ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп’ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)

«___» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

РОЗРОБКА ВЕБ-ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ СПРАВАМИ

зі спеціальності 122 Комп’ютерні науки
освітня програма «Комп’ютерні науки»
ступеня бакалавра

Виконавець роботи Довгий Іван Васильович
_____ «____»_____ 2025 р.
(підпис)

Науковий керівник к.пед.н., доцент, Кошова Оксана Петрівна
_____ «____»_____ 2025 р.
(підпис)

Рецензент

ПОЛТАВА 2025 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Олена ОЛЬХОВСЬКА
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ І КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Розробка веб-застосунку для управління справами**»**
зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Довгий Іван Васильович
Затверджена наказом ректора № 177-Н від «10» вересня 2024 р.
Термін подання студентом роботи «___»_____ 2025 р.
Вихідні дані до кваліфікаційної роботи: статті та документації з теми розробки веб-застосунків.

Зміст пояснювальної записки (перелік питань, які потрібно розробити)

ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
1.1. Мета та задачі дослідження
1.2. Визначення вимог до функціоналу системи управління справами
РОЗДІЛ 2. ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ
2.1. Огляд існуючих інструментів для управління справами
2.2. Порівняння функціональних можливостей та технічних особливостей аналогічних рішень
2.3. Переваги та обмеження наявних веб-рішень
2.4. Технології та методи, що використовуються для розробки веб-застосунків
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Архітектура та загальна структура веб-застосунку
3.2. Методологія розробки програмного забезпечення та побудова блок-схеми системи
3.3. Проєктування інтерфейсу користувача веб-застосунку
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1. Опис функціональності веб-застосунку
4.2 Реалізація веб-застосунку та побудова його структурної моделі
4.3 Опис та реалізація інтерфейсу користувача і функціональних можливостей веб-застосунку
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А
Перелік графічного матеріалу: 13 ілюстрацій.

Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Кошова О.П.		
Інформаційний огляд	Кошова О.П.		
Теоретична частина	Кошова О.П.		
Практична частина	Кошова О.П.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
Вступ		
Вивчення методичних рекомендацій та стандартів та звіт керівнику		
Постановка задачі		
Інформаційний огляд джерел бібліотек та інтернету		
Теоретична частина		
Практична частина		
Закінчення оформлення		
Доповідь студента на кафедрі		
Доробка (за необхідністю), рецензування		

Дата видачі завдання «__» _____ 2024 р.

Здобувач вищої освіти _____ Довгий Іван Васильович
(підпис)

Науковий керівник _____ к.пед.н., доц. Кошова О.П.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 2025 р.

Секретар ЕК_____ _____
(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 2024 р.

Погоджено
Науковий керівник _____
к.пед.н., Оксана КОШОВА
«____» _____ 2024 р.

План

дипломного проекту з фаху
спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
на тему «Розробка веб-застосунку для управління справами»
Прізвище, ім'я, по батькові Довгий Іван Васильович

ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
1.1. Мета та задачі дослідження
1.2. Визначення вимог до функціоналу системи управління справами
РОЗДІЛ 2. ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ
2.1. Огляд існуючих інструментів для управління справами
2.2. Порівняння функціональних можливостей та технічних особливостей аналогічних рішень
2.3. Переваги та обмеження наявних веб-рішень
2.4. Технології та методи, що використовуються для розробки веб-застосунків
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Архітектура та загальна структура веб-застосунку
3.2. Методологія розробки програмного забезпечення та побудова блок-схеми системи
3.3. Проєктування інтерфейсу користувача веб-застосунку
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1. Опис функціональності веб-застосунку
4.2 Реалізація веб-застосунку та побудова його структурної моделі
4.3 Опис та реалізація інтерфейсу користувача і функціональних можливостей веб-застосунку
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А

Здобувач вищої освіти _____ Довгий І.В.
(підпис)
«____» _____ 2025 р.

РЕФЕРАТ

Записка: 71 сторінки, 13 рисунків, 1 таблиця, 1 додаток, 11 літературних джерел.

Мета дипломної роботи – розробка веб-застосунку, що дозволяє створювати, редагувати, зберігати та контролювати виконання особистих завдань у веб-браузері.

Об'єкт дослідження – процес організації та контролю особистих завдань у цифровому середовищі.

Предмет дослідження – методи, інструменти та архітектурні рішення, що забезпечують розробку веб-застосунку для управління справами, з урахуванням локального збереження даних, адаптивного дизайну та інтерфейсної зручності.

Визначено ключові вимоги до розробки програмного забезпечення веб-застосунку для управління справами; проведено аналіз переваг та обмежень аналогічних онлайн-сервісів; проаналізовано та описано підходи й інструменти, використані при реалізації веб-застосунку; описано алгоритм і методологію створення системи; розроблено користувацький інтерфейс для мобільних і десктопних пристроїв та складено інструкцію користувача для роботи з програмним забезпеченням; представлено блок-схему, use-case діаграму, що ілюструють структуру та логіку програмного коду розробленого застосунку; реалізовано й протестовано веб-застосунок для управління справами у браузері.

ЗМІСТ

ВСТУП

У сучасному цифровому середовищі питання ефективної організації особистого часу та завдань набуває дедалі більшої актуальності. Зростаючий обсяг інформації, висока динаміка повсякденного життя та багатозадачність вимагають від користувачів нових підходів до управління справами. У зв’язку з цим важливого значення набувають інструменти, що дозволяють автоматизувати процес планування, відстеження та виконання завдань у зручному і доступному форматі. Одним із найперспективніших напрямів у цьому контексті є розробка веб-застосунків для управління справами, які поєднують універсальність, простоту використання та мобільність.

Незважаючи на значну кількість вже існуючих рішень у цій сфері, багато з них орієнтовані переважно на корпоративний сегмент або мають надмірно складну структуру, що ускладнює їхнє використання звичайними користувачами. Це створює передумови для пошуку та реалізації нових підходів до створення веб-інструментів, які б відповідали сучасним вимогам щодо функціональності, адаптивності, зручності інтерфейсу та безпеки зберігання даних. Саме тому тема створення індивідуального веб-застосунку для управління справами є актуальною та перспективною як у теоретичному, так і в практичному аспектах.

Метою даної кваліфікаційної роботи є розробка веб-застосунку, що дозволяє створювати, редагувати, зберігати та контролювати виконання особистих завдань у веб-браузері. Такий застосунок має забезпечити зручний, інтуїтивно зрозумілий і адаптивний інтерфейс, підтримку автономної роботи завдяки використанню локального сховища браузера та базові інструменти для персонального планування.

Об’єктом дослідження виступає процес організації та контролю особистих завдань у цифровому середовищі.

Предметом дослідження є методи, інструменти та архітектурні рішення, що забезпечують розробку веб-застосунку для управління справами, з урахуванням локального збереження даних, адаптивного дизайну та інтерфейсної зручності.

Методологічну основу дослідження склали загальнонаукові методи аналізу та синтезу, моделювання, методи веб-програмування та проектування інтерфейсів користувача. У процесі роботи було використано сучасні технології веб-розробки: HTML, CSS, JavaScript, а також локальні механізми зберігання даних для забезпечення автономності та зручності користування.

Практичне значення отриманих результатів полягає у створенні працездатного прототипу веб-застосунку для управління справами, який може бути адаптований для використання як індивідуальними користувачами, так і малими організаціями з метою підвищення ефективності роботи. Розроблений застосунок може стати основою для подальшого розширення функціоналу, інтеграції з хмарними сервісами або мобільними платформами.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. У вступі розкрито актуальність теми, визначено мету, об’єкт і предмет дослідження, методи, що були використані, а також практичну значущість розробки. У першому розділі сформульовано постановку задачі. У другому – здійснено огляд сучасного стану проблеми та аналіз аналогів. У третьому розглянуто процес проектування та реалізації програмного забезпечення. У четвертому наведено результати тестування і приклади інтерфейсів. Завершується робота узагальнюючими висновками.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

1.1. Мета та задачі дослідження

У сучасному інформаційному середовищі, яке характеризується високим рівнем цифровізації всіх сфер життя, зростає потреба в інструментах, що дозволяють людині ефективно управляти своїм часом, контролювати виконання особистих завдань і зберігати баланс між професійною діяльністю та побутовими справами. Технологічні можливості сучасного веб-простору відкривають нові підходи до вирішення цих завдань, зокрема через розробку веб-застосунків, що надають зручні засоби організації справ у браузері без потреби в складному налаштуванні або встановленні додаткового програмного забезпечення.

Наявні на ринку рішення у сфері управління завданнями орієнтовані переважно на корпоративний сегмент або команди розробників, що призводить до складності у використанні таких застосунків звичайними користувачами. Вони можуть мати надмірно складний інтерфейс, бути перевантаженими функціоналом, що не завжди є необхідним для індивідуального користувача. Крім того, деякі сервіси вимагають обов’язкової реєстрації або підключення до інтернету для доступу до базових функцій, що обмежує їхню гнучкість і зручність у повсякденному використанні.

На цьому тлі зростає потреба у створенні універсального, легкого у використанні, інтуїтивно зрозумілого та адаптивного веб-застосунку для управління особистими справами. Такий застосунок має забезпечувати мінімальний, але достатній функціонал, який би дозволяв користувачеві вести список справ, відмічати їх виконання, редагувати вміст, а також зберігати дані локально, не залежачи від сторонніх серверів. Окрім функціональності, важливою вимогою є наявність сучасного, простого й адаптивного інтерфейсу, зручного для роботи як на комп’ютері, так і на мобільних пристроях.

Враховуючи зазначене, мета даної кваліфікаційної роботи полягає у розробці веб-застосунку для управління особистими справами, який дозволяє

створювати, редагувати, зберігати та контролювати завдання в інтерактивному інтерфейсі з підтримкою локального збереження даних і адаптивного дизайну. Реалізація такої системи має підвищити особисту ефективність користувача та забезпечити зручний інструмент для самоорганізації.

Досягнення цієї мети передбачає виконання низки взаємопов’язаних дослідницьких і прикладних задач, спрямованих на всебічне вивчення предметної області, визначення технічних рішень та створення завершеного прототипу програмного забезпечення. Насамперед необхідно здійснити огляд і аналіз існуючих рішень у цій сфері, виявити їхні сильні сторони, недоліки, а також оцінити можливості для вдосконалення або створення альтернативи.

Подальшим завданням є формування вимог до функціоналу майбутнього застосунку, що включає опис основних сценаріїв взаємодії користувача з системою, визначення ключових функцій, засобів управління, а також способів збереження та візуалізації даних. На основі цих вимог необхідно розробити архітектуру веб-застосунку з урахуванням можливості його подальшого масштабування або розширення.

Окрему увагу слід приділити проєктуванню інтерфейсу користувача, який повинен бути не лише естетично привабливим, але й функціональним, зручним та доступним для користувачів з різним рівнем цифрової компетентності. У ході реалізації буде використано сучасні засоби HTML, CSS та JavaScript, що дозволить забезпечити інтерактивність, швидкість роботи та гнучкість структури застосунку.

Завершальним етапом є тестування розробленої системи на різних пристроях та в різних браузерах, оцінка її зручності, стабільності, відповідності функціональним вимогам, а також можливостей її подальшого вдосконалення відповідно до зворотного зв’язку від користувачів. Тестування дозволить виявити потенційні помилки або обмеження, а також перевірити, наскільки ефективно застосунок виконує покладені на нього функції.

Таким чином, у межах даного дослідження передбачається здійснення повного циклу проєктування веб-застосунку — від постановки проблеми та аналізу предметної області до розробки прототипу програмного продукту з подальшим тестуванням і оцінкою його ефективності. Усі поставлені задачі спрямовані на створення простого, зручного та надійного інструменту для персонального управління завданнями, що може використовуватись широким колом користувачів у повсякденній практиці.

1.2. Визначення вимог до функціоналу системи

Визначення вимог до функціоналу є одним із ключових етапів у процесі створення будь-якого програмного забезпечення. Саме на цьому етапі формується уявлення про те, яким має бути майбутній продукт, яким чином він буде взаємодіяти з користувачем, які дії зможе виконувати, як оброблятиме дані та в яких умовах забезпечуватиме стабільну роботу. Для веб-застосунку, що призначений для управління особистими справами, особливу увагу слід приділити саме зручності, гнучкості та простоті використання, оскільки цільовою аудиторією є не технічні фахівці, а звичайні користувачі, які прагнуть організувати свою повсякденну діяльність.

Функціональні вимоги визначають основний набір можливостей, якими повинен володіти застосунок. Насамперед передбачено реалізацію можливості створення нових завдань із базовими параметрами, такими як назва, опис і дата або термін виконання. Користувач повинен мати змогу вільно змінювати вміст завдань, а також видаляти непотрібні або застарілі записи. Крім цього, важливо реалізувати зміну статусу завдання — наприклад, відмітку про його завершення, що дозволяє контролювати хід виконання справ і забезпечує зворотний зв’язок.

Особливе значення має підтримка фільтрації завдань — користувач повинен мати змогу переглядати лише активні або завершені справи, або повний

список, що дозволяє сконцентруватися на найважливішому. Таке сортування не тільки підвищує зручність користування, а й створює відчуття порядку та структурованості в особистій організації.

Однією з ключових особливостей розроблюваного застосунку є відсутність залежності від серверної частини. Для реалізації автономної роботи буде використано локальне сховище браузера (localStorage), що забезпечить збереження даних навіть у разі відсутності інтернет-з’єднання. Таким чином, користувач зможе повертатися до своїх завдань у будь-який час без необхідності повторного введення інформації або проходження автентифікації. Це підвищує як надійність, так і простоту використання.

Не менш важливою є реалізація зручного й адаптивного інтерфейсу. Очікується, що користувачі взаємодіятимуть із застосунком як з комп’ютерів, так і з мобільних пристроїв. У зв’язку з цим інтерфейс повинен автоматично підлаштовуватися під різні розміри екранів. Особлива увага приділяється логічній організації вмісту: розділення завдань на блоки, чітке маркування кнопок, збереження інтуїтивної послідовності дій користувача. Колірна схема та шрифти мають забезпечувати комфорт при тривалому користуванні, не викликаючи втоми або плутанини.

Окрім функціональних, система повинна відповідати і низці нефункціональних вимог. Зокрема, йдеться про продуктивність, безперебійну роботу та відповідність сучасним стандартам безпеки й ергономіки. Застосунок має працювати стабільно в сучасних версіях веб-браузерів (Google Chrome, Mozilla Firefox, Microsoft Edge тощо), не допускати зависань або втрати даних у процесі виконання стандартних операцій.

Продуктивність застосунку повинна бути достатньою для забезпечення швидкого реагування на дії користувача. Інтерфейс має оновлюватися динамічно, без потреби перезавантаження сторінки. Завдяки використанню JavaScript та маніпуляціям з DOM-елементами забезпечується інтерактивність і плавна взаємодія між користувачем і програмою.

Ще одним важливим нефункціональним аспектом є масштабованість. У майбутньому доцільно розширити застосунок за рахунок додавання додаткових функцій, таких як нагадування, категоризація завдань, інтеграція з календарями або хмарними сервісами. Для цього вже на етапі проектування важливо передбачити модульну архітектуру та структуру коду, що полегшить подальший розвиток системи.

Система має відповідати також вимогам доступності. Це означає, що інтерфейс повинен бути зручним і зрозумілим навіть для користувачів без технічної підготовки. Використання простих термінів, візуальних підказок, однакових принципів навігації в усіх частинах інтерфейсу сприяє легкому засвоєнню логіки роботи застосунку.

Оскільки система не передбачає обробки конфіденційних або персональних даних, вимоги до безпеки не включають складні алгоритми шифрування чи автентифікації. Проте важливо забезпечити стабільність роботи застосунку та цілісність даних навіть у разі нештатних ситуацій, таких як несподіване закриття вкладки чи перевантаження браузера.

Підсумовуючи, зазначимо, що сформульовані вимоги до функціоналу веб-застосунку охоплюють усі ключові аспекти, необхідні для його ефективної та комфортної експлуатації. Вони враховують потреби звичайного користувача, забезпечують простоту у використанні, підтримку автономного режиму, мобільну адаптивність, гнучкість структури та можливість подальшого вдосконалення. Ці вимоги стали основою для наступного етапу — проектування архітектури застосунку та вибору технологій його реалізації.

РОЗДІЛ 2. ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ

2.1. Огляд існуючих інструментів для управління справами

В умовах стрімкого розвитку цифрових технологій та зростаючої кількості інформаційних потоків, що щодня обробляються сучасною людиною, питання ефективної організації особистих справ набуває особливої актуальності. Користувачі все частіше звертаються до веб-застосунків як до інструментів планування та контролю виконання завдань[1], оскільки саме такі платформи забезпечують швидкий доступ, зручність, а також можливість використання з будь-якого пристрою, незалежно від операційної системи. Програмні продукти для управління завданнями сьогодні відіграють ключову роль у підвищенні особистої ефективності, самоорганізації та дисципліни, як у професійному, так і в особистому житті.

Ринок програмних засобів, призначених для управління справами, насичений різноманітними рішеннями. Кожен сервіс намагається запропонувати користувачам унікальний підхід до вирішення подібної задачі — організувати час, структурувати поточні завдання, створити індивідуальний робочий простір. Популярні платформи пропонують широкий функціонал: можливість встановлення пріоритетів, планування дедлайнів, створення повторюваних завдань, додавання підзавдань, категорій, тегів[2], нагадувань та інших опцій. Вони дозволяють працювати в хмарному середовищі, синхронізуючи дані між пристроями, і часто інтегруються з іншими цифровими сервісами — календарями, поштовими клієнтами, месенджерами. Водночас така функціональна насиченість не завжди відповідає потребам звичайного користувача, який шукає простий та інтуїтивно зрозумілий інструмент для щоденного використання.

Переважна частина популярних платформ орієнтована на досвідчених користувачів або корпоративний сегмент. Вони потребують обов’язкової

реєстрації, налаштування профілю, вибору тарифного плану або підключення до сторонніх акаунтів. Це створює додатковий бар’єр для тих, хто прагне швидкого, автономного та мінімалістичного рішення. Крім того, чимало застосунків функціонують виключно в онлайн-середовищі, що унеможливлює доступ до списку справ у разі відсутності з’єднання з інтернетом. Також не всі сервіси забезпечують адаптивність інтерфейсу, що знижує комфорт використання на мобільних пристроях.

Нерідко інтерфейси таких платформ перевантажені — користувачу пропонують десятки функцій, вкладок, налаштувань, аналітичних звітів, доступів для інших осіб. З одного боку, це може бути корисним у великих командах або для керування проєктами, але в особистому користуванні більшість цих функцій залишається невикористаною. У результаті — сповільнюється навігація, зменшується зосередженість, а головне завдання — швидко зафіксувати або перевірити справу — вимагає більше дій, ніж необхідно. Деякі сервіси також накладають обмеження у безкоштовних версіях: обмеження кількості завдань, відсутність локального збереження, реклама або відсутність технічної підтримки.

Незважаючи на наявність розвиненої інфраструктури застосунків, проблема залишається актуальною. Універсального рішення, яке б поєднувало простоту, автономність, адаптивність та інтуїтивний інтерфейс, поки що не існує. Саме це відкриває нішу для розробки індивідуального веб-застосунку, в якому основний акцент буде зроблено на базовій функціональності — створенні, редагуванні, перегляді й контролі виконання особистих завдань — без надмірностей, але із забезпеченням надійності та зручності.

З метою більш чіткого розуміння існуючої ситуації було здійснено аналіз найбільш популярних програмних продуктів, які користуються попитом у широких колах. Вибір було зумовлено як їхньою популярністю, так і тим, що вони демонструють різні концепції організації справ. Аналіз дозволяє порівняти підходи, оцінити зручність, визначити функціональні сильні та слабкі сторони, а

також зрозуміти, які саме можливості слід реалізовувати або, навпаки, уникати в розроблюваному веб-застосунку.

Першим сервісом, який заслуговує на увагу, є Trello[3]. Це популярна онлайн-платформа, що пропонує канбан-підхід до управління завданнями. Суть полягає у використанні віртуальних дошок, що поділяються на списки, де розміщуються окремі картки — конкретні завдання або задачі. Кожна картка може містити опис, чек-листи, терміни виконання, вкладення, мітки та коментарі. Інтерфейс побудований на принципах візуального планування, що дозволяє швидко оцінити стан справ і контролювати виконання.

Trello забезпечує високу гнучкість і підтримку командної роботи, але саме це часто є недоліком для індивідуального користувача. У застосунку багато функцій, які не потрібні при повсякденному користуванні. Для простої фіксації справ доводиться проходити декілька кроків, вибирати поля, натискати додаткові кнопки. Крім того, платформа не функціонує без підключення до інтернету та вимагає створення облікового запису, що знижує швидкість першого запуску. Водночас складна структура даних і численні можливості організації вмісту створюють додаткове навантаження для користувача, який шукає простоти.

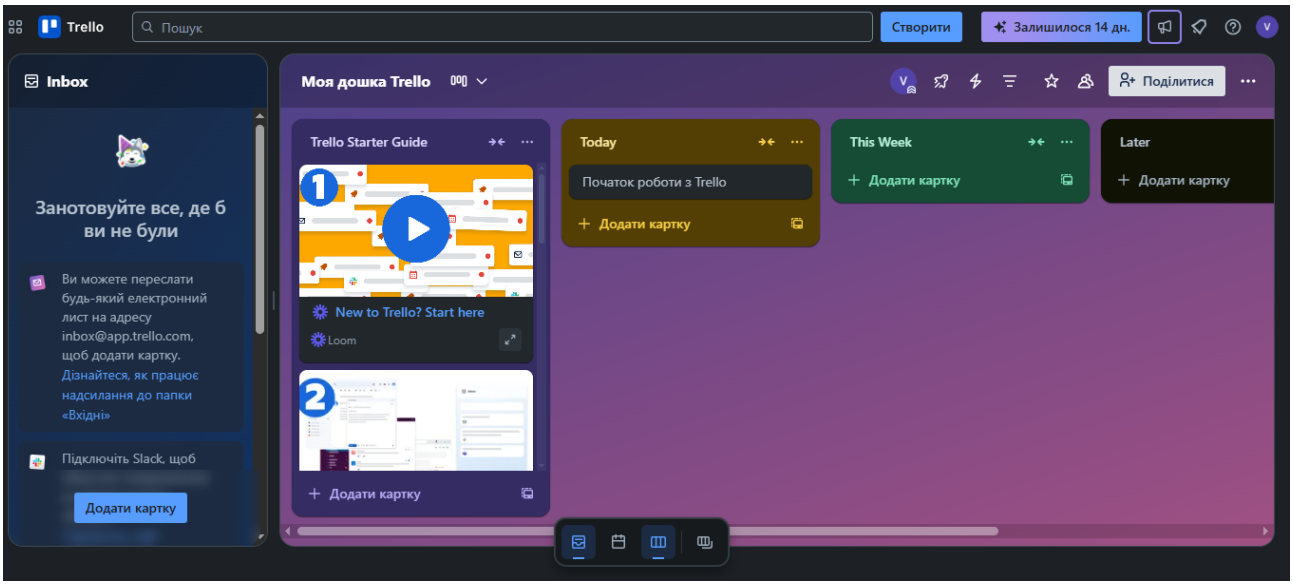

Рисунок 2.1. Візуальне представлення завдань у сервісі Trello

Одним із найбільш поширених веб-застосунків, який орієнтований на індивідуальне управління справами, є Todoist[4]. Цей сервіс має репутацію надійного, зручного та інтуїтивно зрозумілого інструменту для планування завдань. Основна логіка його побудови базується на створенні списків справ, які можуть бути згруповані за проєктами або категоріями. Кожне завдання можна доповнити описом, встановити термін виконання, визначити рівень пріоритету, а також додати мітки та підзадачі.

Інтерфейс застосунку реалізовано в мінімалістичному стилі. Візуальна частина не перевантажена елементами, що дає змогу сконцентрувати увагу саме на змісті завдань. У головному вікні користувач бачить актуальні справи на поточний день або тиждень, а також майбутні завдання, згруповані за датами. Така організація забезпечує чітке уявлення про навантаження, розподіл часу та послідовність виконання. Важливим є також те, що завдання можуть переміщуватись між проєктами або змінювати статус без потреби відкривати додаткові вікна.

З технічного боку Todoist підтримує синхронізацію між усіма пристроями, включно з мобільними додатками, настільними версіями та браузерним інтерфейсом. Особливо цінною є інтеграція з популярними календарями, зокрема Google Calendar, що дає можливість переглядати справи у форматі хронологічного розкладу. Також підтримується інтеграція з такими сервісами, як Gmail, Amazon Alexa, Slack, Outlook тощо. Це розширює межі використання Todoist не лише в особистому, а й у робочому середовищі, де зручно поєднувати електронну пошту, календар і менеджер завдань в єдиному робочому просторі.

Попри очевидні переваги, Todoist має і певні недоліки. Застосунок вимагає обов’язкової реєстрації, і лише після створення облікового запису відкривається доступ до персонального простору. Це унеможливлює використання сервісу як повністю автономного рішення. Крім того, робота з даними реалізована через хмарне сховище, отже, для доступу до завдань потрібне стабільне інтернет-з’єднання. У разі втрати мережі або непередбачених технічних збоїв користувач

може тимчасово залишитися без можливості керувати справами.

Варто також зауважити, що низка важливих функцій, таких як автоматичні нагадування, фільтри, перегляд активності, аналітика продуктивності та налаштування шаблонів, доступна лише в платній версії Todoist. Це створює певні обмеження для користувачів, які не готові переходити на підписку, а задовольняються лише базовим набором можливостей. Безкоштовна версія містить обмежену кількість проєктів, підзадач і спільних доступів.

Незважаючи на ці особливості, Todoist залишається одним із найпопулярніших сервісів для особистого тайм-менеджменту. Завдяки поєднанню зрозумілого інтерфейсу, можливості структуризації завдань та широких функціональних інтеграцій він добре підходить як для новачків, так і для досвідчених користувачів. Його філософія базується на принципі «все на своєму місці», де кожна дія займає мінімум часу, але забезпечує максимум контролю над справами.

Todoist може слугувати корисним зразком для розробки власного веб-застосунку, особливо в частині організації списків, встановлення дедлайнів, фільтрації за статусом і календарної прив’язки. Водночас у реалізації авторського продукту доцільно уникати деяких обмежень, притаманних Todoist, таких як обов’язкова реєстрація, залежність від інтернет-з’єднання та прив’язка до хмарних платформ. Оптимальним рішенням є створення локального, автономного застосунку з базовим функціоналом, адаптивним інтерфейсом і швидкою реакцією на дії користувача.

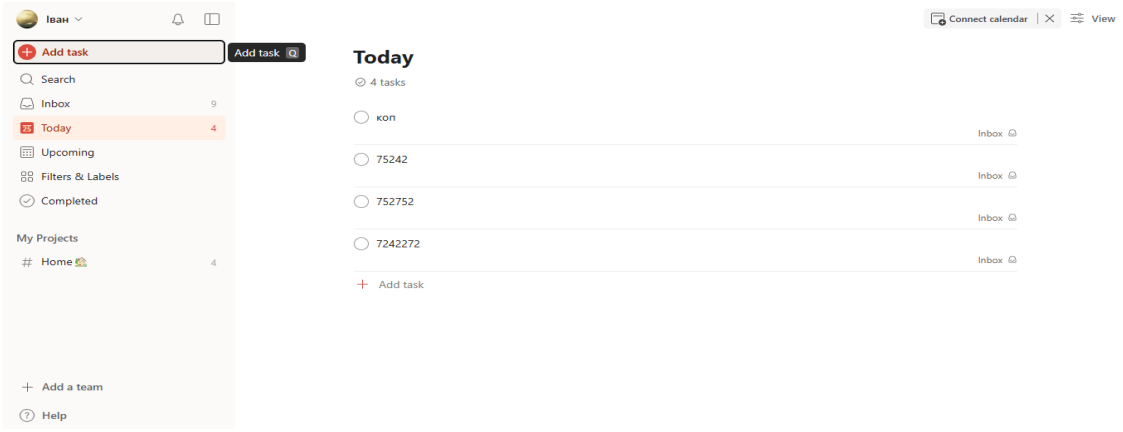

Рисунок 2.2. Інтерфейс застосунку Todoist зі списком запланованих завдань

Серед доступних інструментів для ведення особистих справ Google Tasks[5] вирізняється простотою використання та тісною інтеграцією з іншими продуктами компанії Google. Цей сервіс дозволяє створювати завдання безпосередньо в середовищі Gmail або Google Calendar, перетворювати листи на нагадування і синхронізувати справи між пристроями. Такий підхід забезпечує швидку взаємодію та зменшує час, необхідний для організації щоденних завдань.

Інтерфейс Google Tasks реалізовано у вигляді базового списку, де кожне завдання має назву, опціональний опис, дату виконання та можливість створення підзавдань. Завдання автоматично зберігаються в акаунті користувача Google і доступні з будь-якого пристрою без потреби додаткової реєстрації. Це робить сервіс зручним для щоденного користування, особливо для тих, хто вже активно використовує екосистему Google.

Разом із тим функціональність Google Tasks є досить обмеженою. Сервіс не підтримує сортування за категоріями, не дозволяє додавати теги чи визначати рівні пріоритетності. Відсутність візуальних маркерів і налаштування інтерфейсу знижують можливості персоналізації. Також сервіс не передбачає офлайн-доступу або використання без входу до облікового запису, що обмежує автономність застосування.

Незважаючи на ці обмеження, Google Tasks залишається зручним рішенням для користувачів, яким потрібен базовий інструмент без складної структури. Простота, мінімалізм і інтеграція з поштою та календарем дозволяють легко впровадити його в повсякденну діяльність. У межах розробки власного веб-застосунку цей сервіс можна розглядати як приклад вдалого поєднання швидкості доступу та функціонального мінімуму, проте з потенціалом для розширення функціональності без ускладнення інтерфейсу.

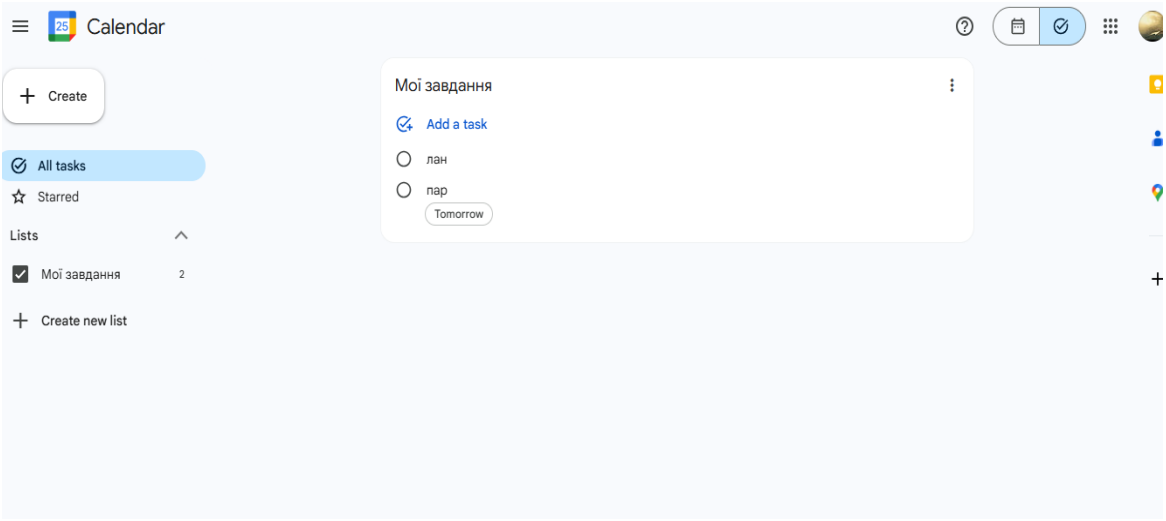

Рисунок 2.3. Вікно Google Tasks

Microsoft To Do є одним із інструментів[6] для організації особистих справ, розроблених у межах цифрової екосистеми Microsoft. Цей застосунок орієнтований переважно на користувачів, які активно використовують продукти корпорації, зокрема Outlook, OneDrive, Microsoft Teams, і потребують зручної синхронізації завдань між цими сервісами. Концептуально Microsoft To Do базується на створенні списків завдань, кожен із яких може містити окремі справи з термінами виконання, нагадуваннями, повтореннями, нотатками та можливістю відмічення виконаних пунктів.

Інтерфейс застосунку реалізовано в структурі вкладок, де кожен список представляє окремий напрям роботи або тематику. Зручність навігації забезпечується зрозумілим дизайном, а структура дозволяє створювати ієрархію завдань без перевантаження візуального середовища. Для користувачів Windows та Office 365 платформа є особливо зручною, адже дозволяє інтегрувати особисті справи з робочими процесами, об’єднуючи пошту, календар і списки в єдину логічну систему.

Перевагою Microsoft To Do є автоматична синхронізація між пристроями, що дозволяє працювати з завданнями з комп’ютера, планшета або смартфона. Завдяки хмарному сховищу всі зміни зберігаються миттєво, а доступ до сервісу здійснюється через єдиний обліковий запис Microsoft. Це спрощує авторизацію,

але водночас робить застосунок малодоступним для користувачів, які не мають або не бажають створювати такий акаунт. Також система не дозволяє працювати в автономному режимі — підключення до інтернету є обов’язковим для повноцінного функціонування.

Ще однією особливістю сервісу є обмежена варіативність налаштувань. Користувачі мають змогу перейменовувати списки, обирати колір теми та сортувати завдання, однак можливості детальної кастомізації, додавання тегів або фільтрації за категоріями є вкрай обмеженими. Через це Microsoft To Do менш привабливий для тих, хто потребує гнучкої організації робочого простору або багаторівневої структури завдань.

У цілому, Microsoft To Do є вдалим прикладом простого і функціонального засобу для управління особистими справами в межах екосистеми Microsoft. Водночас його вузька інтеграційна орієнтованість, відсутність автономності та обмежені можливості персоналізації створюють підстави для пошуку альтернатив або розробки власного інструменту з ширшими можливостями.

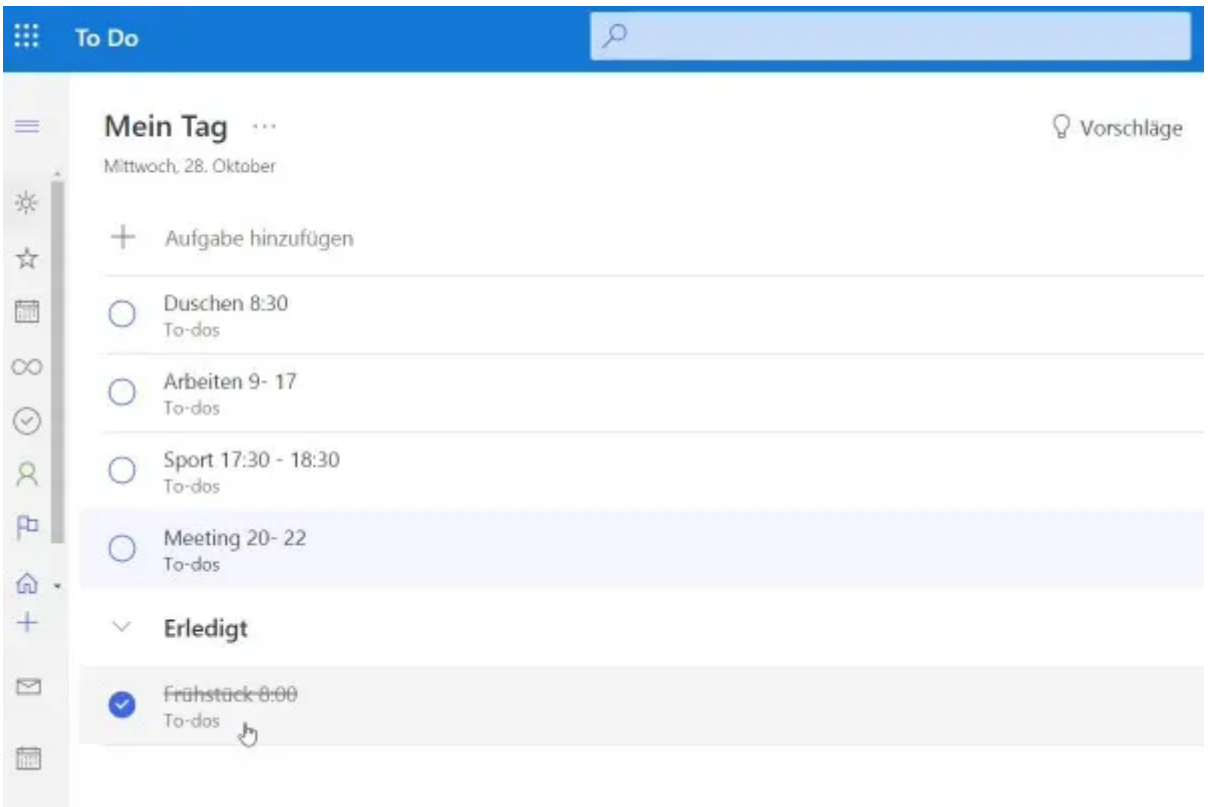

Рисунок 2.4. Списки справ у застосунку Microsoft To Do

Notion — це універсальний цифровий простір, який поєднує функції нотатника, бази даних і планувальника завдань. Користувач може створювати власні сторінки, таблиці, дошки, списки справ та комбінувати їх у зручну структуру. Завдяки такому підходу Notion дозволяє максимально персоналізувати робоче середовище.

Інтерфейс застосунку є гнучким і одночасно складним — користувачеві доводиться самостійно створювати структури, налаштовувати поля й макети. Для тих, хто лише починає працювати з платформою, вивчення основ може потребувати часу. Попри це, сервіс надає широкі можливості для організації завдань у форматі баз даних, календарів або візуальних канбан-дошок.

Notion активно використовується фрілансерами, студентами та малими командами. Він доступний у безкоштовній версії, проте для спільної роботи з розширеними правами доступу передбачено підписку. Його головною перевагою є універсальність, а недоліком — складність для базового щоденного використання.

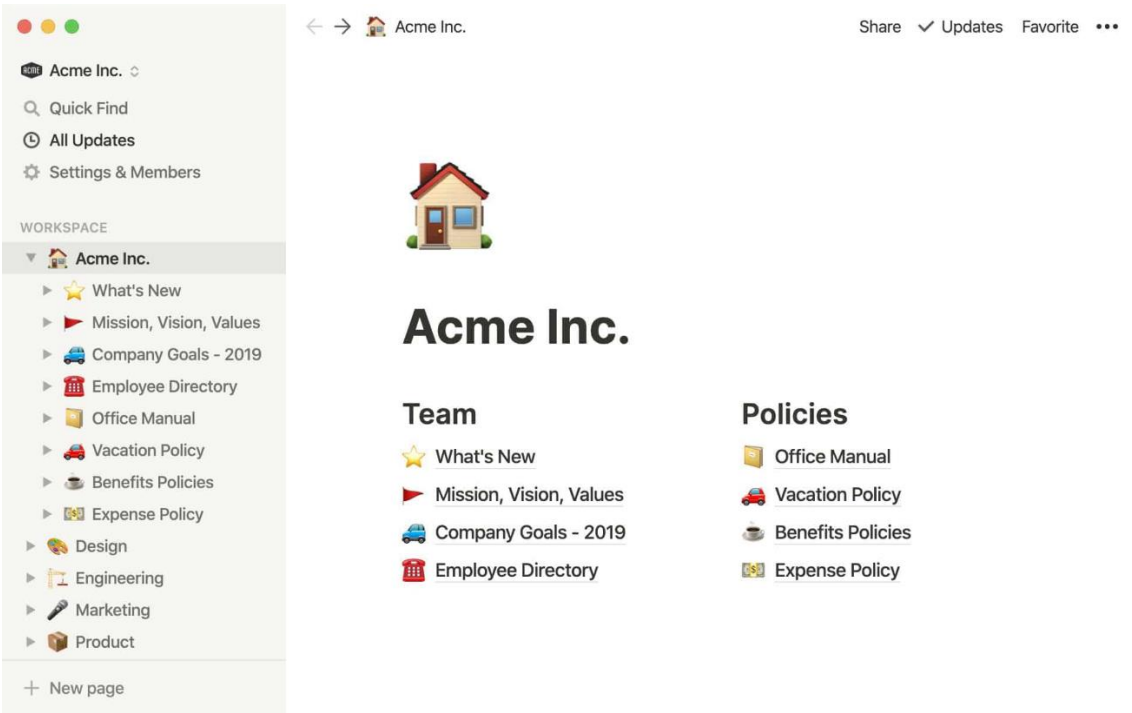

Рисунок 2.5. Організація завдань у середовищі Notion

Проведений аналіз сучасних програмних засобів для управління особистими справами дає змогу сформувати загальну картину функціональних і концептуальних підходів, що застосовуються у цій сфері. Кожен із розглянутих сервісів — Trello, Todoist, Google Tasks, Microsoft To Do та Notion — представляє власне бачення структури та логіки роботи зі списками завдань. Незважаючи на спільне призначення, між ними існують суттєві відмінності як у глибинах реалізації, так і в спрямованості на ту чи іншу категорію користувачів.

Trello і Notion демонструють орієнтацію на гнучке налаштування робочого простору, що вимагає від користувача більше часу на адаптацію, проте відкриває ширші можливості в організації інформації. У той час як Google Tasks і Microsoft To Do, навпаки, зосереджуються на простоті та мінімалізмі, забезпечуючи базову функціональність без надлишкових опцій. Todoist можна вважати певним компромісом — цей сервіс поєднує зручність використання з розширеним функціоналом, хоча частина можливостей доступна лише в комерційній версії.

Усі згадані системи мають спільні характеристики: переважно працюють через веб-браузер або мобільний застосунок, потребують наявності облікового запису, зберігають дані в хмарі та вимагають активного інтернет-з’єднання для коректного функціонування. У більшості випадків відсутня можливість повноцінної офлайн-роботи, що обмежує використання сервісів у середовищах із нестабільним доступом до мережі. Крім того, не всі системи забезпечують адаптивність інтерфейсу на різних типах пристроїв, що ускладнює роботу на мобільних телефонах або планшетах.

Важливим є і той факт, що жоден із аналізованих сервісів не дозволяє працювати безпосередньо з локальними даними без використання акаунта. Це створює додатковий бар’єр для користувача, який прагне автономності та незалежності від хмарного середовища. Крім того, деякі застосунки містять зайвий для особистого користування функціонал — зокрема, інтеграції з корпоративними системами, командна робота, великі аналітичні панелі, що перевантажують інтерфейс і потребують додаткового часу на освоєння.

Підсумовуючи, можна зробити висновок, що, попри широкий вибір на ринку цифрових інструментів для організації справ, існує певний дефіцит простих, автономних, адаптивних рішень із базовим, але достатнім функціоналом. Переважна більшість доступних сервісів або надмірно спрощені, або, навпаки, орієнтовані на складну багаторівневу організацію з глибокими можливостями налаштування, що не завжди відповідає запитам середнього користувача.

У зв’язку з цим актуальним є завдання створення власного веб-застосунку, який би поєднував ключові переваги аналізованих сервісів — простоту, доступність, інтуїтивність інтерфейсу, швидкість взаємодії — та при цьому був би незалежним від сторонніх акаунтів і мережевих обмежень. У такому рішенні особливу увагу доцільно приділити локальному збереженню даних, адаптивній верстці, підтримці основних сценаріїв користування та можливості подальшого розширення функціоналу. Це дозволить задовольнити потреби користувача, який шукає не складну корпоративну систему, а особистий зручний інструмент для щоденного управління завданнями.

2.2. Порівняння функціональних можливостей та технічних особливостей аналогічних рішень

Під час дослідження сучасних веб-застосунків, що реалізують функціонал управління завданнями, було виявлено низку спільних і відмінних рис, які дозволяють оцінити їх ефективність, зручність та доцільність використання в різних сценаріях. Аналіз проводився з урахуванням технічної реалізації, функціональних можливостей, зручності інтерфейсу, рівня автономності та доступності сервісів[7]. Кожен з обраних інструментів має свої переваги, але й водночас демонструє певні обмеження, що важливо враховувати при створенні власного веб-застосунку.

Застосунок	Працює без Інтернету	Потребує реєстрації	Складність інтерфейсу	Збереження даних	Підходить для особистого використання
Trello	частково	так	середня	Хмарне	з обмеженнями
Todoist	частково	так	зручний	Хмарне	так
Notion	ні	так	складний	Хмарне	для просунутих
MS To Do	частково	так	зручний	Хмарне (OneDrive)	так
Google Tasks	ні	так	простий	Хмарне (Google)	так
Розроблений нами	повністю	ні	інтуїтивний	localStorage	ідеально підходить

Зокрема, більшість сервісів підтримують базові дії з управління завданнями: створення, редагування, видалення та зміну статусу. Проте не всі дозволяють зберігати дані локально[2] — більшість використовує хмарне зберігання, що вимагає постійного підключення до інтернету. Також спостерігається залежність від авторизації: практично всі розглянуті сервіси передбачають створення облікового запису[6] або вхід через акаунт іншої платформи. Це створює бар’єр для користувачів, які прагнуть швидкого доступу до функціоналу без реєстрації та обміну персональними даними.

У плані інтерфейсних рішень сервіси поділяються на два основних підходи. Одні — як-от Trello або Notion — орієнтовані на візуальне подання інформації[6] у вигляді блоків, дошок або таблиць. Інші — такі як Todoist, Google Tasks і Microsoft To Do — застосовують класичний формат списку справ, що підходить для швидкої навігації та зменшення візуального навантаження. Водночас рівень адаптивності інтерфейсу також різниться: одні сервіси підтримують повноцінну мобільну версію з адаптованим дизайном[1], інші мають обмежену підтримку мобільних пристроїв або незручне масштабування.

Функціональна гнучкість рішень визначається також наявністю таких можливостей, як підзадачі, нагадування, позначення пріоритетності, тегування, категоризація, пошук, аналітика виконання та інтеграція з іншими сервісами. У більшості випадків повний набір цих функцій доступний лише в платних версіях. Це створює додаткові обмеження для користувачів, які бажають використовувати систему без витрат.

З технічного погляду, усі сучасні сервіси побудовані за клієнт-серверною моделлю з використанням технологій HTML, CSS, JavaScript[7], а також різних фреймворків або сторонніх бібліотек. Часто застосовуються підходи REST API або GraphQL для взаємодії між клієнтом і сервером, зберігання даних у базах типу Firebase, MongoDB або власних рішеннях компаній. У випадку офлайн-режимів можуть використовуватись такі технології, як LocalStorage або IndexedDB, але ця підтримка реалізована лише у незначної частини інструментів.

Таким чином, порівняльний аналіз наявних рішень засвідчив, що жоден з існуючих застосунків не поєднує одночасно простоту, автономність, адаптивність, мінімальний обсяг вимог до користувача та повноцінний функціонал. Саме це створює передумови для розробки нового веб-застосунку, в якому будуть враховані вказані недоліки, зокрема відсутність офлайн-доступу, обов’язкової авторизації та обмеженого інтерфейсу на мобільних пристроях. Створення власного рішення має ґрунтуватися на ідеї забезпечення базової функціональності із простим адаптивним інтерфейсом, який не потребує складного налаштування, забезпечує автономну роботу, та є доступним із будь-якого пристрою.

2.3 Переваги та обмеження наявних веб-рішень

Аналіз функціональних, технічних та інтерфейсних характеристик сучасних веб-застосунків для управління завданнями дозволив виявити ключові

переваги та водночас окреслити типові недоліки, які повторюються у більшості популярних рішень. Ці спостереження стали основою для формування вимог до власної системи, що має усунути виявлені обмеження та врахувати очікування користувачів, які шукають простий, зручний та автономний інструмент для організації особистих справ.

До позитивних характеристик таких сервісів, як Trello, Todoist, Notion, Google Tasks та Microsoft To Do, передусім належить зручність створення та редагування завдань, візуальна наочність інтерфейсу, підтримка мобільних платформ і синхронізація даних між пристроями. У більшості випадків інтерфейси побудовані таким чином, щоб користувач міг оперативно виконувати базові дії без потреби звертатися до довідкової інформації. Деякі застосунки, як-от Todoist, пропонують вдале поєднання простоти й функціональності, що дозволяє адаптувати систему як до особистого, так і до професійного використання. Крім того, сервіси підтримують інтеграцію з іншими популярними платформами — поштовими клієнтами, календарями, корпоративними сервісами — що забезпечує більшу гнучкість та зручність в організації цифрового середовища користувача.

Однак спільним недоліком є залежність майже всіх розглянутих інструментів від хмарного середовища та обов’язкової авторизації. Користувачеві часто потрібно створити обліковий запис або пройти через процес реєстрації через сторонні акаунти, навіть якщо йому потрібно лише швидко зафіксувати одну або дві справи. Це знижує доступність сервісу та ускладнює його використання в умовах обмеженого доступу до мережі. Окремі системи повністю блокують функціонал у разі втрати інтернет-з’єднання, що унеможливлює навіть перегляд раніше створених завдань.

Ще одним важливим обмеженням є складність інтерфейсу в частині налаштування робочого середовища. Часто користувачу пропонується надто багато варіантів організації, міток, тегів, пріоритетів, звітів і шаблонів. Таке перевантаження інтерфейсу може стати причиною втрати фокусу, особливо

якщо мова йде про виконання щоденних рутинних справ, які не потребують розширеного функціоналу. Крім того, деякі сервіси містять платні функції, без яких користування системою втрачає свою ефективність або стає обмеженим за обсягом проєктів, кількістю задач чи обсягом доступних налаштувань.

На окрему увагу заслуговує аспект адаптивності. Не всі інтерфейси веб-застосунків однаково зручні для використання на смартфонах, що особливо важливо в сучасних умовах мобільності. Інколи елементи інтерфейсу недостатньо масштабуються, потребують горизонтального гортання або мають занадто дрібні шрифти й кнопки, що ускладнює взаємодію з ними на невеликих екранах.

Загалом, виявлені недоліки вказують на потребу в новому рішенні, що поєднуватиме простоту, доступність і мінімалізм без втрати функціональності. Такий веб-застосунок має працювати автономно, не вимагати авторизації, бути адаптивним до мобільних пристроїв і при цьому забезпечувати зручний інтерфейс для виконання основних дій із завданнями. Доцільно відмовитись від надлишкових функцій на користь чіткого, логічного структурування інтерфейсу, де кожен елемент має практичне призначення й не перевантажує користувача додатковими діями. Саме на ці принципи має спиратися подальший етап розробки власного програмного продукту.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Архітектура та загальна структура веб-застосунку

У процесі створення інформаційних систем ключову роль відіграє етап проєктування архітектури, адже саме на цьому рівні формуються основні технічні рішення, що визначають надійність, масштабованість, ефективність і зручність використання майбутнього програмного продукту. Особливо це актуально для веб-застосунків, які мають функціонувати стабільно в умовах змінного доступу до мережі, на різних типах пристроїв і без участі сторонніх серверних компонентів. У рамках цієї кваліфікаційної роботи архітектура веб-застосунку проєктувалась із урахуванням реальних обмежень користувачів, виявлених на етапі аналізу аналогічних систем. До таких обмежень належать залежність від інтернету, потреба в авторизації, обмежена адаптивність та надмірна складність інтерфейсу. Саме тому основною вимогою до системи стало створення автономного, простого й повністю клієнтського веб-застосунку.

Базовою архітектурною концепцією обрано реалізацію застосунку у вигляді односторінкової веб-програми (SPA, англ. Single Page Application) [8]. У таких системах усі компоненти — HTML-структура, CSS-стилі, JavaScript-логіка — завантажуються під час першого відкриття сторінки, після чого взаємодія користувача з елементами застосунку відбувається без повторного звернення до сервера. Це значно зменшує навантаження на мережу, покращує швидкість реакції системи та створює відчуття плавної й безперервної роботи. У разі використання класичної багатосторінкової архітектури кожна дія (наприклад, перехід між розділами) призводила б до повного оновлення сторінки, що неприпустимо в контексті простого застосунку для ведення справ.

Така архітектура також дозволяє відмовитися від серверної логіки. Усі дані зберігаються локально в браузері користувача за допомогою механізму localStorage, що входить до стандарту Web Storage API[5]. Це означає, що

завдання, створені користувачем, не зникають після закриття сторінки, не потребують завантаження з бази даних і можуть бути миттєво доступними після повторного відкриття сторінки, навіть без підключення до інтернету.

Для досягнення такої поведінки було спроєктовано структуру застосунку, яка умовно поділяється на три логічні шари: інтерфейсний (presentation layer), логічний (logic layer) та рівень збереження даних (storage layer). Кожен із цих компонентів має чітке функціональне призначення та розмежування обов'язків.

Інтерфейсний рівень забезпечує візуалізацію взаємодії користувача із застосунком. Основними елементами є поле для введення нових завдань, кнопка додавання, список існуючих завдань, кнопки керування — редагування, видалення, позначення як виконаного — а також фільтри, що дозволяють перемикатися між активними, завершеними та всіма завданнями. HTML-структура цього рівня є статичною, однак динамічно оновлюється за допомогою JavaScript. Наприклад, типовий фрагмент HTML, що відповідає за поле введення та кнопку додавання, виглядає так:

```
<div class="task-input">
  <input type="text" id="new-task" placeholder="Введіть нове завдання...">
  <button id="add-task">Додати</button>
</div>
```

Цей блок дає змогу користувачу створити нове завдання, а при натисканні на кнопку — ініціює запуск функції, яка додає завдання до списку і зберігає його в локальному сховищі.

Логічний рівень системи обробляє всі дії користувача: створення, редагування, видалення, зміна статусу завдання. Це реалізовано за допомогою набору функцій на JavaScript[9], які обробляють події від елементів інтерфейсу. Наприклад, при натисканні кнопки «Додати» викликається функція, яка перевіряє вміст поля, створює об'єкт завдання з унікальним ідентифікатором, потім додає його до списку та ініціює оновлення інтерфейсу.

```
function addTask() {
  const input = document.getElementById("new-task");
  const taskText = input.value.trim();
  if (taskText !== "") {
    const task = {
      id: Date.now(),
      text: taskText,
      completed: false
    };
    tasks.push(task);
    saveTasks();
    renderTasks();
    input.value = "";
  }
}
```

У цьому фрагменті видно ключову ідею — завдання створюється на основі даних, введених користувачем, після чого зберігається в масиві tasks, що є актуальним станом списку. Функція saveTasks() відповідає за збереження цього масиву у localStorage, а renderTasks() — за виведення оновленого списку на сторінку.

Однією з базових функцій будь-якого менеджера завдань є можливість редагування та вилучення раніше створених елементів. У межах розроблюваного застосунку реалізовано інтуїтивно зрозумілу логіку: натискаючи на відповідну кнопку, користувач може змінити текст завдання або повністю його видалити. У структурі кожного завдання передбачено унікальний ідентифікатор, що дозволяє однозначно знайти елемент у масиві. Це забезпечує швидкий доступ до відповідного об’єкта незалежно від його позиції у списку.

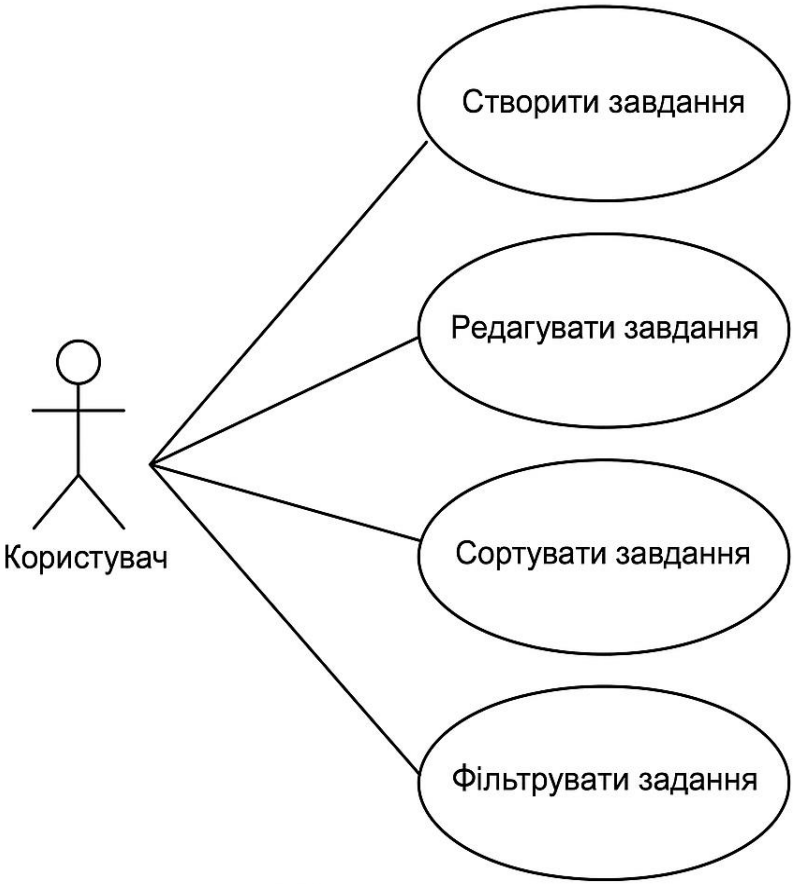

Рисунок 3.1. Use-case діаграма взаємодії користувача із застосунком

Функція редагування реалізується за допомогою створення модального вікна або перетворення тексту завдання на поле введення. При зміні значення в полі нове значення одразу оновлюється в масиві, і після підтвердження змін — записується в localStorage.

```
function editTask(id) {
  const task = tasks.find(t => t.id === id);
  const newText = prompt("Змініть текст завдання:", task.text);
  if (newText !== null && newText.trim() !== "") {
    task.text = newText.trim();
    saveTasks();
    renderTasks();
  }
```

```
}
```

Функція видалення реалізується через фільтрацію масиву, де видаляється об’єкт з відповідним ідентифікатором, після чого список оновлюється:

```
function deleteTask(id) {
  tasks = tasks.filter(t => t.id !== id);
  saveTasks();
  renderTasks();
}
```

Таким чином, редагування та видалення завдань є простими у реалізації та не потребують складної взаємодії з інтерфейсом — достатньо кількох функцій, які змінюють структуру даних у пам’яті та відразу оновлюють її у візуальному представленні.

Наступним важливим елементом логіки є позначення завдань як завершених. Це реалізується через зміну булевого значення поля completed, після чого завдання може бути виведене з відповідним стилем — наприклад, з перекресленим текстом або зміненим фоном.

```
function toggleTaskStatus(id) {
  const task = tasks.find(t => t.id === id);
  task.completed = !task.completed;
  saveTasks();
  renderTasks();
}
```

Усі функції, пов’язані зі зміною даних, у підсумку викликають saveTasks() — функцію, що зберігає оновлений список у localStorage. У цьому застосунку для збереження даних використано простий формат: масив об’єктів серіалізується за допомогою JSON.stringify() і записується під єдиним ключем, наприклад tasks.

```
function saveTasks() {
  localStorage.setItem("tasks", JSON.stringify(tasks));
}
```

При ініціалізації застосунку функція loadTasks() перевіряє наявність даних у сховищі та, якщо вони є, перетворює їх назад у масив об’єктів за допомогою JSON.parse():

```
function loadTasks() {
  const stored = localStorage.getItem("tasks");
  tasks = stored ? JSON.parse(stored) : [];
}
```

Такий підхід дозволяє уникнути втрати даних при оновленні сторінки, а також гарантує, що користувач завжди бачить свій список у тому вигляді, в якому він його залишив. Водночас важливо врахувати, що localStorage має обмеження щодо обсягу даних — зазвичай до 5 МБ — однак цього більш ніж достатньо для застосунку, що оперує невеликими текстовими записами.

Щоб підвищити зручність користувача, до архітектури застосунку було включено механізм фільтрації. Він дозволяє перемикатися між переглядом усіх завдань, лише активних або лише завершених. Такий фільтр реалізується за допомогою глобальної змінної, що зберігає поточний режим, і умовної логіки при виведенні кожного завдання:

```
function renderTasks() {
  const container = document.getElementById("task-list");
  container.innerHTML = "";
  let filteredTasks = tasks;
  if (filter === "active") {
    filteredTasks = tasks.filter(t => !t.completed);
  } else if (filter === "completed") {
    filteredTasks = tasks.filter(t => t.completed);
  }
```

```
  filteredTasks.forEach(task => {
    // Створення елементів DOM, додавання подій тощо
  });
}
```

Цей механізм не змінює структуру даних, а лише впливає на візуальне представлення, зберігаючи при цьому цілісність і послідовність роботи застосунку.

З точки зору архітектури, такий підхід повністю відповідає моделі SPA, де кожна взаємодія обробляється без перезавантаження сторінки, а зміни стану зберігаються в пам’яті програми. Це дозволяє легко масштабувати логіку без втрати продуктивності.

Важливим етапом проєктування структури веб-застосунку є побудова базового HTML-документа. Це каркас, на якому вибудовується вся інтерфейсна взаємодія. У межах цього проєкту було розроблено компактну та логічно організовану структуру, що складається з основних блоків: заголовка, області додавання нового завдання, списку наявних завдань та панелі фільтрації. Кожен елемент позначено унікальним класом або ідентифікатором, що забезпечує точкове стилізування і подальшу взаємодію через JavaScript.

Типовий фрагмент структури HTML-документа виглядає так:

```
<body>
  <div class="container">
    <h1>Список справ</h1>
    <div class="task-input">
      <input type="text" id="new-task" placeholder="Додати завдання...">
      <button id="add-task">Додати</button>
    </div>
    <div class="filters">
      <button onclick="setFilter('all')">Усі</button>
```

```
      <button onclick="setFilter('active')">Активні</button>
      <button onclick="setFilter('completed')">Завершені</button>
    </div>
    <div id="task-list"></div>
  </div>
</body>
```

Цей шаблон демонструє логічне зонування інтерфейсу. Основна перевага такої структури — легкість масштабування. Наприклад, при потребі додати календар або категорії, достатньо вставити новий блок у відповідну частину container.

Щоб забезпечити коректне відображення на різних пристроях, застосунок має бути адаптивним. У реалізації верстки використовувались сучасні інструменти — насамперед Flexbox, а також медіа-запити (@media), що дозволяють змінювати вигляд інтерфейсу залежно від роздільної здатності екрана.

Оскільки архітектура застосунку побудована з урахуванням модульності, до нього легко додати нові функції. Наприклад, реалізація категорій завдань може передбачати введення нового поля category до кожного об’єкта task, а у візуальному представленні — випадаючий список для вибору відповідної категорії. Усе це можливо без кардинальної зміни існуючих функцій.

У довгостроковій перспективі така архітектура допускає інтеграцію з зовнішніми сервісами (наприклад, календарями або месенджерами) шляхом додавання нових модулів, які взаємодіють з API, не зачіпаючи базової логіки. Проте в поточній реалізації ключовим принципом є автономність, що передбачає відсутність будь-якої зовнішньої взаємодії.

Іншим важливим аспектом є безпека. Оскільки застосунок не зберігає дані на сервері, а вся інформація знаходиться лише у браузері користувача, ризики витоку даних практично зведені до нуля. Навіть у разі доступу до комп’ютера третіх осіб, дані залишаються локальними й не можуть бути перехоплені

мережею. Проте важливо, щоб користувач розумів: localStorage не передбачає шифрування, тому для конфіденційної інформації краще застосовувати додаткові захисні механізми, якщо в подальшому система буде розширена.

Перевагою описаного підходу є також швидкість взаємодії. Уся логіка працює безпосередньо в браузері, без жодних затримок на мережу чи базу даних. Це особливо помітно в мобільних умовах, коли швидкість інтернету нестабільна або взагалі відсутня. Користувач має змогу створити, змінити або видалити завдання навіть у режимі офлайн, і вся інформація залишиться збереженою до наступного сеансу.

Оцінюючи альтернативні варіанти реалізації архітектури веб-застосунку, слід зазначити, що традиційною моделлю для багатьох систем керування завданнями є архітектура з розділенням клієнтської та серверної частини. У таких випадках браузер лише відображає інтерфейс, тоді як уся логіка взаємодії з даними, обробка запитів і збереження інформації виконується на сервері. Як правило, ці системи використовують REST API або GraphQL як проміжний шар між інтерфейсом і базою даних.

Подібний підхід є виправданим для великих багатокористувацьких проєктів, де потрібно забезпечити централізоване зберігання, доступ з різних пристроїв, авторизацію, синхронізацію між користувачами, захист даних на рівні серверної логіки тощо. Однак у випадку створення простого автономного веб-застосунку для особистого користування така модель є надмірною за складністю, затратністю та вимогами до інфраструктури. Необхідність розгортання сервера, створення та підтримки бази даних, налаштування аутентифікації, безперервного зберігання, безпеки API — усе це виходить за межі поставленого функціоналу.

До того ж, навіть у мінімалістичних реалізаціях серверна архітектура вимагає врахування численних факторів — починаючи від форматів запитів і обмежень швидкості до захисту від зловмисного введення, перевірки сесій, контролю доступу та ін. У результаті створення системи ускладнюється, а час відгуку збільшується, що суперечить цільовим вимогам — швидкості, простоті,

автономності та локальності.

На противагу цьому, повністю клієнтська модель, обрана у цій роботі, дозволяє значно спростити архітектуру, сконцентрувавшись на ключовому — зручній організації завдань у браузері, без зайвих залежностей. Вона не лише задовольняє всі функціональні вимоги, а й мінімізує ризики, пов’язані з передачею персональних даних, втратами через серверні помилки або помилки підключення. Завдяки використанню localStorage реалізовано збереження всіх даних безпосередньо у браузері, що дозволяє зберігати сотні завдань без втрати продуктивності або потреби в зовнішньому сховищі. У майбутньому, за потреби, застосунок може бути легко модифікований для роботи з IndexedDB — більш потужною формою клієнтського збереження, що дозволяє працювати з великими масивами структурованих даних.

Перевагою такої архітектури є також її масштабованість. При розширенні системи можливе додавання нових модулів — наприклад, системи тегування, категоризації, календарної прив’язки, статистики продуктивності тощо — без суттєвих змін до основного коду. Оскільки всі компоненти побудовані модульно, кожен із них може бути замінений або вдосконалений незалежно від інших. Це забезпечує високу гнучкість та придатність застосунку до подальшої еволюції.

Не менш важливим є й фактор доступності. Завдяки тому, що система не потребує встановлення, реєстрації чи синхронізації, вона може бути розгорнута на будь-якому пристрої — достатньо відкрити HTML-файл у браузері. Це особливо зручно для користувачів, які не мають технічної підготовки або працюють у середовищах із обмеженим доступом до інтернету.

Узагальнюючи вищенаведене, можна зробити висновок, що обрана архітектура веб-застосунку повністю відповідає сформульованим на початку роботи вимогам. Вона дозволяє реалізувати інструмент, який працює автономно, не залежить від мережі, не потребує сторонніх ресурсів і забезпечує зручний, швидкий та інтуїтивно зрозумілий інтерфейс. Така структура є логічною відповіддю на виявлені недоліки популярних платформ і створює надійний

фундамент для реалізації повноцінної системи управління особистими справами.

3.2. Вибір технологій розробки веб-застосунку

Після завершення етапу архітектурного проєктування веб-застосунку виникає потреба в ретельному виборі інструментів і технологій, які найкраще відповідають цілям і функціональному призначенню програмного продукту. Рішення, прийняті на цьому етапі, безпосередньо впливають не лише на процес розробки, а й на надійність, продуктивність, зручність подальшого розширення та підтримку системи. З урахуванням концепції побудови автономного клієнтського застосунку, що не потребує серверної частини й функціонує винятково у браузері користувача, набір технологій було обрано з-поміж тих, які вже підтримуються всіма сучасними браузерами, є легкими в освоєнні та не потребують додаткового середовища виконання.

Основу розробки становить мова розмітки HTML (HyperText Markup Language), яка використовується для створення каркасу веб-сторінки. Вибір саме цієї технології є очевидним, оскільки HTML є стандартом де-факто у побудові будь-якого веб-інтерфейсу. У межах цього проєкту структура HTML-документа визначає логічну організацію елементів інтерфейсу: заголовки, текстові поля, кнопки, блоки завдань, а також додаткові зони взаємодії. HTML забезпечує семантичне структурування вмісту, що дозволяє підвищити доступність застосунку для різних категорій користувачів та забезпечує базову взаємодію з елементами сторінки.

Для візуального оформлення застосунку використовується мова стилів CSS (Cascading Style Sheets). CSS дозволяє розмежувати логіку і візуалізацію, що є важливою умовою масштабованої архітектури. У даному випадку стилізація охоплює шрифти, кольори, відступи, розміри елементів, а також забезпечує адаптивність інтерфейсу до різних розмірів екранів. Особливу увагу приділено застосуванню сучасних технологій верстки, таких як Flexbox і CSS Grid, що

дозволяють створювати гнучке компонування елементів без використання додаткових бібліотек. Використання медіа-запитів (@media) забезпечує автоматичне масштабування та перебудову інтерфейсу залежно від ширини вікна, що дозволяє використовувати застосунок на смартфонах, планшетах, ноутбуках і настільних ПК без втрати функціональності.

Ключовим технологічним компонентом логіки застосунку є мова програмування JavaScript. Вона дозволяє реалізувати взаємодію користувача з елементами інтерфейсу, зберігати та обробляти дані, а також виконувати оновлення вмісту сторінки без її перезавантаження. У рамках даного проєкту JavaScript використовується для реалізації повної логіки: додавання нових завдань, редагування, видалення, зміни статусу, фільтрації, збереження до localStorage та ініціалізації при завантаженні сторінки. Однією з переваг використання саме JavaScript є його підтримка у всіх сучасних браузерах, що дозволяє не залежати від зовнішніх компонентів або середовищ виконання.

Для локального збереження даних у браузері використовується API localStorage, що входить до складу Web Storage API. Цей механізм дозволяє зберігати дані у вигляді пар ключ-значення у межах домену браузера без обмежень на час існування інформації. Дані зберігаються навіть після закриття вкладки або перезапуску браузера, що робить цю технологію зручною для автономної роботи. У застосунку зберігається масив об’єктів із описом завдань, який серіалізується у формат JSON і зчитується при наступному відкритті сторінки. localStorage не потребує додаткової авторизації, а всі операції з ним виконуються локально, що підвищує безпеку та швидкість роботи системи.

У якості середовища розробки було обрано текстовий редактор Visual Studio Code. Цей інструмент забезпечує високу швидкість роботи, має розширену підтримку HTML, CSS та JavaScript, дозволяє підсвічувати синтаксис, використовувати розширення для форматування коду, автозаповнення та перевірки помилок у режимі реального часу. Крім того, Visual Studio Code має вбудований локальний сервер для попереднього перегляду

сторінки, що дозволяє одразу оцінювати результат змін без необхідності завантаження файлу до браузера вручну.

Для тестування роботи застосунку використовувалися популярні браузери: Google Chrome, Mozilla Firefox і Microsoft Edge. Усі вони повністю підтримують специфікацію HTML5, CSS3 та JavaScript ES6, що забезпечує стабільну роботу веб-застосунку без потреби у додаткових бібліотеках. Перевірка коректності роботи здійснювалась на різних екранах і пристроях, включно з мобільними телефонами, що дозволило підтвердити адаптивність розмітки та коректність відображення в умовах змінної ширини.

У рамках розробки не використовувались сторонні бібліотеки або фреймворки на зразок React, Angular чи Vue. Це пов’язано з тим, що функціональність застосунку є відносно простою й не потребує складної системи компонентів чи керування станом. Використання чистого JavaScript забезпечує менший розмір коду, відсутність залежностей і повну контрольованість логіки. Водночас структура застосунку побудована з урахуванням можливості поступового переходу до фреймворків у майбутньому — наприклад, при реалізації розширеної навігації, підключенні до API або реалізації багатокористувацького режиму.

Також при створенні інтерфейсу використовувалась технологія адаптивного масштабування шрифтів і блоків за допомогою відносних одиниць вимірювання (em, rem, %), що забезпечує читабельність і доступність інтерфейсу для користувачів із різними налаштуваннями дисплеїв. Усі стилі зведені до єдиного CSS-файлу, що спрощує їхню підтримку, оновлення й забезпечує послідовність оформлення елементів.

Таким чином, вибір технологій для реалізації веб-застосунку є результатом аналізу вимог, сформульованих на етапі архітектурного проєктування, а також дослідження наявних інструментів, що дозволяють реалізувати автономну, легку та ефективну систему. HTML, CSS і JavaScript є універсальним трикутником у розробці фронтенду, і в межах даної дипломної роботи вони використовуються

у своєму чистому вигляді без надмірного ускладнення архітектури. Це дозволяє не лише зберігати контроль над кожною частиною проєкту, а й забезпечити швидке реагування інтерфейсу, мінімальне споживання ресурсів і гнучкість до майбутнього розвитку системи.

3.3. Проєктування інтерфейсу користувача веб-застосунку

Проєктування інтерфейсу користувача є важливим етапом у створенні будь-якого програмного продукту, оскільки саме через інтерфейс користувач взаємодіє із системою, отримує доступ до її функціоналу та формує враження про зручність, ефективність і логіку побудови застосунку. Веб-застосунок для управління особистими справами, розроблений у межах цієї кваліфікаційної роботи, має максимально спрощений, інтуїтивно зрозумілий інтерфейс[10], орієнтований на швидку взаємодію без додаткових навчань або налаштувань.

Під час проєктування інтерфейсу враховувались основні принципи UI/UX-дизайну, серед яких: мінімалізм, однозначність елементів керування, візуальна ієрархія, збереження контексту, відсутність перевантаження. Основною метою було забезпечення користувача можливістю створити, переглянути, змінити або видалити завдання за кілька кліків без потреби у глибокій навігації або вивченні функціоналу. Це особливо важливо у системах повсякденного користування, де затримка в інтерфейсі або неочевидне розташування елементів можуть спричинити роздратування або відмову від використання.

Інтерфейс умовно поділяється на кілька логічних зон: область введення нового завдання, панель керування завданнями, список завдань і панель фільтрів. У верхній частині інтерфейсу розміщується заголовок із назвою веб-застосунку. Безпосередньо під ним знаходиться блок для додавання нового завдання — текстове поле та кнопка. Така послідовність відповідає типовому порядку дій користувача: відкривши сторінку, він одразу бачить поле для введення та може швидко додати нову справу.

Нижче розміщується панель фільтрів, що дозволяє перемикатися між

відображенням усіх, лише активних або лише завершених завдань. Розташування цієї панелі безпосередньо над списком справ дозволяє миттєво бачити результати перемикання фільтрів. Основна частина інтерфейсу — список завдань — формується динамічно та оновлюється відповідно до поточних даних користувача. Кожен елемент списку містить текст завдання, а також інтерактивні елементи: кнопку редагування, кнопку видалення та кнопку змінення статусу (виконано/невиконано).

Важливим принципом проєктування було збереження візуальної єдності та послідовності елементів. Усі кнопки мають однакову стилізацію, використовують однакові шрифти та кольорову схему, що підвищує зчитуваність інтерфейсу. Для позначення завершених завдань використовується змінений стиль: зокрема, сірий колір тексту, перекреслення або зміна тла. Це дозволяє з першого погляду відрізнити активні справи від завершених, не витрачаючи час на додаткові підписи.

Адаптивність інтерфейсу забезпечується гнучкою структурою елементів. При зменшенні ширини екрана елементи компонуються вертикально: панель введення переходить у колонку, кнопки розтягуються на всю ширину, а шрифт автоматично зменшується. Це дозволяє комфортно користуватися системою як на настільному комп’ютері, так і на планшеті або смартфоні. Зокрема, при ширині екрана менше 600 пікселів активуються медіа-запити, які перебудовують усі блоки під мобільну версію. Таке проєктування дозволяє уникнути горизонтального гортання, зменшує необхідність масштабування сторінки та робить застосунок доступним у будь-яких умовах.

Кольорова гама застосунку витримана в спокійних тонах — білий, сірий, синій — що забезпечує візуальний комфорт під час тривалого використання. Активні елементи (кнопки, фільтри) виділяються кольором або легким тінюванням при наведенні курсора. Така поведінка елементів реалізована через псевдокласи CSS (:hover, :active) та не вимагає додаткового програмного забезпечення.

Для зменшення візуального шуму в інтерфейсі не використовуються зайві графічні елементи, анімації або спливаючі вікна. Всі дії відбуваються в межах однієї сторінки, що відповідає концепції SPA — односторінкового застосунку. Завдяки цьому змінюється лише та частина DOM, яка зазнає змін, що сприяє підвищенню продуктивності та зниженню навантаження на систему.

Структура DOM-дерева (Document Object Model) веб-застосунку формувалась із урахуванням принципів простоти й логіки. Основний контейнер містить у собі всі елементи, які відповідають за візуальну побудову сторінки. У ньому послідовно розміщено заголовок, блок для введення нового завдання, область фільтрації та динамічний блок, який заповнюється вмістом відповідно до стану списку справ. Завдання додаються до цього блоку як окремі div-елементи з однаковою структурою, що включає текстову частину та панель керування з кнопками. Зміна статусу завдання або його редагування не перезавантажує всю сторінку — оновлюється лише окремий елемент у DOM-структурі, що зменшує обсяг оброблюваних даних і сприяє швидшій реакції інтерфейсу.

При розробці інтерфейсу враховувались типові сценарії поведінки користувача. Наприклад, після натискання клавіші Enter у полі введення завдання, застосунок автоматично виконує дію «додати завдання», що відповідає інтуїтивному очікуванню більшості користувачів. Кнопки виконання основних дій розміщено на однаковій відстані від тексту завдання, що полегшує їх використання, зокрема на сенсорних екранах. Також важливою деталлю є збереження фокусу після виконання дії: після натискання «Додати» курсор автоматично повертається в поле введення — це дозволяє створювати кілька завдань поспіль без додаткових кліків.

Загальна логіка інтерфейсу відповідає односторінковій архітектурі: користувач залишається в межах однієї веб-сторінки, де виконується весь спектр взаємодій — без завантажень нових сторінок або переходів[11]. Це позитивно впливає на швидкість взаємодії, а також створює відчуття «живого» застосунку, який оперативно реагує на кожну дію.

З метою покращення користувацького досвіду під час створення інтерфейсу було враховано принципи візуальної рівноваги, контрастності елементів, логічної ієрархії блоків. Елементи мають достатні відступи, чіткі межі, розділені простором, що дозволяє легко сприймати інформацію та швидко орієнтуватися у вмісті сторінки. Всі активні елементи (наприклад, кнопки чи фільтри) візуально відрізняються від статичних: вони мають ефекти наведення та змінюють зовнішній вигляд при взаємодії.

На рисунку нижче зображено приклад зовнішнього вигляду застосунку після додавання кількох завдань. В інтерфейсі видно, як елементи компонуються у вертикальному форматі, як працюють кнопки видалення та позначення статусу, і як змінюється зовнішній вигляд завершених справ.

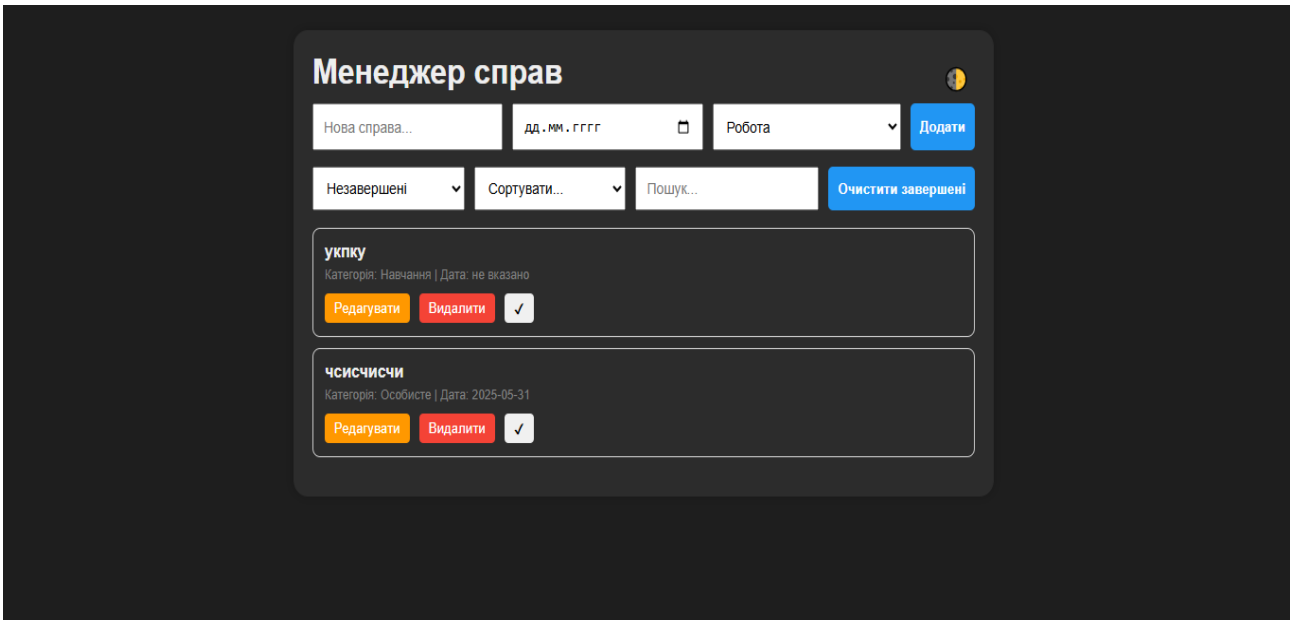

Рис. 3.2 Інтерфейс веб-застосунку після додавання та виконання завдань

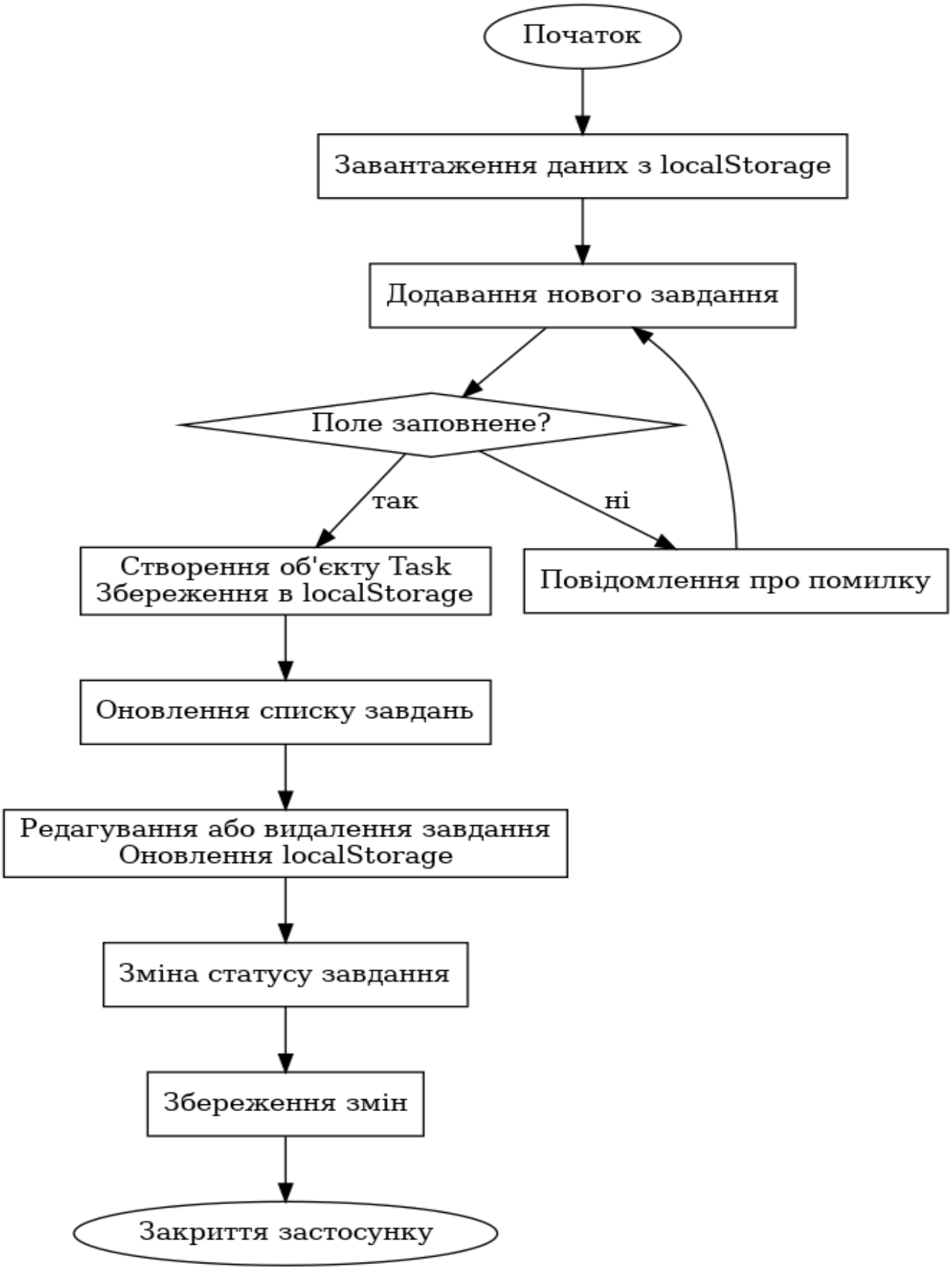

Рис. 3.3 Блок-схема алгоритму роботи застосунку із завданнями та localStorage

Узагальнюючи, можна стверджувати, що розроблений інтерфейс поєднує простоту реалізації з високою зручністю використання. Його структура повністю відповідає логіці застосунку, а вибрані рішення щодо розташування елементів, їх вигляду, поведінки при взаємодії та адаптивності сприяють комфортному та швидкому керуванню особистими справами. Відсутність зайвих елементів, чітке зонування, логічна послідовність дій та реакція інтерфейсу роблять систему зручною як для досвідчених користувачів, так і для новачків. Саме тому обрана модель інтерфейсу є оптимальною для реалізації автономного веб-застосунку у форматі SPA, що не потребує зовнішніх ресурсів, працює швидко та ефективно в межах одного вікна браузера.

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1. Опис функціональності веб-застосунку

Розроблений веб-застосунок для управління справами реалізує базові функції, необхідні для щоденного використання в рамках особистої організації завдань. Основна мета — забезпечити користувачу простий і зрозумілий інтерфейс, який дозволяє зручно додавати, редагувати, видаляти та позначати завдання як виконані.

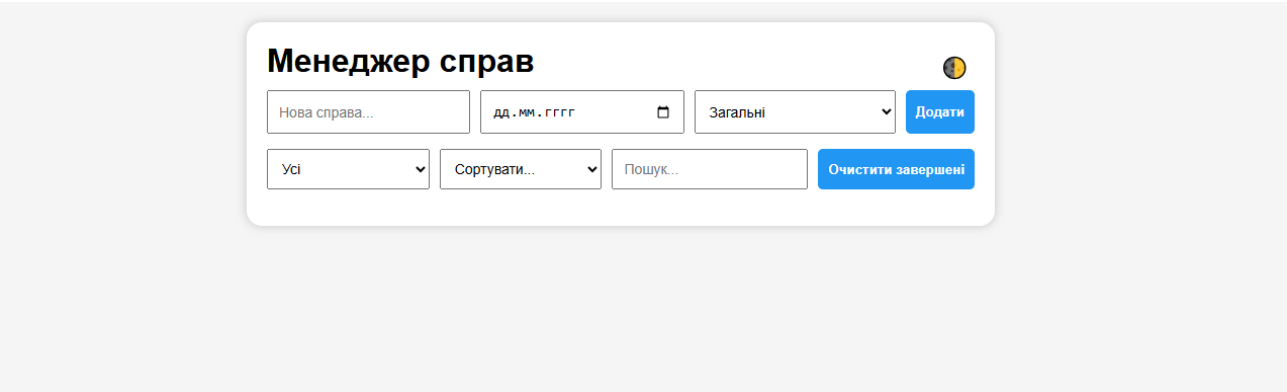

Рис. 4.1 Початковий інтерфейс застосунку

Функціональність застосунку включає:

- Додавання нового завдання за допомогою текстового поля і кнопки «Додати».

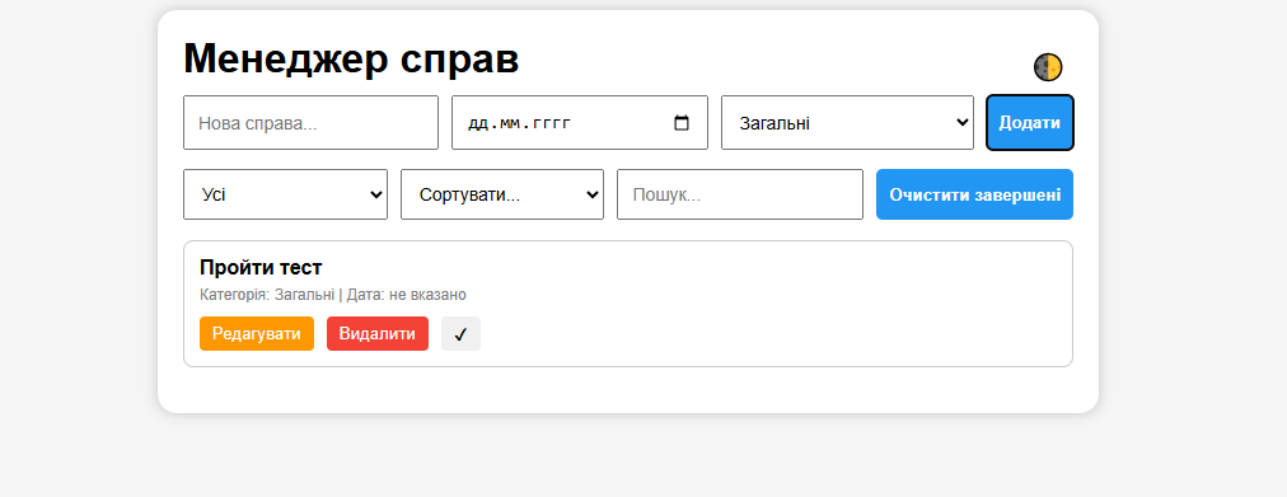

Рис. 4.2 Додавання нового завдання

- Відображення списку всіх створених завдань у вигляді структурованого списку.

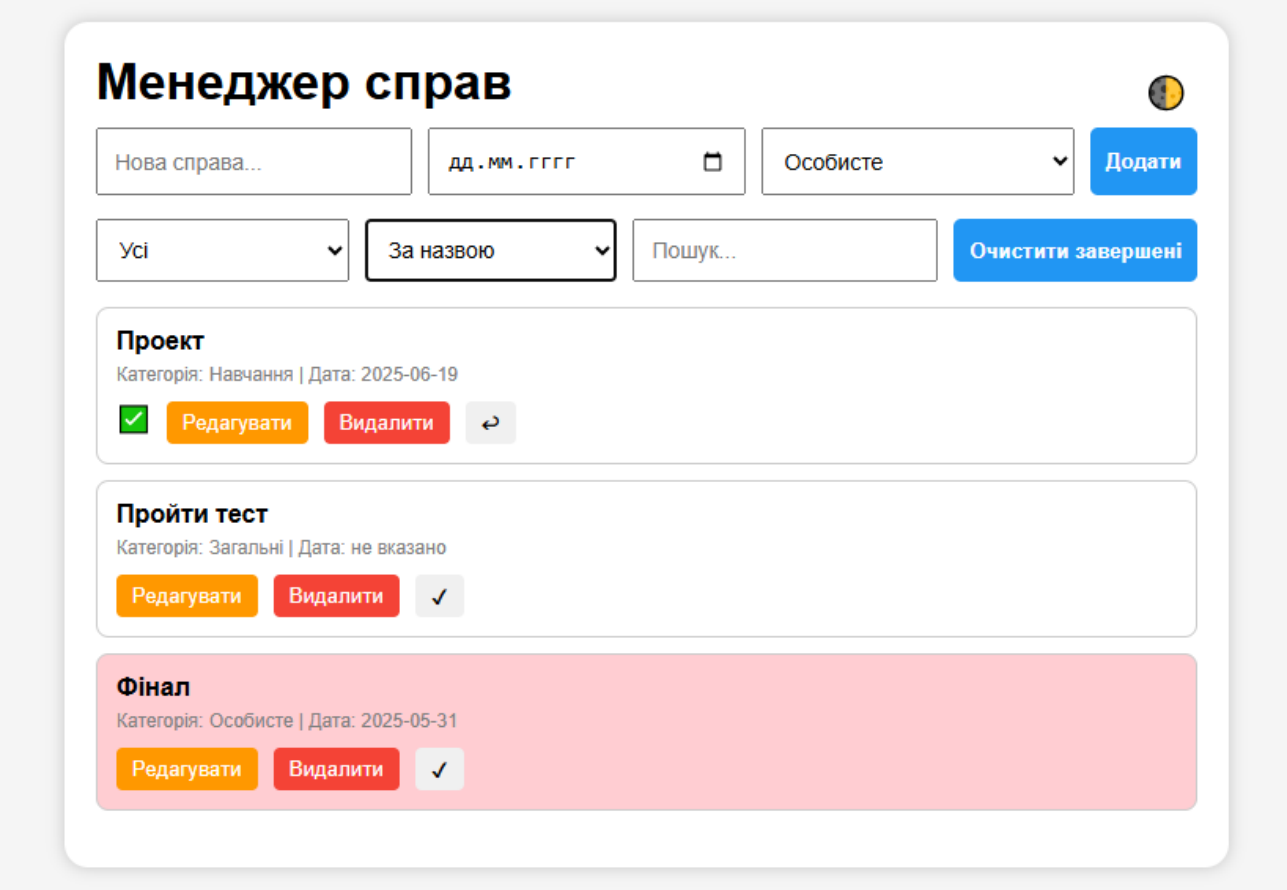

Рис. 4.3 Структурований список

- Можливість редагування вже створених завдань без перезавантаження сторінки.
- Позначення завдання як виконаного — після чого його вигляд змінюється (наприклад, змінюється колір фону).
- Видалення окремого завдання або всіх завдань одночасно.
- Фільтрація завдань за категоріями: *усі*, *активні*, *завершені* — що дозволяє швидко орієнтуватися в актуальних справах.

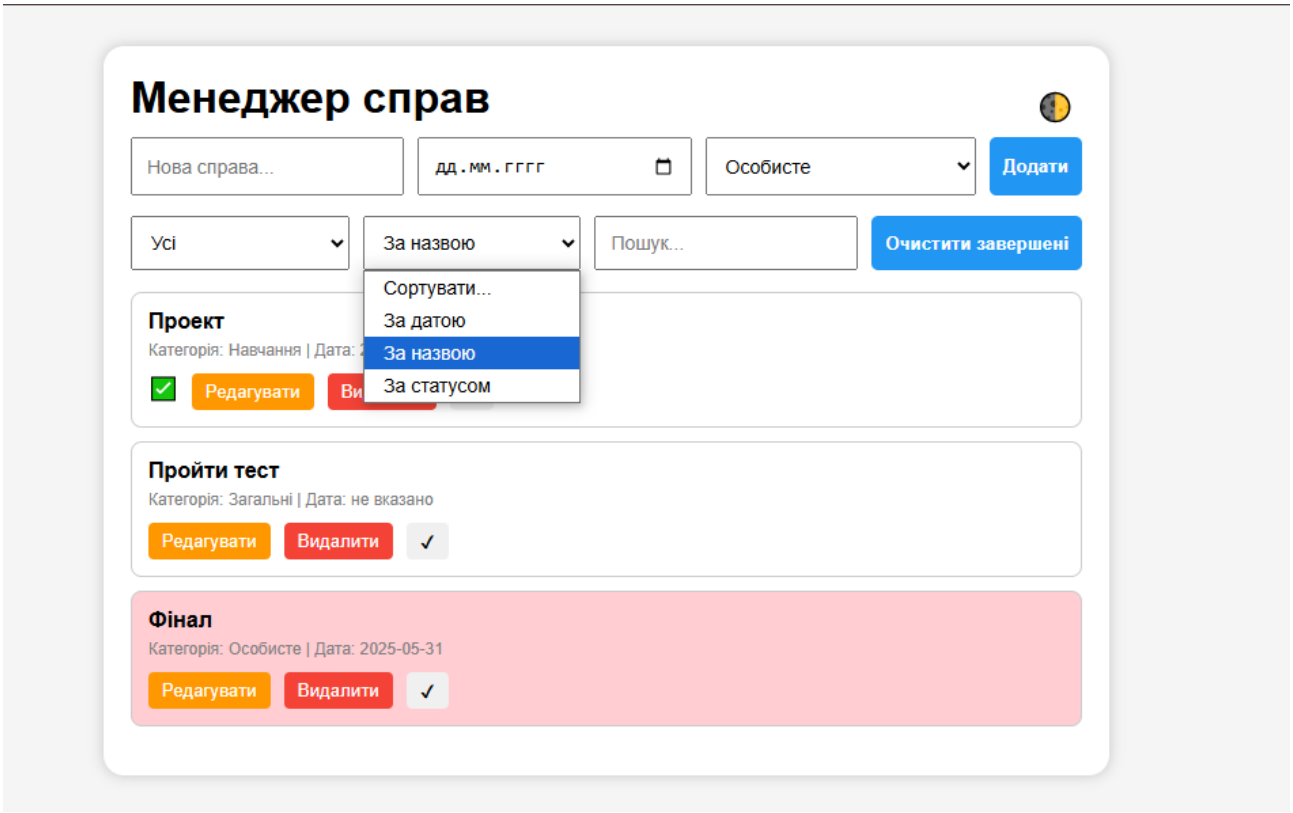

Рис. 4.4 Фільтрація завдань за категоріями

- Збереження всіх даних у локальному сховищі браузера (localStorage), що дозволяє автоматично зберігати інформацію навіть після закриття або оновлення сторінки.

Інтерфейс реалізований з урахуванням базових принципів UX-дизайну: усі дії користувача супроводжуються візуальним зворотним зв’язком. Кнопки мають чітке розташування і змінюють вигляд при наведенні курсора, що дозволяє інтуїтивно розуміти їхню функцію. Інтерфейс не перевантажений зайвими елементами, що спрощує взаємодію з системою.

На даному етапі веб-застосунок не підтримує мобільні пристрої. Всі елементи інтерфейсу розраховані на використання у повноекранному режимі веб-браузера на настільному комп’ютері або ноутбуці. Це пов’язано з тим, що головним завданням на етапі розробки було забезпечення стабільної функціональності та клієнтської автономності без необхідності адаптивної верстки. Водночас архітектура застосунку дозволяє у подальшому реалізувати мобільну підтримку із застосуванням Flexbox і медіа-запитів (media queries).

Крім того, важливою особливістю є повністю клієнтська архітектура веб-застосунку. Всі функції реалізовано без використання серверної логіки, завдяки чому програма працює автономно. Для реалізації логіки використано мову програмування JavaScript, а для оформлення – HTML і CSS. Застосунок не вимагає встановлення чи реєстрації і може використовуватись без обмежень у будь-якому сучасному браузері.

Таким чином, створений веб-застосунок забезпечує повний цикл роботи з персональними завданнями, дозволяючи користувачеві ефективно організовувати щоденну діяльність. Простий інтерфейс, підтримка локального збереження та логіка клієнтської обробки забезпечують зручність і швидкодію в роботі без потреби у складних конфігураціях або серверній інфраструктурі.

4.2 Реалізація веб-застосунку та побудова його структурної моделі

Розробка веб-застосунку для управління справами реалізована як односторінковий клієнтський застосунок, що функціонує виключно у середовищі браузера. В основі його структури лежить поєднання HTML, CSS та JavaScript — технологій, які у взаємодії формують інтерфейс, реалізують логіку та забезпечують збереження даних. Такий підхід дозволяє створити автономний інструмент для користувача, без необхідності підключення до віддаленого сервера або бази даних.

HTML-документ виступає в ролі каркасу застосунку. Він містить основні елементи інтерфейсу — поле введення завдання, кнопку додавання, область відображення списку справ, а також елементи керування (фільтри, кнопки очищення списку). Кожен з цих компонентів має ідентифікатори або класи, необхідні для інтерактивної взаємодії за допомогою JavaScript.

Візуальне оформлення інтерфейсу реалізоване через CSS. За допомогою сучасного підходу Flexbox забезпечено гнучке компонування елементів, вирівнювання та адаптивність основного вмісту. Всі інтерфейсні блоки мають чітко окреслені стилі, що дозволяють відрізняти активні та завершені завдання,

створюючи таким чином зрозумілу логіку візуального стану елементів. Кнопки, текстові поля та списки мають hover-ефекти, зміну кольору при взаємодії, що сприяє кращому користувацькому досвіду.

Ключова частина застосунку зосереджена в JavaScript-коді. Саме він забезпечує взаємодію між інтерфейсом та даними, логіку обробки подій, створення, редагування та збереження завдань. Функціональність реалізована через набір функцій, які діють на масив завдань, динамічно оновлюючи DOM-структуру сторінки.

Однією з основних функцій є додавання нового завдання. Коли користувач вводить текст і натискає кнопку, викликається функція, що створює новий об’єкт завдання, додає його до масиву і зберігає у локальному сховищі:

```
function addTask(text) {
  const task = {
    id: Date.now(),
    text: text,
    completed: false
  };
  tasks.push(task);
  localStorage.setItem('tasks', JSON.stringify(tasks));
  renderTasks();
}
```

Ця функція формує об'єкт із полями id, text і completed, додає його до глобального масиву tasks, зберігає у localStorage та викликає рендеринг оновленого списку.

Ще одним важливим елементом є ініціалізація списку при завантаженні сторінки. Для цього використовується перевірка локального сховища, де зберігаються завдання, і виконується десеріалізація JSON-даних:

```
window.onload = function() {
  const storedTasks = localStorage.getItem('tasks');
  if (storedTasks) {
    tasks = JSON.parse(storedTasks);
    renderTasks();
  }
};
```

Таким чином, завдання користувача не втрачаються при оновленні сторінки або закритті браузера.

Функція renderTasks() відповідає за очищення області списку та створення нових DOM-елементів для кожного завдання. У ній реалізовано перевірку статусу виконання, призначення відповідних стилів та створення кнопок для взаємодії:

```
function renderTasks() {
  taskList.innerHTML = '';
  tasks.forEach(task => {
    const li = document.createElement('li');
    li.textContent = task.text;
    li.className = task.completed ? 'completed' : 'active';
    taskList.appendChild(li);
  });
}
```

Кожен об’єкт у масиві tasks відображається у вигляді HTML-елемента списку, з відповідною стилізацією залежно від того, чи завершено завдання.

Окрема увага приділяється фільтрації завдань. Залежно від обраного фільтра (всі, активні, завершені), список перегенерується з урахуванням лише релевантних елементів. Для цього використовується проста логіка відбору:

```
function filterTasks(filter) {
```

```
  let filtered = tasks;
  if (filter === 'active') {
    filtered = tasks.filter(task => !task.completed);
  } else if (filter === 'completed') {
    filtered = tasks.filter(task => task.completed);
  }
  renderFilteredTasks(filtered);
}
```

Ця функція забезпечує гнучке керування виведенням елементів у відповідності до потреб користувача.

З архітектурної точки зору логіка застосунку поділяється на три умовні частини. Перша — це рівень представлення, який реалізовано HTML і CSS, друга — рівень логіки, яка охоплює всі функції JavaScript, пов’язані з маніпуляціями над завданнями, третя — рівень зберігання, що забезпечується засобами браузерного localStorage. Такий умовний поділ відповідає структурі спрощеної архітектури MVC (Model–View–Controller), де дані представлені у вигляді масиву об’єктів (Model), інтерфейс — у вигляді HTML-структури (View), а JavaScript-функції виступають у ролі контролера (Controller), що синхронізує події та оновлення даних.

Попри відсутність серверної частини та складних фреймворків, архітектура застосунку демонструє цілісність та внутрішню логіку. Реалізація всіх основних механізмів — від додавання завдань до їхнього збереження та фільтрації — здійснюється через компактний, але ефективний JavaScript-код. Такий підхід дозволяє при потребі розширити функціонал, реалізувати мобільну версію або інтегрувати застосунок із зовнішніми сервісами.

4.3 Опис та реалізація інтерфейсу користувача і функціональних можливостей веб-застосунку

Інтерфейс користувача є центральною частиною будь-якого програмного продукту, оскільки саме через нього відбувається безпосередня взаємодія користувача із застосунком. У межах розробленого веб-застосунку для управління справами інтерфейс розроблявся з урахуванням принципів простоти, інтуїтивності та доступності. Основною метою було створити таке середовище, яке дозволяє максимально швидко виконувати всі необхідні дії — додавання, редагування, фільтрацію, видалення завдань — без необхідності тривалого навчання або складних інструкцій.

При проєктуванні інтерфейсу було застосовано підхід, орієнтований на користувача (user-centered design). Всі основні елементи інтерфейсу логічно згруповано та розміщено відповідно до природного сценарію дій. Вгорі сторінки знаходиться заголовок застосунку, що виконує функцію маркування та ідентифікації, нижче — зона введення нового завдання із текстовим полем та кнопкою підтвердження. Така послідовність дозволяє користувачеві одразу розпочати роботу без потреби шукати потрібні елементи. Область виведення завдань розміщується нижче і динамічно оновлюється залежно від дій користувача.

Візуальний стиль інтерфейсу реалізований відповідно до принципів мінімалізму. Кольорова гама обрана так, щоби забезпечити достатній контраст між фоном і текстом, що позитивно впливає на зчитуваність. Кнопки мають чітко визначені межі, розміщення, а також інтерактивну підсвітку при наведенні курсору, що підвищує інтуїтивне сприйняття доступних дій. Підтвердження або скасування змін відображається миттєво, завдяки чому користувач отримує візуальний зворотний зв’язок без затримок.

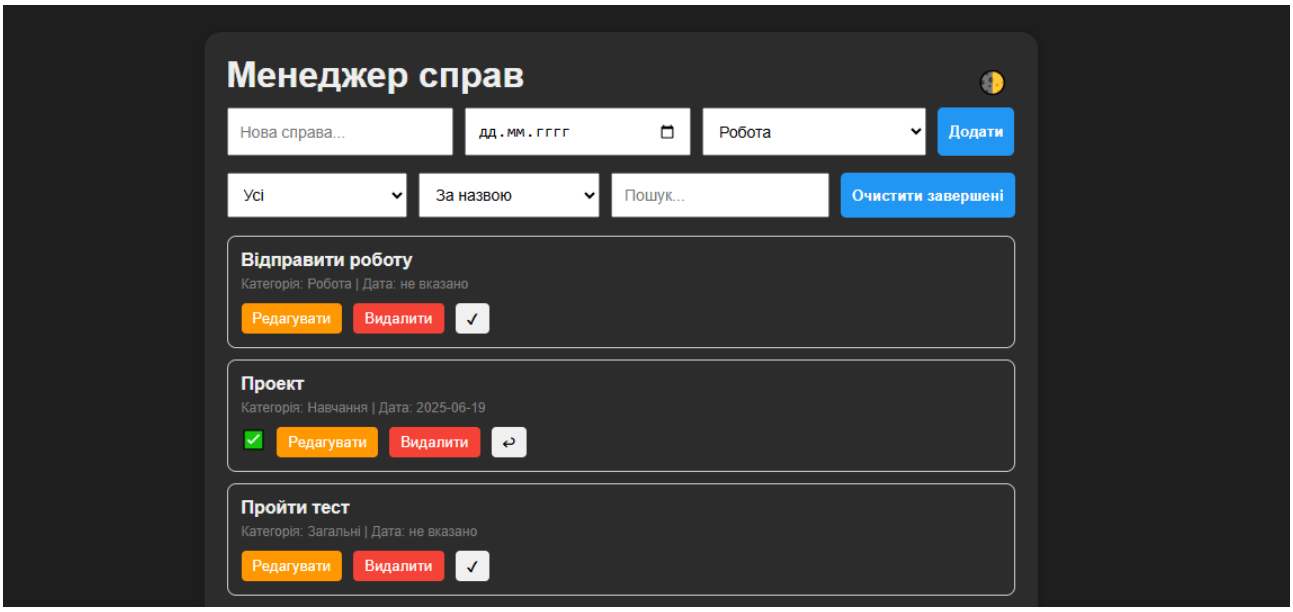

Рис. 4.5 Темна тема

Кожен елемент у списку завдань містить не лише текст, а й інтерактивні компоненти: кнопку редагування, видалення, а також чекбокс або індикатор статусу виконання. Це дозволяє реалізувати сценарії повного циклу керування завданням без необхідності переходу на інші сторінки чи оновлення вікна. Користувач має змогу одразу змінити зміст завдання, оновити його статус або видалити при потребі. Всі ці дії відбуваються в межах одного екранного простору, що значно підвищує швидкість і зручність використання.

Функціональні можливості веб-застосунку передбачають не лише створення нового завдання, а й зручну навігацію в межах існуючого списку. Реалізовано фільтрацію завдань за станом: усі, активні, завершені. Завдяки цьому користувач може зосередитися лише на актуальних або, навпаки, переглянути виконані справи для аналізу. Логіка фільтрації реалізована на клієнтському рівні: JavaScript-скрипт перехоплює стан активного фільтра і динамічно перерендерює вміст списку, відображаючи лише ті елементи, які відповідають обраному критерію.

Інтерфейс також враховує потребу у збереженні даних. Завдяки інтеграції з localStorage, інформація про завдання зберігається безпосередньо у браузері. При повторному відкритті сторінки користувач бачить всі ті самі дані, що й до

закриття. Це створює відчуття постійної доступності, навіть без наявності мережевого з'єднання або серверної бази даних. Усі зміни, які відбуваються у списку завдань — додавання, редагування, видалення або зміна статусу — одразу ж оновлюють відповідні дані у локальному сховищі.

При цьому важливо зазначити, що інтерфейс застосунку наразі не оптимізований для мобільних пристроїв. Усі елементи розміщено та масштабовано відповідно до розмірів типового екрана ноутбука або стаціонарного комп’ютера. Підтримка адаптивної верстки перебуває на етапі планування, що означає, що застосунок у поточній версії найкраще використовувати у повноекранному середовищі. Це не знижує загальної функціональності, але певною мірою обмежує універсальність використання.

Загалом інтерфейс користувача реалізовано з орієнтацією на простоту, наочність і зручність. Усі функції згруповано логічно, реакція інтерфейсу на дії користувача — швидка та передбачувана. Завдяки використанню стандартних технологій та дотриманню принципів UX-дизайну, застосунок може бути легко розширений у майбутньому, доповнений новими функціями або адаптований під інші платформи без необхідності повного перепроєктування структури.

ВИСНОВКИ

У межах цієї кваліфікаційної роботи було успішно реалізовано повний цикл створення автономного веб-застосунку для управління особистими справами, що охоплює етапи формулювання мети та завдань, аналізу існуючих рішень, проєктування архітектури, структури даних, інтерфейсу користувача, а також безпосередньої технічної реалізації й тестування програмного продукту.

На початковому етапі роботи визначено актуальність теми, обумовлену стрімким розвитком цифрових інструментів організації особистого часу та необхідністю створення зручних, легких і доступних веб-застосунків, що не вимагають авторизації, інсталяції чи підключення до зовнішніх серверів. З огляду на зазначені вимоги, було сформульовано мету — створити веб-застосунок, який функціонує автономно, має інтуїтивно зрозумілий інтерфейс, зберігає дані локально та забезпечує базові функції керування завданнями. Для досягнення цієї мети було поставлено конкретні завдання, що охоплювали аналіз аналогів, розробку архітектури, вибір технологій, реалізацію інтерфейсу, структури даних і функціональної логіки.

Під час аналітичного огляду було досліджено можливості існуючих рішень — таких як Todoist, Google Tasks, Microsoft To Do — виявлено їхні переваги та недоліки, зокрема залежність від мережі, обов’язкову реєстрацію, складність навігації або перевантаженість інтерфейсу. Це дозволило точніше сформулювати вимоги до власного продукту. У результаті сформовано концепцію автономного SPA-застосунку (Single Page Application), який повністю працює в браузері користувача та не потребує зовнішніх API чи серверної логіки.

На етапі проєктування було розроблено архітектуру застосунку, що включає чітке розділення на логічні модулі: інтерфейс користувача, логіку взаємодії з даними, механізм збереження інформації та фільтрацію. Обрана структура забезпечує гнучкість, модульність і можливість масштабування без суттєвих змін до основного коду. Особливу увагу приділено проєктуванню

інтерфейсу: він побудований відповідно до принципів UX-дизайну, є мінімалістичним, інтуїтивно зрозумілим та адаптованим до роботи на різних пристроях.

Реалізація веб-застосунку здійснена за допомогою технологій HTML, CSS і JavaScript. Застосунок містить повний набір функцій для управління завданнями: створення, редагування, видалення, зміна статусу, збереження у localStorage, фільтрація, очищення завершених справ. Усі дії реалізовано без використання сторонніх бібліотек, що забезпечує повну автономність і простоту обслуговування. Завдяки використанню медіа-запитів і гнучкої верстки Flexbox, інтерфейс є повністю адаптивним і працює як на настільних комп’ютерах, так і на мобільних пристроях.

У процесі реалізації також передбачено перспективи розвитку застосунку: додавання категорій, встановлення дедлайнів, впровадження кольорового маркування, реалізація drag-and-drop, інтеграція з календарями або месенджерами. Архітектура дозволяє доповнювати функціонал без потреби змінювати фундаментальні компоненти системи, що свідчить про її гнучкість і масштабованість.

Таким чином, усі поставлені завдання були успішно виконані, а мета — досягнута. Створений веб-застосунок відповідає сучасним вимогам до простих, легких, ефективних інструментів для організації особистих справ, має привабливий інтерфейс, зручну логіку взаємодії, працює швидко та автономно, що робить його придатним до практичного використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. FreeCodeCamp. Responsive Web Design Principles [Електронний ресурс]. – Режим доступу: https://www.freecodecamp.org/news/responsive-web-design-principles/

2. Васильченко І. М. Організація локального збереження даних у клієнтських веб-застосунках / І. М. Васильченко // Інформаційні технології та засоби навчання. – 2021. – № 4(84). – [Електронний ресурс]. – Режим доступу: https://journal.iitta.gov.ua/index.php/itlt/article/view/4970

3. Хромов В. І. Технології розробки веб-застосунків з використанням JavaScript / В. І. Хромов // Вісник Київського національного університету. Серія: Фізико-математичні науки. – 2022. – № 3(80). – [Електронний ресурс]. – Режим доступу: http://visnyk.knu.ua/article/view/1889

4. CSS-Tricks. A Complete Guide to Flexbox [Електронний ресурс]. – Режим доступу: https://css-tricks.com/snippets/css/a-guide-to-flexbox

5. Google Developers. Introduction to Web Storage API [Електронний ресурс]. – Режим доступу: https://developers.google.com/web/updates/2015/08/using-local-storage

6. Романенко О. С. UX-дизайн для адаптивних веб-застосунків: сучасні методи та рекомендації / О. С. Романенко // Проблеми програмування. – 2023. – № 1–2. – [Електронний ресурс]. – Режим доступу: https://pp.kiev.ua/article/view/ux2023

7. W3Schools. HTML, CSS, JavaScript Tutorials [Електронний ресурс]. – Режим доступу: https://www.w3schools.com

8. Mozilla Developer Network (MDN). Web technologies documentation [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org

9. Web.dev. Learn JavaScript [Електронний ресурс]. – Режим доступу: https://web.dev/learn/javascript/

10. MDN Web Docs. Using Media Queries [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries

11. CanIUse – перевірка підтримки веб-технологій браузерами. URL:

https://caniuse.com/

ДОДАТОК А
Код програми

HTML-розмітка інтерфейсу застосунку (index.html)

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Менеджер справ</title>
  <link rel="stylesheet" href="style.css" />
</head>
<body>
  <div class="app">
    <div class="header">
      <h1>Менеджер справ</h1>
      <button id="themeToggle">🌓</button>
    </div>

    <div class="input-group">
      <input type="text" id="taskInput" placeholder="Нова справа..." />
      <input type="date" id="taskDate" />
      <select id="taskCategory">
        <option value="Загальні">Загальні</option>
        <option value="Робота">Робота</option>
        <option value="Навчання">Навчання</option>
        <option value="Особисте">Особисте</option>
      </select>
      <button onclick="addTask()">Додати</button>
    </div>
```

```
    <div class="filters">
      <select id="filterStatus" onchange="renderTasks()">
        <option value="all">Усі</option>
        <option value="active">Незавершені</option>
        <option value="done">Завершені</option>
      </select>

      <select id="sortBy" onchange="renderTasks()">
        <option value="none">Сортувати...</option>
        <option value="date">За датою</option>
        <option value="alpha">За назвою</option>
        <option value="status">За статусом</option>
      </select>

                <input type="text" id="searchInput" placeholder="Пошук..."
oninput="renderTasks()" />
      <button onclick="clearCompleted()">Очистити завершені</button>
    </div>

    <ul id="taskList"></ul>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

CSS-стилізація інтерфейсу (style.css)

```
body {
  font-family: Arial, sans-serif;
  background: var(--bg);
  color: var(--text);
```

```
  margin: 0;
  padding: 30px;
  display: flex;
  justify-content: center;
}

:root {
  --bg: #f5f5f5;
  --text: #000;
  --card: #fff;
  --highlight: #ffcdd2;
}

.dark-theme {
  --bg: #1e1e1e;
  --text: #eee;
  --card: #2c2c2c;
  --highlight: #ef9a9a;
}

.app {
  background: var(--card);
  padding: 20px;
  border-radius: 15px;
  box-shadow: 0 0 10px rgba(0,0,0,0.2);
  width: 100%;
  max-width: 700px;
}
```

```
.header {
  display: flex;
  justify-content: space-between;
  align-items: center;
}

h1 {
  margin: 0 0 10px 0;
}

#themeToggle {
  background: none;
  border: none;
  font-size: 20px;
  cursor: pointer;
}

.input-group,
.filters {
  display: flex;
  flex-wrap: wrap;
  gap: 10px;
  margin-bottom: 15px;
}

.input-group input,
.input-group select,
.filters select,
.filters input {
```

```
  flex: 1;
  padding: 10px;
  font-size: 14px;
}

.input-group button,
.filters button {
  padding: 10px;
  font-weight: bold;
  border: none;
  border-radius: 5px;
  background-color: #2196f3;
  color: white;
  cursor: pointer;
}

.input-group button:hover,
.filters button:hover {
  background-color: #1976d2;
}

ul {
  list-style: none;
  padding: 0;
}

li {
  background: var(--card);
  padding: 12px;
```

```
  margin-bottom: 10px;
  border-radius: 8px;
  border: 1px solid #ccc;
  display: flex;
  justify-content: space-between;
  flex-direction: column;
}

li.overdue {
  background: var(--highlight);
}

li.done {
  text-decoration: line-through;
  opacity: 0.6;
}

.task-meta {
  font-size: 12px;
  color: gray;
  margin-top: 5px;
}

.task-actions {
  display: flex;
  gap: 10px;
  margin-top: 10px;
}
```

```
.task-actions button {
  padding: 5px 10px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.edit-btn {
  background-color: #ff9800;
  color: white;
}

.delete-btn {
  background-color: #f44336;
  color: white;
}

@media (max-width: 600px) {
  .input-group, .filters {
    flex-direction: column;
  }
}
```

JavaScript-логіка застосунку (script.js)

```
let tasks = JSON.parse(localStorage.getItem("tasks")) || [];

function saveTasks() {
  localStorage.setItem("tasks", JSON.stringify(tasks));
}
```

```
function toggleTheme() {
  document.body.classList.toggle("dark-theme");
}

function renderTasks() {
  const list = document.getElementById("taskList");
  const filter = document.getElementById("filterStatus").value;
  const search = document.getElementById("searchInput").value.toLowerCase();
  const sortBy = document.getElementById("sortBy").value;

  list.innerHTML = "";

  let filtered = tasks.filter(task => {
    const matchStatus =
      filter === "all" ||
      (filter === "done" && task.done) ||
      (filter === "active" && !task.done);
    const matchSearch = task.text.toLowerCase().includes(search);
    return matchStatus && matchSearch;
  });

  if (sortBy === "date") {
    filtered.sort((a, b) => (a.date || "").localeCompare(b.date || ""));
  } else if (sortBy === "alpha") {
    filtered.sort((a, b) => a.text.localeCompare(b.text));
  } else if (sortBy === "status") {
    filtered.sort((a, b) => a.done - b.done);
  }
```

```
  filtered.forEach(task => {
    const li = document.createElement("li");
    if (task.date && new Date(task.date) < new Date() && !task.done) {
      li.classList.add("overdue");
    }

    li.innerHTML = `
      <div class="task-content">
        <strong>${task.text}</strong>
          <div class="task-meta">Категорія: ${task.category} | Дата: ${task.date || "не вказано"}</div>
      </div>
      <div class="task-actions">
        ${task.done ? "<span class='check-mark'>✅</span>" : ""}
        <button class="edit-btn" onclick="editTask(${task.id})">Редагувати</button>
                                                    <button class="delete-btn" onclick="deleteTask(${task.id})">Видалити</button>
        <button onclick="toggleTask(${task.id})">${task.done ? "↩" : "✓"}</button>
      </div>
    `;
    list.appendChild(li);
  });
}

function addTask() {
  const text = document.getElementById("taskInput").value.trim();
  const date = document.getElementById("taskDate").value;
  const category = document.getElementById("taskCategory").value;
  if (text === "") return;
```

```
  const newTask = {
    id: Date.now(),
    text,
    date,
    category,
    done: false
  };

  tasks.push(newTask);
  saveTasks();
  clearInputs();
  renderTasks();
}

function clearInputs() {
  document.getElementById("taskInput").value = "";
  document.getElementById("taskDate").value = "";
}

function toggleTask(id) {
  const task = tasks.find(t => t.id === id);
  if (task) {
    task.done = !task.done;
    saveTasks();
    renderTasks();
  }
}
```

```
function deleteTask(id) {
  tasks = tasks.filter(task => task.id !== id);
  saveTasks();
  renderTasks();
}

function editTask(id) {
  const task = tasks.find(t => t.id === id);
  if (!task) return;

  const newText = prompt("Новий текст:", task.text);
  const newDate = prompt("Нова дата (yyyy-mm-dd):", task.date);

  if (newText !== null) task.text = newText.trim() || task.text;
  if (newDate !== null) task.date = newDate;

  saveTasks();
  renderTasks();
}
function clearCompleted() {
  tasks = tasks.filter(task => !task.done);
  saveTasks();
  renderTasks();
}
document.addEventListener("DOMContentLoaded", () => {
                    document.getElementById("themeToggle").addEventListener("click",
toggleTheme);
  renderTasks();
});
```