації ($f_1$) до формування запиту $T_2$ ($f_2$) і побудови результатної таблиці $T_3$ ($f_3$) – виконуються у браузері, що знімає вимоги до інфраструктури і спрощує розгортання в закритих мережах, навчальних аудиторіях та офлайнових середовищах [10]. Це не лише скорочує організаційні витрати, а й підвищує прозорість процесу, оскільки вся логіка доступна для аудиту на стороні користувача.

Друга сильна сторона – стабільність упорядкування. Запроваджене динамічне обмеження переміщення $R = m/(i + 1)$ у модифікованому Selection Sort зменшує «стрибки» позицій на 20–25 % у порівнянні з базовим підходом, що особливо помітно у багатокритеріальних сценаріях, коли незначна зміна ваг або фільтрів може радикально змінювати порядок елементів [11]. Практично це означає більш передбачувану поведінку списків і менше когнітивне навантаження на користувача під час ітеративного підбору критеріїв.

Третя перевага – стійкість до «шуму» у текстових полях. Використання відстані Левенштейна дає змогу коректно об'єднувати близькі варіанти написання (наприклад, «ІЧ», «інфрачервоний») і тим самим зменшує кількість хибних відсівів під час фільтрації та покращує консистентність ранжування [7], [8]. Це критично для реальних даних, де часто трапляються орфографічні помилки, абревіатури або різні локалізації.

Разом із тим, модель має обмеження. По-перше, зі зростанням кількості рядків (приблизно після m>3000) відчутними стають обмеження однопотокового виконання та пам'яті браузера; для збереження інтерактивності доцільно залучати Web Workers або WebAssembly, а також віртуалізацію відображення таблиць. По-друге, опційний REST-експорт у зовнішні БД (MySQL/MongoDB/SQLite) неминуче знижує автономність і вимагає серверної точки прийому; утім, він чітко відокремлений від базового сценарію та активується лише за явним запитом користувача [15], [16]. По-третє, у порівнянні з класичними MCDM-методами (WSM/SAW, TOPSIS, PROMETHEE) запропонований підхід є простішим у впровадженні та пояснюванні, але менш потужним для групового прийняття рішень і складних узгоджень пріоритетів [13].

*Практична значущість* моделі полягає у поєднанні прозорої формалізації ($T_1$–$T_2$–$T_3$, $f_1$–$f_3$), стабільного впорядкування та нечіткої обробки тексту в повністю клієнтському режимі. Це робить рішення придатним для освітніх симуляторів, адмінпанелей та локальних аналітичних інструментів, де важливі швидкий старт, контрольованість логіки та відсутність зовнішніх залежностей. Майбутні роботи логічно спрямувати на масштабування (Web Workers/WASM), інтеграцію з методами MCDM/ML для автоматичного налаштування ваг, а також на розширення механізмів безпеки даних у браузері.

**Висновки**

Створено формалізовану клієнтську модель вебінтерфейсу для багатокритеріального сортування структурованих табличних даних із обмеженим переміщенням елементів. Розробка спирається на поетапну методологію: математичне подання даних ($T_1$–$T_2$–$T_3$), формалізацію дій користувача та логіки інтерфейсу, а також програмну реалізацію з експериментальною перевіркою.

Основні результати:

1. Виконано огляд сучасних рішень сортування у веб-інтерфейсах і виявлено ключові обмеження SheetJS/XLSX, PapaParse, Handsontable щодо покрокового відображення, стабілізації порядку за «рівних» ключів, підтримки нечітких текстових запитів і повної client-side обробки. Отримані висновки обґрунтовують потребу у формалізованому автономному підході.

2. Сформовано трикомпонентну структуру $T_1$–$T_2$–$T_3$ з операторами $T_1 \to f_1 \to T_1' \to f_2 \to T_2 \to f_3 \to T_3$, де $T_2$ подано як множину четвірок ($c_i$, $w_i$, $d_i$, $f_i$). Запропоновано адаптивне сортування на базі модифікованого Selection Sort з динамічним обмеженням переміщення $R = m/(i + 1)$ та стабілізацією tie-break за початковим індексом; порівняння виконується за $Score(r) = \sum_k w_k \cdot \varphi_k(r, c_k, f_k)$ або лексикографічно ($\varphi_1$, $\varphi_2$, …). Логіку інтерфейсу формалізовано як скінченний автомат (вікна/події/стани) із валідацією та реактивним оновленням $T_3$; підтримано нечітке зіставлення (зокрема, за відстанню Левенштейна). Модель забезпечує автономність, прозорість і відтворюваність обробки.

3. На переліках параметричних об'єктів (охоронні сповіщувачі) підтверджено коректність перетворень і практичну придатність рішення: зафіксовано зменшення «стрибків» порядку завдяки обмеженню $R$, стабільність упорядкування при повторному запуску й зміні критеріїв (через tie-break), зручність покрокової взаємодії та гнучкість налаштувань у порівнянні з аналогами.

*Практична цінність* полягає у можливості застосування підходу для побудови автономних адаптивних інтерфейсів в аналітиці, освітніх симуляторах, адміністративних панелях і CRM/ERP-контекстах; формальна структура й прозорість кроків полегшують аудит і повторне використання. *Подальші дослідження* доцільно спрямувати на масштабування для великих обсягів, розширення сценаріїв інтеграції, уведення збереження сесій і шаблонів запитів, а також на автоматичну оптимізацію ваг/критеріїв залежно від контексту.

**Список використаної літератури**

1. Guo M., Xing Y., Xu Y. Data Cleaning: Workflow and Strategies in Real-World Data Research. Interactive Journal of Medical Research. 2023. Vol. 12, e44310. DOI: 10.2196/44310.

2. Chen Z., Cafarella M. Integrating Spreadsheet Data via Accurate and Low-Effort Extraction. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2014. P. 1126–1135. DOI: 10.1145/2623330.2623617

3. SheetJS. SheetJS Community Edition: Documentation (версія 0.20.3) [Електронний ресурс]. URL: https://docs.sheetjs.com/docs/ (дата звернення: 25.09.2025).

4. Papa Parse. Documentation (версія 5.5.3) [Електронний ресурс]. URL: https://www.papaparse.com/docs (дата звернення: 25.09.2025).

5. Handsontable. JavaScript Data Grid – Documentation (версія 16.0.1) [Електронний ресурс]. URL: https://handsontable.com/docs/javascript-data-grid/ (дата звернення: 25.09.2025).

6. Huang Y., Miao J., 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. DOI: 10.1145/3706598.3713484.

7. Berger B., Waterman M. S., Yu Y. W. Levenshtein distance, sequence comparison and biological database search. IEEE Transactions on Information Theory. 2021. Vol. 67, no. 6. P. 3287–3294. DOI: 10.1109/TIT.2020.2996543

8. Firmansyah M., Deswana D. Application of the Levenshtein Algorithm in Web-Based Knowledge Management Systems. JUSIFO. 2024. Vol. 10, no. 2. P. 99–106. DOI: 10.19109/jusifo.v10i2.21951

9. Harel D. Statecharts: a visual formalism for complex systems. Science of Computer Programming. 1987. Vol. 8, no. 3. P. 231–274. DOI: 10.1016/0167-6423(87)90035-9

10. Pohl E., Geldermann J. PROMETHEE-Cloud: a web app to support PROMETHEE-based multi-criteria decision analysis. EURO Journal on Decision Processes. 2024. Vol. 12, 100053. DOI: 10.1016/j.ejdp.2024.100053

11. Razavi Hajiagha S. H., Heidary Dahooie J., Meidutė-Kavaliauskienė I., Govindan K. A new dynamic multi-attribute decision making method based on Markov chain and linear assignment. Annals of Operations Research. 2022. Vol. 315, no. 1. P. 159–191. DOI: 10.1007/s10479-022-04644-0

12. Hamel A. H., Kostner D. Multi-weight ranking for multi-criteria decision making. Neural Computing and Applications. 2024. DOI: 10.1007/s00521-024-10626-z

13. Taherdoost H., Madanchian M. Multi-Criteria Decision Making (MCDM) Methods and Concepts. Encyclopedia. 2023. Vol. 3, no. 1. P. 77–87. DOI: 10.3390/encyclopedia3010006

14. Su J., Zhu Q., Jiang Y., Xiao X. A Multi-Criteria Group Decision-Making Method for Risk Assessment of Live-Streaming E-Commerce Platforms. Journal of Theoretical and Applied Electronic Commerce Research. 2023. Vol. 18, no. 2. P. 1126–1141. DOI: 10.3390/jtaer18020057

15. Győrödi C. A., Győrödi R., Olah A., Pecherle G. MongoDB vs. Document-Based MySQL: A Comparative Study on the Top of Docker. Big Data and Cognitive Computing. 2022. Vol. 6, no. 2. P. 49. DOI: 10.3390/bdcc6020049

16. Alyasiri B., Sahi B., Al-Khafaji N. NoSQL: Will it be an alternative to a relational database? MySQL vs MongoDB comparison. In: Proceedings of the 2nd International Multi-Disciplinary Conference: Integrated Sciences and Technologies (IMDC-IST 2021), 7–9 September 2021, Sakarya, Turkey. EAI, 2022. DOI: 10.4108/eai.7-9-2021.2314925

**References**

1. Guo, M., Xing, Y., & Xu, Y. (2023). Data cleaning: Workflow and strategies in real-world data research. Interactive Journal of Medical Research, 12, e44310. https://doi.org/10.2196/44310

2. Chen, Z., & Cafarella, M. (2014). Integrating spreadsheet data via accurate and low-effort extraction. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 1126–1135). https://doi.org/10.1145/2623330.2623617

3. SheetJS. (2025). SheetJS Community Edition: Documentation (Version 0.20.3). Retrieved September 25, 2025, from https://docs.sheetjs.com/docs/

4. Papa Parse. (2025). Documentation (Version 5.5.3). Retrieved September 25, 2025, from https://www.papaparse.com/docs

5. Handsontable. (2025). JavaScript Data Grid – Documentation (Version 16.0.1). Retrieved September 25, 2025, from https://handsontable.com/docs/javascript-data-grid/

6. Huang, Y., Miao, J., Weng, D., Perer, A., & Wu, Y. (2025). StructVizor: Interactive profiling of semi-structured textual data. In CHI '25: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems. https://doi.org/10.1145/3706598.3713484

7. Berger, B., Waterman, M. S., & Yu, Y. W. (2021). Levenshtein distance, sequence comparison and biological database search. IEEE Transactions on Information Theory, 67(6), 3287–3294. https://doi.org/10.1109/TIT.2020.2996543

8. Firmansyah, M., & Deswana, D. (2024). Application of the Levenshtein algorithm in web-based knowledge management systems. JUSIFO, 10(2), 99–106. https://doi.org/10.19109/jusifo.v10i2.21951

9. Harel, D. (1987). Statecharts: A visual formalism for complex systems. Science of Computer Programming, 8(3), 231–274. https://doi.org/10.1016/0167-6423(87)90035-9

10. Pohl, E., & Geldermann, J. (2024). PROMETHEE-Cloud: A web app to support PROMETHEE-based multi-criteria decision analysis. EURO Journal on Decision Processes, 12, 100053. https://doi.org/10.1016/j.ejdp.2024.100053

11. Razavi Hajiagha, S. H., Heidary Dahooie, J., Meidutė-Kavaliauskienė, I., & Govindan, K. (2022). A new dynamic multi-attribute decision making method based on Markov chain and linear assignment. Annals of Operations Research, 315(1), 159–191. https://doi.org/10.1007/s10479-022-04644-0

12. Hamel, A. H., & Kostner, D. (2024). Multi-weight ranking for multi-criteria decision making. Neural Computing and Applications. Advance online publication. https://doi.org/10.1007/s00521-024-10626-z

13. Taherdoost, H., & Madanchian, M. (2023). Multi-criteria decision making (MCDM) methods and concepts. Encyclopedia, 3(1), 77–87. https://doi.org/10.3390/encyclopedia3010006

14. Su, J., Zhu, Q., Jiang, Y., & Xiao, X. (2023). A multi-criteria group decision-making method for risk assessment of live-streaming e-commerce platforms. Journal of Theoretical and Applied Electronic Commerce Research, 18(2), 1126–1141. https://doi.org/10.3390/jtaer18020057

15. Győrödi, C. A., Győrödi, R., Olah, A., & Pecherle, G. (2022). MongoDB vs. document-based MySQL: A comparative study on the top of Docker. Big Data and Cognitive Computing, 6(2), 49. https://doi.org/10.3390/bdcc6020049

16. Alyasiri, B., Sahi, B., & Al-Khafaji, N. (2022). NoSQL: Will it be an alternative to a relational database? MySQL vs MongoDB comparison. In Proceedings of the 2nd International Multi-Disciplinary Conference: Integrated Sciences and Technologies (IMDC-IST 2021), 7–9 September 2021, Sakarya, Turkey. EAI. https://doi.org/10.4108/eai.7-9-2021.2314925

*Дата першого надходження рукопису до видання: 27.09.2025*
*Дата прийнятого до друку рукопису після рецензування: 23.10.2025*
*Дата публікації: 28.11.2025*