КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій
(факультет)

Кафедра кібербезпеки та комп'ютерної інженерії
(назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР

на тему:

Інтелектуальна веб-система для управління персональними завданнями користувача та оцінки ефективності виконання

Овчарука Ігоря Вікторовича
(прізвище, ім'я та по батькові здобувача повністю)

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій

(факультет)

Кафедра кібербезпеки та комп'ютерної інженерії

(назва кафедри)

ЗАТВЕРДЖУЮ

Завідувач кафедри

к.т.н., доцент Максим ДЕЛЕМБОВСЬКИЙ

_____

„___" _____2025 року

КВАЛІФІКАЦІЙНА РОБОТА ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР

Інтелектуальна веб-система для управління персональними завданнями користувача та оцінки ефективності виконання

(назва)

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач Овчарук Ігор Вікторович

(прізвище, ім'я та по батькові повністю)

123 «Комп'ютерна інженерія»

(спеціальність)

Комп'ютерні системи і мережі

(освітня програма)

Група КСМм-24

Керівник Вишняков В.М.

(прізвище та ініціали)

Кандидат технічних наук, доцент

(вчене звання, науковий ступінь)

Рецензент_____

(прізвище та ініціали)

Ідентичність підтверджую

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій
Кафедра: кібербезпеки та комп'ютерна інженерія
Освітній рівень: магістр
Спеціальність: 123 Комп'ютерна Інженерія
ОПП: Комп'ютерні системи і мережі

ЗАТВЕРДЖУЮ
Завідувач кафедри
к.т.н., доцент Максим Делембовський
„___"_____2025 року

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР

Овчарука Ігоря Вікторовича
(прізвище, ім'я та по батькові здобувача)

1. Тема роботи: Інтелектуальна веб-система для управління персональними завданнями користувача та оцінки ефективності виконання затверджена наказом ректора КНУБА № 1636/36/23.2/25 від «30» вересня 2025 року

2. Керівник роботи
Вишняков В.М. кандидат технічних наук, доцент
(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)

3. Термін подання здобувачем роботи до захисту_____

4. Зміст пояснювальної записки за розділами:

Р. 1. Аналіз проблеми та теоретиччні основи створення інтелектуальної веб-системи

Р. 2. Проєктування програмного забезпечення веб-системи управління завданнями.

Р. 3. Розробка інтелектуальної системи управління персональними завданнями та оцінки ефективності.

5. Графічний матеріал за розділами:

Р. 1. 2 рисунки

Р. 2. 3 рисунки

Р. 3._____

6. Консультанти розділів кваліфікаційної випускної роботи

Розділи	Прізвища, ініціали та посади консультанта	Перевірив	
		дата	підпис
Розділ 1.	Вишняков В.М., к.т.н, доцент		
Розділ 2.	Вишняков В.М., к.т.н, доцент		
Розділ 3.	Вишняков В.М., к.т.н, доцент		

7. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Розділ 1. Аналіз проблеми та теоретиччні основи створення інтелектуальної веб-системи.	Вересень 2025 р.
Розділ 2. Проєктування програмного забезпечення веб-системи управління завданнями.	Жовтень 2025 р.
Розділ 3. Розробка інтелектуальної системи управління персональними завданнями та оцінки ефективності.	Жовтень 2025 р.
Остаточне оформлення роботи	Листопад 2025 р.
Направлення роботи на рецензування, перевірку на плагіат	Грудень 2025 р.
Попередній захист роботи на кафедрі	Грудень 2025 р.

8. Дата видачі завдання 30 вересня 2025 року

Керівник _____ _____
(підпис) (прізвище та ініціали)

Здобувач _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Овчарук І.В. «Інтелектуальна веб-система для управління персональними завданнями користувача та оцінки ефективності виконання».

Тема магістерської роботи присвячена вирішенню актуальної проблеми підвищення особистої продуктивності та самоорганізації шляхом розробки спеціалізованого веб-застосунку. Система забезпечує автоматизацію повного циклу управління завданнями (створення, редагування, контроль термінів), реалізує інтелектуальний алгоритм розрахунку KPI-показників на основі пріоритетів і дедлайнів, а також надає інструменти для візуалізації прогресу через інтерактивний дашборд. У роботі проведено порівняльний аналіз існуючих task-менеджерів, досліджено методи оцінки ефективності персоналу та обґрунтовано вибір архітектури Single Page Application (SPA). Виконано проєктування документо-орієнтованої моделі даних (LocalStorage), розроблено адаптивний графічний інтерфейс за принципами Material Design та реалізовано рольову модель доступу (керівник/працівник). Описано програмну реалізацію логічного ядра мовою JavaScript, наведено результати функціонального тестування та підтверджено зручність використання системи на мобільних пристроях.

Ключові слова: веб-система, управління завданнями, KPI, SPA, JavaScript, оцінка ефективності, тайм-менеджмент, адаптивний інтерфейс.

SUMMARY

Ovcharuk I.V. " Intelligent Web-Based System for Managing Personal Tasks and Evaluating Performance."

The master's thesis topic is dedicated to solving the urgent problem of increasing personal productivity and self-organization by developing a specialized web application. The system ensures automation of the full task management cycle, implements an intelligent algorithm for calculating KPI indicators based on priorities and deadlines, and provides tools for progress visualization through an interactive dashboard. The paper conducts a comparative analysis of existing solutions, justifies the choice of Single Page Application (SPA) architecture. The design of a document-oriented data model (LocalStorage) was performed, an adaptive user interface based on Material Design principles was developed, and a role-based access model was implemented. The practical implementation of the logical core in JavaScript is described, along with the results of functional testing, confirming the system's usability on mobile devices.

Keywords: web system, task management, KPI, SPA, JavaScript, performance evaluation, time management, adaptive interface.

РЕЗЮМЕ (SUMMARY) *до кваліфікаційної випускової роботи здобувача*	ПІБ Овчарук Ігор Вікторович Ovcharuk Ihor Viktorovuch		
ЗВО	Київський національний університет будівництва і архітектури		
Тема *(українською та англійською)*	Інтелектуальна веб-система для управління персональними завданнями користувача та оцінки ефективності виконання Intelligent Web-Based System for Managing Personal Tasks and Evaluating Performance		
Освітній ступінь	Магістр		
Факультет	Автоматизації і інформаційних технологій		
Випускова кафедра	Кібербезпеки та комп'ютерної інженерія		
Спеціальність	Комп'ютерна інженерія		
Освітня програма	Комп'ютерні системи і мережі		
Керівник	Вишняков Володимир Михайлович		
Обсяг роботи:	*Поснювальна записка, стор.*	*Розділів*	*Презентація, кількість слайдів*
	122	3	11
Розділ 1	Аналіз проблеми та теоретиччні основи створення інтелектуальної веб-системи		
Розділ 2	Проєктування програмного забезпечення веб-системи управління завданнями.		
Розділ 3	Розробка інтелектуальної системи управління персональними завданнями та оцінки ефективності.		
Висновки по роботі	Розроблено та програмно реалізовано інтелектуальну веб-систему, ключовою відмінністю якої є впровадження унікального алгоритму комплексної оцінки ефективності на основі аналізу пріоритетності та дотримання дедлайнів. Практичним результатом дослідження став готовий до використання, протестований веб-ресурс із модулем візуальної		

	аналітики, що надає користувачам дієвий інструментарій для самоконтролю, оптимізації робочих процесів та підвищення особистої продуктивності.
Ключові слова: Keywords:	веб-система, управління завданнями, KPI, SPA, JavaScript, оцінка ефективності, тайм-менеджмент, адаптивний інтерфейс. web system, task management, KPI, SPA, JavaScript, performance evaluation, time management, adaptive interface.

Здобувач _____/ _____

Керівник _____/ _____
Efficienc

ЗМІСТ

ВСТУП

У сучасних умовах стрімкого розвитку цифрових технологій ефективне планування часу та управління особистими завданнями стає необхідною складовою продуктивної діяльності людини. Зростання обсягів інформації, інтенсивність робочих і навчальних процесів, а також потреба в систематичному контролі власної ефективності спонукають користувачів шукати інструменти, які дозволяють оптимізувати виконання щоденних задач. Традиційні методи планування — паперові щоденники, статичні календарі чи прості To-Do списки — часто виявляються недостатніми, оскільки не враховують індивідуальні особливості користувача, не забезпечують аналітики та не дозволяють адаптувати план під зміну умов.

У цьому контексті особливої актуальності набувають веб-системи, які поєднують можливості управління завданнями з інструментами аналітики та інтелектуальної підтримки прийняття рішень. Інтелектуальні алгоритми дають змогу не лише фіксувати перелік задач, а й оцінювати ефективність їх виконання, аналізувати навантаження, виявляти закономірності, надавати рекомендації та формувати персоналізовані прогнози.

Розроблення веб-системи, що включає адаптивний модуль оцінювання ефективності, є актуальним завданням, оскільки поєднує елементи персонального планування, управління даними, аналітики й інтелектуальних технологій. Створення подібного інструменту дозволить користувачу отримати цілісне бачення власної діяльності, оцінити прогрес, визначити сильні та слабкі сторони, а також побудувати індивідуальну траєкторію розвитку.

Саме тому метою даної дипломної роботи є розробка інтелектуальної веб-системи для управління персональними завданнями та оцінювання ефективності їх виконання, яка забезпечує структурування задач, автоматизований збір статистики, формування аналітичних показників та підтримку користувача в підвищенні продуктивності.

1 АНАЛІЗ ПРОБЛЕМИ ТА ТЕОРЕТИЧНІ ОСНОВИ СТВОРЕННЯ ІНТЕЛЕКТУАЛЬНОЇ ВЕБ-СИСТЕМИ

1.1 Аналіз проблеми

Управління персональними завданнями є однією з ключових складових організації діяльності користувача у професійній, навчальній та повсякденній сферах. Зі зростанням обсягів інформації та збільшенням кількості паралельних задач виникає необхідність у інструментах, що дозволяють систематизувати, пріоритезувати та контролювати виконання завдань, забезпечуючи при цьому можливість оцінювання ефективності. Сучасний ринок програмних рішень пропонує велику кількість платформ, орієнтованих на персональне та командне планування. Проте більшість із них залишаються на рівні класичних планерів, не застосовуючи інтелектуальні методи для аналізу поведінки користувача або прогнозування його продуктивності.

Серед найбільш поширених інструментів слід виділити Todoist, Trello, Asana, Notion, Google Tasks, Microsoft To Do. Усі ці системи надають механізми створення задач, встановлення термінів, позначення пріоритетів, ведення проєктів та синхронізації між пристроями. Попри це, їх функціональність переважно зосереджена на організації записів, а не на глибокому аналізі даних або інтелектуальному керуванні продуктивністю. Наприклад, Todoist має систему балів Karma, що фіксує активність користувача, однак вона не аналізує якість виконання задач, їх складність, часові витрати та не формує рекомендацій. Trello та Asana орієнтовані здебільшого на командну роботу, а їх аналітичні інструменти прив’язані до корпоративних процесів, а не до персональної ефективності. Notion надає свободу у створенні структур, але більшість аналітичних можливостей створюється вручну за допомогою баз даних і формул.

Для більш глибокого розуміння ринкової ніші доцільно детальніше розглянути функціональні особливості та обмеження провідних рішень.

Todoist є одним із найпопулярніших персональних менеджерів завдань. Його ключовою перевагою є використання обробки природної мови (NLP) для швидкого

введення завдань (наприклад, введення "Купити молоко завтра о 10 ранку" автоматично встановлює дату і час). Система гейміфікації "Todoist Karma" візуалізує прогрес користувача за допомогою графіків, проте цей підхід має суттєвий недолік: він базується виключно на кількісних показниках (кількість закритих задач) і не враховує їх складність або пріоритетність. Виконання десяти дрібних побутових справ дає стільки ж балів, скільки реалізація складного етапу проєкту, що спотворює реальну картину продуктивності.

Trello використовує методологію Kanban, що робить його ідеальним для візуалізації потоку виконання завдань (Workflow). Карткова система дозволяє легко переміщувати задачі між станами "To Do", "Doing", "Done". Проте функціональні обмеження Trello у контексті персонального планування для календарного планування без використання платних розширень (Power-Ups). Крім того, у базовій версії відсутні інструменти для аналізу ефективності: користувач не бачить, скільки часу витрачено на задачу і чи дотримано дедлайнів, якщо не налаштує складну систему автоматизації.

Asana та Jira є потужними інструментами для командної роботи. Вони пропонують діаграми Ганта, звіти про завантаженість (Workload) та трекінг часу. Однак для індивідуального користувача ці системи є надлишковими ("over-engineered"). Інтерфейс перевантажений функціями, орієнтованими на корпоративний менеджмент, що збільшує час на внесення та адміністрування простого персонального завдання. До того ж, їхні алгоритми пріоритезації розраховані на проектні команди, а не на динаміку особистої продуктивності.

Існуючі дослідження показують, що користувачі стикаються зі значними труднощами у підтриманні довготривалої мотивації, правильному розподіленні часу, оцінюванні власної продуктивності та адаптації плану до змін умов. Це підтверджує актуальність застосування інтелектуальних технологій – рекомендаційних алгоритмів, аналізу історії виконання завдань, побудови індивідуальних моделей продуктивності. У зарубіжних публікаціях також простежується тренд на створення «smart productivity tools», які повинні не лише

зберігати інформацію, а й розуміти контекст виконання задач. Такими контекстами можуть бути час доби, тип діяльності, рівень складності, попередні затримки, частота прокрастинації, непередбачені переривання тощо. Проте більшість доступних інструментів або реалізують лише часткові елементи цього підходу, або залишаються експериментальними дослідженнями без переходу в реальні продукти.

Таким чином, аналіз сучасного стану предметної області показує, що існує суттєвий розрив між потребами користувачів та можливостями існуючих систем управління завданнями. Актуальним стає створення веб-системи, яка поєднує структурованість планера, автоматизоване збирання статистики, інтелектуальний аналіз поведінки користувача та формування рекомендацій щодо підвищення ефективності.

Узагальнюючи результати аналізу сучасних інструментів управління завданнями та проєктною діяльністю, доцільно порівняти їх за ключовими характеристиками, що визначають можливості користувача у плануванні, пріоритезації та оцінюванні ефективності. Такий аналіз дозволяє визначити сильні та слабкі сторони популярних рішень та окреслити функціональні обмеження, які існують у контексті персональної продуктивності. Порівняльні відомості наведено в таблиці 1.1.

Таблиця 1.1 — Порівняльний аналіз підходів до оцінки ефективності в системах управління завданнями

Характеристика порівняння	**Task Managers (Todoist, TickTick)**	**Project Management (Jira, Asana)**	**Проектована система**
Підхід до пріоритезації	**Ручний** (користувач сам ставить мітку "High/Low")	**Директивний** (менеджер призначає пріоритет виконавцю)	**Алгоритмічний** (розрахунок на основі ваги, дедлайну та складності)

Продовження таблиці 1.1

Метод оцінки ефективності	Кількісний (кількість закритих задач)	Часовий (витрачені години vs план)	**Комплексний KPI** (інтегральна оцінка якості планування)
Зворотний зв'язок	Відсутній або гейміфікація (бали "Карми")	Звіти для керівництва (Burn-down charts)	**Аналітичний дашборд** для самоконтролю та корекції звичок
Адаптивність	Низька (статичні списки)	Середня (залежить від налаштувань Workflow)	**Висока** (авто-сортування списку при зміні умов)
Призначення	Оперативне планування	Контроль виконання проектів	**Стратегічне підвищення персональної продуктивності**

Проведений аналіз показує, що жодна з розглянутих систем не забезпечує комплексного підходу до оцінювання персональної ефективності, оскільки існуючі рішення сфокусовані переважно або на оперативному плануванні, або на командному проєктному менеджменті. У більшості випадків відсутні інструменти інтегрального аналізу, автоматичного формування показників продуктивності та адаптивного коригування робочого навантаження користувача. Це підтверджує необхідність створення інтелектуальної веб-системи, здатної поєднати управління завданнями з аналітичними механізмами підтримки продуктивності, що і визначає актуальність подальшого дослідження та розробки.

1.1.1 Огляд патентної ситуації

У рамках аналізу предметної області було проведено пошук патентних документів, що описують методи та системи автоматизованого управління

завданнями та пріоритезації. Пошук показав, що індустрія активно рухається в бік інтелектуалізації процесів планування.

Зокрема, патент US20220391803A1 "Method and system for using artificial intelligence for task management" описує підхід, де система машинного навчання генерує план виконання завдання на основі його текстового опису та історичних даних користувача. Ключовою відмінністю цього винаходу є спроба прогнозувати необхідні навички та час для виконання задачі. Однак, описана система фокусується на корпоративному середовищі і розподілі задач між працівниками, а не на оптимізації персонального розкладу.

Іншим важливим документом є патент US6961720B1 "System and method for automatic task prioritization". У ньому пропонується використання "двигуна прийняття рішень" (decision engine), який автоматично присвоює коди пріоритету вхідним завданням. Система навчається на діях користувача: якщо користувач часто обирає певний тип задач першими, система підвищує їх пріоритет у майбутньому. Цей підхід є релевантним для нашої розробки, проте він не враховує фактор дедлайнів та штрафів за прострочення, фокусуючись лише на черзі виконання.

Також варто відзначити патент US20120180060A1 "Prediction based priority scheduling", який розглядає алгоритми прогнозування навантаження для планування запитів. Хоча цей патент стосується більше обчислювальних систем, описана в ньому математична модель мінімізації "перешкод" (interference) при плануванні задач з різним пріоритетом може бути адаптована для управління людськими ресурсами часу.

Проведений патентний аналіз підтверджує, що існуючі рішення переважно вирішують задачу сортування списків або розподілу ресурсів у команді. Задача створення персонального адаптивного асистента, який би комплексно оцінював ефективність користувача за системою KPI, залишається відкритою для дослідження.

1.2 Типові проблеми

Попри велику кількість наявних систем управління завданнями, користувачі часто стикаються з повторюваними труднощами, які безпосередньо впливають на продуктивність, мотивацію та якість виконання задач. Однією з ключових проблем є відсутність структурованого підходу до планування. Користувачі заносять у планер велику кількість завдань, проте не формують зрозумілих пріоритетів, не розподіляють задачі за складністю, тривалістю чи рівнем важливості. У результаті списки завдань перетворюються на хаотичні переліки, які складно інтегрувати у реальний робочий ритм, що провокує зниження мотивації та небажання повертатися до інструменту.

Ще однією поширеною проблемою є відсутність зворотного зв'язку щодо власної продуктивності. Більшість систем дають змогу лише відмічати виконання задач, але не аналізують динаміку, не відстежують закономірності, не порівнюють обсяг виконаної роботи з попередніми періодами. Через це користувач не розуміє, чи працює він ефективно, у який час дня продуктивність вища, які типи задач виконуються швидше, а які викликають системні затримки. Низький рівень аналітики у наявних інструментах призводить до того, що людина не може об'єктивно оцінити свої сильні та слабкі сторони, а також не отримує рекомендацій щодо поліпшення результативності.

Важливою проблемою залишається й складність підтримання довготривалої мотивації. За відсутності інтелектуальних алгоритмів, що адаптують план до можливостей та навичок користувача, виникає ризик накопичення прострочених завдань та формування "ефекту заваленості". Користувач бачить великий список невиконаних пунктів, що знижує мотивацію та формує відчуття неефективності, хоча проблема може полягати не в користувачеві, а у неправильно побудованому плануванні. Окремої уваги потребує проблема некоректної оцінки часу. Користувачі часто неправильно прогнозують тривалість виконання задач, що призводить до постійних зривів дедлайнів, перенесень та накопичення стресу. Більшість існуючих систем не аналізує історичні дані про виконання конкретних типів задач і не

пропонує користувачеві реалістичних прогнозів часу, ґрунтованих на його власних попередніх діях. Крім того, сучасні системи мало враховують поведінкові фактори. Наприклад, не фіксуються зміни темпу роботи, періоди перевтоми, частота відволікань, індивідуальні цикли активності. Відсутність адаптивності робить планер “жорстким”, однаковим для всіх користувачів, що не відповідає реальним потребам людей, які мають різні робочі стилі, біоритми та когнітивні особливості. Таким чином, типові проблеми користувачів у сучасних системах управління завданнями зводяться до відсутності інтелектуального аналізу, недостатньої адаптивності, низького рівня аналітичної підтримки, невміння систем підлаштовуватися під реальні поведінкові моделі людини та відсутності персоналізованих рекомендацій. Це підкреслює необхідність створення веб-системи нового покоління, яка поєднає традиційні інструменти планування з інтелектуальними механізмами аналізу та оцінювання ефективності.

Наступною поширеною проблемою є суб’єктивність оцінки власної ефективності. Користувачі, як правило, оцінюють свою продуктивність за кількістю виконаних завдань, не враховуючи їх складність, пріоритетність та дотримання встановлених термінів. Такий підхід не дозволяє отримати достовірні кількісні показники результативності та унеможливлює довгостроковий аналіз динаміки продуктивності.

Важливою проблемою є також ігнорування фактору дедлайнів у процесі оцінки результатів діяльності. У багатьох системах управління завданнями прострочене виконання не має суттєвих наслідків для загальної статистики, що призводить до викривлення реальної оцінки ефективності. Виконання завдання після завершення терміну фактично прирівнюється до своєчасного виконання, що знижує мотивацію користувача дотримуватися планів.

Окрему групу складають технологічні проблеми, пов’язані з обмеженим аналітичним функціоналом більшості існуючих рішень. Багато популярних task-менеджерів орієнтовані виключно на збереження та візуалізацію списків завдань, не надаючи інструментів для формування узагальнених показників ефективності.

Відсутність вбудованих аналітичних механізмів змушує користувачів здійснювати аналіз вручну або взагалі відмовлятися від нього.

Значний вплив має і високе когнітивне навантаження, що виникає при роботі з перевантаженими інтерфейсами. Надмірна кількість елементів управління, складна навігація та відсутність чіткої структури інформації ускладнюють сприйняття даних і знижують зручність використання системи, особливо для користувачів без технічної підготовки.

Для наочності виявлені проблеми, описані у цьому підрозділі, було узагальнено у вигляді структурної схеми. На рисунку 1.1 показано логічні взаємозв’язки між основними труднощами користувачів, що виникають під час використання традиційних методів планування завдань. Схема демонструє, як відсутність аналітики, інтелектуальної підтримки та ефективної організації призводить до хаосу у завданнях, помилок у пріоритезації, зривів дедлайнів і перевантаження.

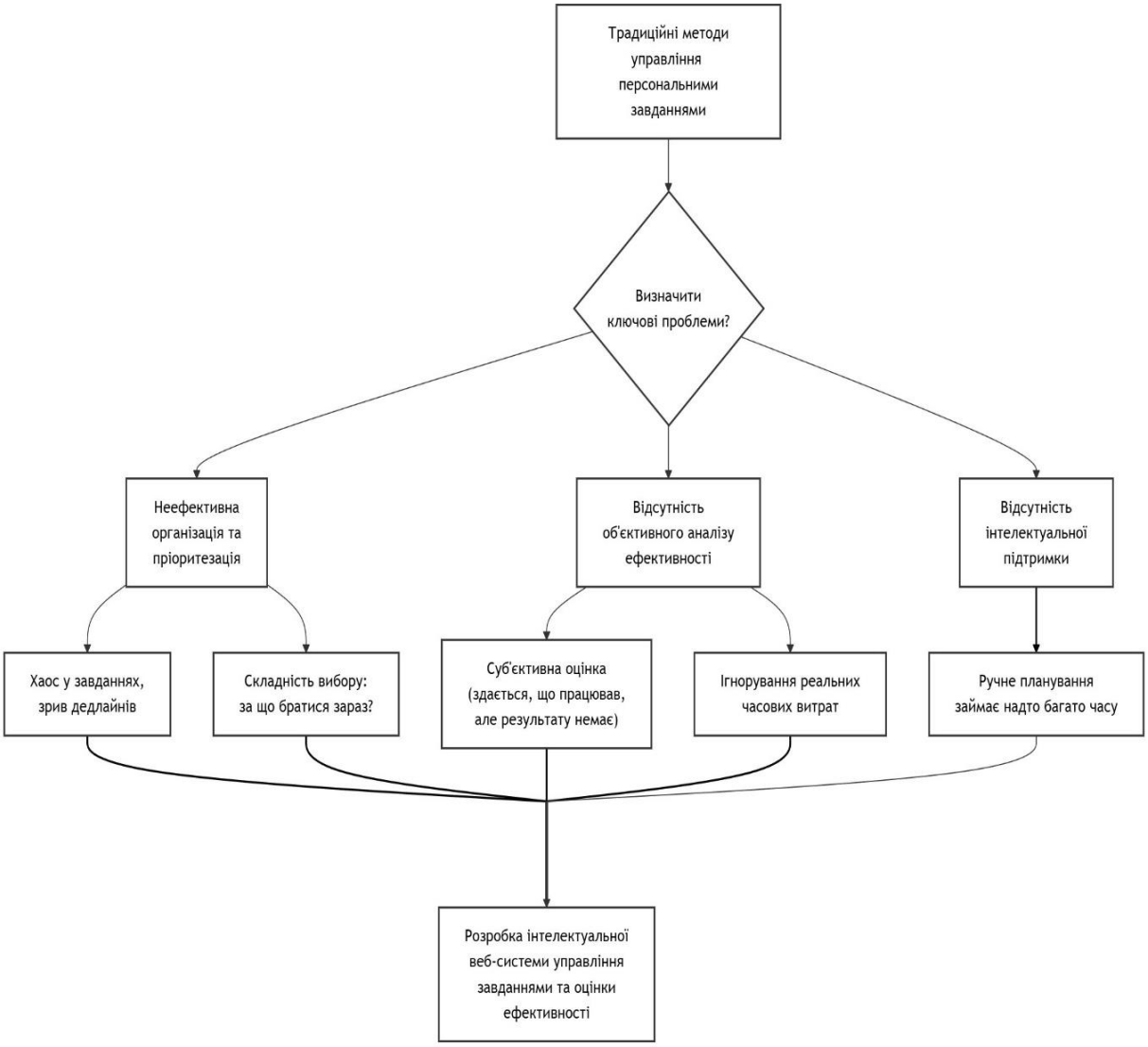

Рисунок 1.1 – Структурна схема ключових проблем у традиційних методах управління персональними завданнями

Подані на рисунку 1.1 взаємозв’язки підтверджують, що проблема управління завданнями має комплексний характер. Таким чином, аналіз типових проблем управління персональними завданнями свідчить про необхідність створення інтелектуальної веб-системи, яка поєднувала б зручні механізми планування з аналітичними інструментами оцінки ефективності. Усунення зазначених недоліків є передумовою підвищення продуктивності користувачів та формування обґрунтованих управлінських рішень у процесі персональної діяльності.

1.3 Огляд сучасних методологій тайм-менеджменту як теоретична основа проєктування системи

Розробка ефективної інформаційної системи управління завданнями неможлива без глибокого розуміння теоретичних основ організації часу. Програмне забезпечення виступає лише інструментом, який автоматизує певний алгоритм діяльності людини. Тому для проєктування логіки роботи веб-системи, зокрема її модулів пріоритезації та оцінки ефективності, було проаналізовано ключові світові методології тайм-менеджменту: Матриця Ейзенхауера, Getting Things Done (GTD) та техніка Pomodoro.

1.3.1 Матриця Ейзенхауера та її алгоритмізація

Однією з найефективніших моделей пріоритезації завдань є матриця Ейзенхауера, яка базується на розподілі задач за двома критеріями: «Важливість» та «Терміновість». Ця методика дозволяє уникнути поширеної когнітивної помилки, коли користувач фокусується на термінових, але неважливих справах, ігноруючи стратегічні цілі.

У контексті проєктування веб-системи, чотири квадранти матриці Ейзенхауера трансформуються у відповідні атрибути програмних об'єктів «Завдання»:

- Квадрант I (Важливі та Термінові): Це критичні завдання, невиконання яких призводить до негативних наслідків (штрафи, зрив проєкту, проблеми зі здоров'ям).

  Програмна реалізація: У системі таким завданням присвоюється пріоритет High (Високий). Алгоритм розрахунку KPI (Key Performance Indicators) надає їм найбільшу вагу (коефіцієнт 3.0), а їх прострочення призводить до найбільшого зниження рейтингу ефективності.
- Квадрант II (Важливі, але Не термінові): Завдання, пов'язані зі стратегічним плануванням, навчанням, профілактикою. Саме робота в цьому квадранті забезпечує довгострокову ефективність.

Програмна реалізація: Цим завданням відповідає пріоритет Medium (Середній). Система повинна стимулювати користувача виконувати їх до того, як вони стануть терміновими.

- Квадрант III (Не важливі, але Термінові): Рутинні задачі, телефонні дзвінки, пошта. Вони створюють ілюзію зайнятості.

  Програмна реалізація: Пріоритет Low (Низький). В архітектурі системи передбачено рольову модель (Lead/Member), яка дозволяє реалізувати механізм делегування таких завдань іншим виконавцям.
- Квадрант IV (Не важливі і Не термінові): "Пожирачі часу" (соціальні мережі, ігри).

  Програмна реалізація: Такі завдання не повинні потрапляти до системи або мають підлягати видаленню. Функціонал deleteTask, реалізований для адміністратора системи, дозволяє очищувати список від подібного "інформаційного шуму".

1.3.2 Методологія Getting Things Done

Методологія GTD, розроблена Девідом Алленом, базується на принципі звільнення мозку від запам'ятовування поточних справ шляхом перенесення їх на зовнішній носій. Ключовий алгоритм GTD складається з п'яти етапів: збір, обробка, організація, огляд та виконання.

Для проєктованої веб-системи критично важливими є такі аспекти GTD:

- Принцип "Inbox" (Вхідні): Будь-яка нова ідея чи завдання повинні бути миттєво зафіксовані. Це диктує вимоги до інтерфейсу системи — форма створення завдання (#task-creator) має бути максимально простою, доступною з будь-якого екрана і вимагати мінімум обов'язкових полів.
- Контекст та статус: Завдання не є статичними. Вони змінюють свій стан від "Нового" до "В роботі" або "Виконаного". У розробленій системі це реалізовано через машину станів (State Machine) об'єкта Task, який може набувати значень new, done або failed.

- Регулярний огляд: GTD наголошує на важливості аналізу виконаної роботи. Саме тому в систему інтегровано модуль аналітики, який дозволяє користувачеві (або керівнику) переглядати статистику за певний період та коригувати плани.

1.3.3 Техніка Pomodoro та управління фокусом

Проблема прокрастинації та втрати концентрації є однією з ключових у сучасній інформаційній діяльності. Техніка Pomodoro пропонує розбивати роботу на інтервали (зазвичай 25 хвилин) з короткими перервами.

Хоча в поточній ітерації системи таймер не реалізовано, архітектура Single Page Application (SPA) спроєктована таким чином, щоб мінімізувати відволікаючі фактори:

- Інтерфейс розділено на функціональні зони (Dashboard), де користувач бачить лише актуальні на сьогодні завдання.
- Використання "спокійного дизайну" (Calm Technology) у кольоровій гамі зменшує зорове навантаження і сприяє утриманню фокусу.
- Візуалізація прогресу через інтерактивні графіки створює ефект гейміфікації, що підвищує мотивацію до завершення "помодоро-циклів".

1.3.3 Порівняння підходів та вибір концепції системи

Аналіз наведених методологій дозволив сформулювати гібридну концепцію для розроблюваної інтелектуальної системи. Жодна з методологій у чистому вигляді не покриває всіх потреб сучасного користувача: матриця Ейзенхауера добре пріоритезує, але не допомагає з потоком вхідних даних; GTD чудово структурує хаос, але не дає чіткої оцінки ефективності; Pomodoro фокусується на процесі, а не на результаті.

Тому проєктована система (Task Intelligence System) поєднує:

- Структурність GTD: Списки завдань та статусна модель.
- Стратегічність Ейзенхауера: Обов'язкова пріоритезація (Priority Field) та вагові коефіцієнти для розрахунку KPI.

- Адаптивність Agile/Kanban: Можливість швидкої зміни виконавця (assignedTo) та візуалізація дедлайнів на календарі.

Такий синтез підходів дозволяє створити інструмент, який не просто фіксує список справ, а виступає інтелектуальним асистентом, що оцінює якість планування та виконання завдань згідно з науково обґрунтованими критеріями.

1.4 Інтелектуальні та аналітичні технології

Сучасні системи управління персональними завданнями продовжують розвиватися в напрямку застосування інтелектуальних технологій, які дозволяють не лише зберігати та структурувати інформацію, а й здійснювати глибокий аналіз поведінкових патернів користувача. Використання таких підходів дає можливість виявляти закономірності, прогнозувати продуктивність і формувати персоналізовані рекомендації, що значно підвищує ефективність роботи з планером. Однією з ключових технологій, що застосовується в сучасних інтелектуальних системах, є машинне навчання. На основі історичних даних про виконання завдань алгоритми можуть визначати, які типи задач користувач виконує швидше, які потребують більше концентрації, а які найчастіше переносяться. Це дозволяє будувати моделі поведінки, що враховують індивідуальний стиль роботи користувача, його продуктивні години та інтенсивність навантаження. Такі моделі можуть автоматично рекомендувати оптимальний час для виконання конкретних задач, нагадувати про необхідність перерви або запобігати перевантаженню. Іншим важливим напрямом є використання рекомендаційних систем. Вони застосовуються для динамічного перерозподілу задач залежно від актуальних умов, таких як зміни пріоритетів, зовнішні фактори, рівень стресу або темп виконання попередніх завдань. Рекомендаційні алгоритми здатні аналізувати структуру плану та пропонувати користувачеві найбільш оптимальну черговість виконання задач, що допомагає уникнути простоїв та підвищує загальну ефективність.

Аналітичні методи відіграють не менш важливу роль. У сучасних системах продуктивності широко застосовують аналіз часових рядів, що дає змогу визначати

тенденції у змінах продуктивності користувача протягом тривалого періоду. Використання статистичних підходів дозволяє оцінювати середню тривалість виконання задач, частоту перенесень, динаміку завершеності великомасштабних проєктів та інші метрики, що формують базову модель ефективності.

Когнітивні технології, такі як розпізнавання патернів прокрастинації, аналіз поведінкових тригерів чи виявлення непродуктивних часових зон, дозволяють вийти за межі стандартного аналізу виконаних задач. Вони моделюють реальні поведінкові аспекти користувача та допомагають зрозуміти, які фактори впливають на результативність у довгостроковій перспективі. Завдяки цьому система може пропонувати поради щодо зміни робочого ритму, структури планування або чергування видів діяльності. Когнітивні технології, такі як розпізнавання патернів прокрастинації, аналіз поведінкових тригерів чи виявлення непродуктивних часових зон, дозволяють вийти за межі стандартного аналізу виконаних задач. Вони моделюють реальні поведінкові аспекти користувача та допомагають зрозуміти, які фактори впливають на результативність у довгостроковій перспективі. Завдяки цьому система може пропонувати поради щодо зміни робочого ритму, структури планування або чергування видів діяльності.

Особливої уваги потребують технології адаптивного планування. Ці алгоритми автоматично підлаштовують план під зміни навантаження або зовнішні обставини. Наприклад, якщо користувач протягом певного періоду виконує складні задачі повільніше, ніж зазвичай, система може запропонувати зменшити кількість завдань або змінити їхню послідовність. Адаптивність дозволяє уникнути ефекту “накопичення прострочених задач”, який є однією з найпоширеніших причин втрати мотивації у традиційних планерах. Особливої уваги потребують технології адаптивного планування. Ці алгоритми автоматично підлаштовують план під зміни навантаження або зовнішні обставини. Наприклад, якщо користувач протягом певного періоду виконує складні задачі повільніше, ніж зазвичай, система може запропонувати зменшити кількість завдань або змінити їхню послідовність.

Адаптивність дозволяє уникнути ефекту “накопичення прострочених задач”, який є однією з найпоширеніших причин втрати мотивації у традиційних планерах.

У перспективних дослідженнях значну увагу приділяють моделям прогнозування ефективності. Вони дозволяють передбачати, чи зможе користувач завершити певну кількість задач у визначений день, які завдання можуть бути проблемними та на яких етапах існує ризик затримок. Прогностичні алгоритми формують основу для створення справді інтелектуальних інструментів планування, які не тільки відображають поточний стан плану, а й прогнозують майбутні результати. Отже, інтелектуальні та аналітичні технології дозволяють створювати системи управління завданнями нового покоління.

Такі системи можуть глибоко аналізувати поведінку користувача, враховувати продуктивність у динаміці, підлаштовуватися під індивідуальний стиль роботи та надавати рекомендації, спрямовані на підвищення ефективності. Це створює підґрунтя для формування веб-системи, яка не лише фіксує завдання, але й активно підтримує користувача у процесі їх виконання.

Розвиток інтелектуальних та аналітичних технологій у сфері інформаційних систем створив передумови для переходу від простих інструментів планування до більш складних програмних рішень, здатних здійснювати аналіз даних та формувати узагальнені показники ефективності. У контексті управління персональними завданнями такі технології відіграють ключову роль, оскільки дозволяють не лише фіксувати факт виконання дій, а й оцінювати результативність діяльності користувача.

Одним із найбільш поширених підходів є rule-based системи, які базуються на наборі наперед визначених правил і умов. У таких системах логіка прийняття рішень формується на основі чітко заданих залежностей між параметрами завдання, зокрема пріоритетом, терміном виконання та статусом. Перевагою rule-based підходу є його прозорість і передбачуваність, що особливо важливо для систем персонального використання. Користувач має можливість розуміти, яким чином формується оцінка ефективності, що підвищує довіру до результатів аналізу.

Іншим класом інтелектуальних технологій є евристичні методи, які застосовуються для прийняття рішень у ситуаціях невизначеності або за відсутності повної інформації. У системах управління завданнями евристики можуть використовуватися для приблизної оцінки складності завдань, прогнозування ризику порушення дедлайнів або визначення оптимального порядку виконання. Хоча евристичні алгоритми не гарантують оптимального результату, вони дозволяють отримати практично корисні рішення з мінімальними обчислювальними витратами.

Окрему увагу заслуговують методи машинного навчання, які активно застосовуються у сучасних аналітичних системах. Дані методи дозволяють автоматично виявляти приховані закономірності у великих масивах даних та адаптуватися до поведінки користувача. У контексті управління персональними завданнями машинне навчання може використовуватися для прогнозування продуктивності, автоматичної зміни пріоритетів або рекомендацій щодо оптимізації робочого навантаження. Проте реалізація таких підходів потребує значних обсягів навчальних даних і складної інфраструктури, що ускладнює їх використання у легких веб-застосунках персонального призначення.

З огляду на зазначені особливості, для розробки інтелектуальної веб-системи управління персональними завданнями доцільним є використання комбінованого аналітичного підходу, який поєднує rule-based логіку з елементами аналітичної обробки даних. Такий підхід дозволяє формувати кількісні показники ефективності на основі чітко визначених критеріїв, зберігаючи при цьому простоту реалізації та зрозумілість для користувача.

Аналітична складова системи базується на обробці накопичених даних про завдання, зокрема інформації про їх пріоритети, терміни виконання та фактичний стан. На основі цих даних можливе формування ключових показників ефективності (KPI), які відображають рівень продуктивності користувача у певний проміжок часу. Використання KPI дозволяє перейти від суб’єктивної оцінки результатів до об’єктивного аналізу, що є важливою складовою підвищення рівня самоорганізації.

Таким чином, застосування інтелектуальних та аналітичних технологій у системах управління персональними завданнями створює основу для формування адаптивних, інформативних та зручних інструментів підтримки користувача. Обґрунтований вибір аналітичного підходу дозволяє забезпечити баланс між функціональністю системи, складністю реалізації та практичною цінністю отриманих результатів, що є особливо важливим у межах магістерського дослідження.

Сучасні інтелектуальні системи управління завданнями складаються з взаємопов’язаних модулів, що відповідають за планування, аналітику, автоматизацію та системні функції користувача. Узагальнену структуру розроблюваної веб-системи представлено на рисунку 1.2.

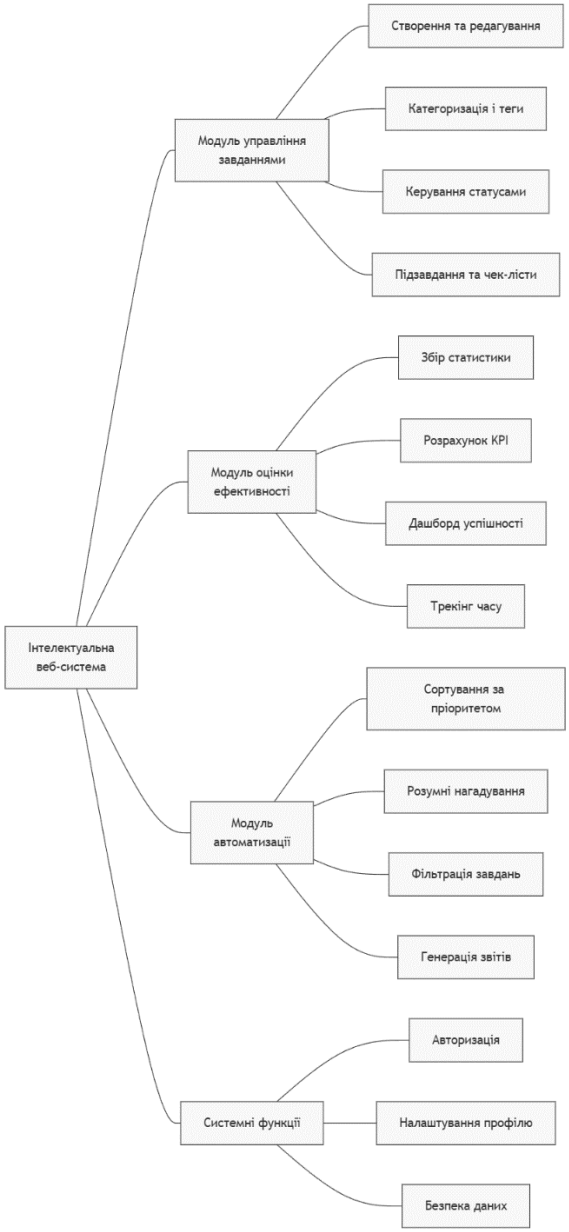

Рис. 1.2 – Структурна схема інтелектуальної веб-системи

1.5 Науково-технічні передумови та актуальність

Стрімкий розвиток інформаційних технологій, зростання цифрових сервісів та розширення можливостей обробки даних створили сприятливі передумови для формування інтелектуальних систем нового покоління. Особливо це стосується інструментів, що забезпечують підтримку прийняття рішень, аналітику поведінки користувача та автоматизацію рутинних процесів. Науково-технічний прогрес у сфері обробки даних, хмарних технологій та машинного навчання суттєво розширив можливості персональних планерів, перетворюючи їх із простих

інструментів організації задач на повноцінні інтелектуальні системи управління діяльністю. Одним із ключових чинників, що визначають актуальність створення інтелектуальної веб-системи управління персональними завданнями, є зміна характеру роботи та навчання.

Розвиток сучасних інформаційних технологій створив сприятливі науково-технічні передумови для впровадження інтелектуальних веб-систем у сфері управління персональними завданнями. Поширення високопродуктивних браузерів, стандартизація мов веб-розробки та зростання обчислювальних можливостей клієнтських пристроїв дозволяють реалізовувати складну логіку обробки даних без використання ресурсоємних серверних рішень.

Важливим фактором є розвиток веб-стандартів HTML5, CSS3 та JavaScript, які забезпечують створення інтерактивних, адаптивних і кросплатформних застосунків. Застосування архітектури Single Page Application (SPA) дозволяє підвищити швидкодію інтерфейсу, зменшити затримки при взаємодії з користувачем та забезпечити безперервний робочий процес без перезавантаження сторінок. Це є особливо важливим для систем управління завданнями, де швидкість доступу до інформації безпосередньо впливає на зручність використання.

Окрему роль відіграє поява та розвиток клієнтських механізмів зберігання даних, зокрема Web Storage API, які дозволяють зберігати структуровану інформацію безпосередньо у браузері користувача. Це створює передумови для розробки автономних веб-систем, здатних працювати без постійного підключення до серверної частини, що знижує складність розгортання та підвищує доступність програмного продукту.

Сучасний користувач щоденно взаємодіє з великою кількістю інформації, виконує різнорідні задачі, працює у високій динаміці та часто перебуває у стані багатозадачності. У таких умовах традиційні планувальні інструменти не можуть забезпечити достатній рівень підтримки, оскільки вони не аналізують продуктивність, не прогнозують навантаження та не враховують контекст виконання завдань. Це формує потребу у системах, які здатні адаптуватися до

індивідуальних особливостей користувача та забезпечувати персоналізовані рекомендації.

Важливою науково-технічною передумовою є розвиток технологій машинного навчання та штучного інтелекту, що дозволяють створювати моделі поведінки користувача, виявляти закономірності в його роботі та прогнозувати майбутні результати. Алгоритми класифікації, регресійні моделі, аналіз часових рядів та методи прогнозування забезпечують технічну основу для формування інтелектуальних модулів продуктивності. Завдяки цьому система може аналізувати накопичені дані, враховувати індивідуальний стиль роботи користувача та формувати рекомендації щодо оптимізації плану.

Додатковою передумовою створення інтелектуальних веб-систем є розвиток клієнтських веб-технологій (HTML5, CSS3, JavaScript) та поява стандартів Web API, які дозволяють реалізовувати складну логіку без встановлення спеціального програмного забезпечення. Саме еволюція браузерних рушіїв і розширення їхніх можливостей (рендеринг інтерфейсу, робота з локальними даними, підтримка адаптивної верстки) зробили веб-додатки повноцінною альтернативою десктопним планерам.

Окремо варто відзначити розвиток підходів до проєктування користувацького досвіду (UX) та інтерфейсних патернів для керування задачами: календарні представлення, візуальні статуси, фільтрація за пріоритетами, швидке редагування та зворотний зв’язок системи через мікровзаємодії. У комплексі це формує основу для побудови інструментів, які не перевантажують користувача, а підтримують його в щоденній роботі.

1.6 Постановка задачі і план реалізації

Для реалізації поставленої задачі необхідно розробити інтелектуальну веб-систему управління персональними завданнями, яка забезпечить структуроване планування діяльності користувача, збір статистичних даних та аналітичне оцінювання ефективності виконання задач. Основною метою є створення адаптивного інструменту, здатного відстежувати динаміку продуктивності,

формувати індивідуальні показники ефективності та підтримувати користувача у прийнятті рішень щодо оптимізації власного робочого навантаження.

Застосування таких систем сприятиме підвищенню організованості, прозорості робочих процесів та покращенню результативності користувача.

Практична реалізація цього проєкту буде складатися з кількох етапів:

1. **Розробка структури бази даних**
   - Збір та зберігання інформації про задачі, категорії, пріоритети, статуси, дедлайни, часові витрати та статистичні показники.
   - Формування зв’язків між об’єктами: задачами, подіями, тегами, історією змін та користувачем.
   - Оптимізація структури для швидкого доступу до даних, побудови аналітики і дашбордів.
2. **Розробка аналітичного модуля оцінки ефективності**
   - Формування алгоритмів розрахунку продуктивності: коефіцієнтів завершення задач, виконання в строк, розподілу навантаження, трендів активності.
   - Оцінка динаміки виконання задач у часових інтервалах: день, тиждень, місяць.
   - Побудова механізмів аналізу відхилень: визначення перевантажених періодів, невиконаних задач, закономірностей у роботі.
   - Формування персональних рекомендацій на основі даних.
3. **Розробка інтерфейсу користувача (фронтенду)**
   - Створення інтуїтивного інтерфейсу для управління задачами: створення, редагування, сортування, фільтрація, пошук, позначки пріоритетності.
   - Візуалізація аналітичних показників у вигляді графіків, діаграм, ліній часу, KPI-карток.
   - Адаптація інтерфейсу під різні пристрої та забезпечення високої юзабіліті.
4. **Розробка серверної частини (бекенду)**

- Обробка запитів користувача, взаємодія з базою даних та забезпечення логіки роботи системи.
- Реалізація механізмів обробки статистики та збереження історії змін задач.
- Створення API для інтеграції фронтенду з бекендом і можливого подальшого розширення системи.

5. Механізми підтримки користувача

- Реалізація системи нагадувань про дедлайни та важливі події.
- Впровадження підказок і мікрорекомендацій для оптимізації планування.
- Формування звітів про виконання задач, динаміку активності та продуктивність за певний період

6. Забезпечення безпеки та стабільності роботи

- Захист даних користувача, шифрування чутливої інформації та контроль авторизації.
- Виявлення помилок та аномалій у роботі системи.
- Оптимізація продуктивності та стабільності веб-застосунку.

Для реалізації веб-системи планується використання сучасних веб-технологій, зокрема:

JavaScript — для динамічної взаємодії з інтерфейсом.

HTML/CSS — для побудови та стилізації сторінок.

Python або PHP — для розробки серверної логіки та роботи з даними.

SQL / NoSQL — для організації збереження інформації про задачі та показники ефективності.

Для досягнення поставленої мети в межах магістерської роботи необхідно вирішити комплекс взаємопов’язаних задач, що охоплюють етапи аналізу, проєктування та практичної реалізації інтелектуальної веб-системи управління персональними завданнями. Формалізація даних задач дозволяє забезпечити

системний підхід до розробки програмного продукту та логічну послідовність виконання дослідження.

На етапі аналітичного дослідження передбачається здійснення огляду існуючих підходів до управління персональними завданнями та методів оцінки ефективності діяльності користувачів. Особлива увага приділяється виявленню недоліків сучасних рішень, зокрема відсутності інтегрованої аналітики, обмеженості механізмів пріоритезації та формалізованих показників результативності.

Наступним етапом є проєктування програмного забезпечення веб-системи, що включає вибір архітектури, визначення структури даних, розробку логіки взаємодії між компонентами та формування ролей користувачів. На цьому етапі також здійснюється обґрунтування вибору технологічних засобів та принципів побудови інтерфейсу користувача з урахуванням вимог зручності та адаптивності.

Етап практичної реалізації передбачає розробку інтелектуального ядра системи, яке забезпечує обробку даних про завдання, розрахунок показників ефективності та формування аналітичних результатів. Окрему увагу приділено реалізації механізмів зберігання даних, обробки подій користувача та візуалізації результатів у вигляді зрозумілих індикаторів і статистичних зведень.

Завершальним етапом виконання роботи є тестування розробленої системи, аналіз отриманих результатів та оцінка відповідності функціональних можливостей поставленим вимогам. Проведення тестування дозволяє підтвердити працездатність веб-системи та доцільність обраних технічних і аналітичних рішень.

Розробка цієї інтелектуальної системи дозволить користувачам підвищити власну продуктивність, сформувати здорові робочі патерни, уникати перевантаження, а також отримувати об’єктивну та наочну оцінку ефективності своєї діяльності.

1.7 Об’єкт, предмет та мета дослідження

Об’єктом дослідження є процес інформаційної підтримки прийняття рішень при управлінні персональними потоками завдань та часовими ресурсами користувача.

Предметом дослідження є методи багатокритеріального оцінювання ефективності, моделі адаптивного планування та алгоритми розрахунку інтегральних показників продуктивності (KPI) у веб-орієнтованих системах.

Метою роботи є підвищення ефективності персонального тайм-менеджменту шляхом розробки інтелектуальної веб-системи, що реалізує автоматизований збір статистики, аналіз патернів поведінки користувача та надання рекомендацій на основі розроблених математичних моделей оцінювання.

Досягнення поставленої мети потребує чіткого визначення меж дослідження та взаємозв’язку між об’єктом і предметом роботи. Об’єкт дослідження відображає загальну сферу, в межах якої здійснюється наукове дослідження, тоді як предмет зосереджує увагу на конкретних процесах, методах і засобах, що підлягають детальному аналізу та вдосконаленню.

У межах даної магістерської роботи об’єкт дослідження охоплює процеси управління персональними завданнями користувача в інформаційних системах, які характеризуються необхідністю планування, контролю та оцінки результатів діяльності. Предмет дослідження, у свою чергу, зосереджений на методах і програмних засобах реалізації інтелектуальної веб-системи, що забезпечує автоматизований аналіз ефективності виконання завдань з урахуванням їх пріоритетності та термінів виконання.

Формульована мета дослідження визначає загальний напрям роботи та зумовлює вибір методів і засобів її досягнення. Реалізація поставленої мети передбачає поєднання теоретичних підходів до аналізу продуктивності з практичною реалізацією програмного продукту, здатного забезпечити кількісну оцінку результативності діяльності користувача. Такий підхід дозволяє не лише

обґрунтувати наукову новизну роботи, а й підтвердити її практичну значущість у контексті сучасних вимог до персональної ефективності та самоорганізації.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- Здійснити аналітичний огляд існуючих підходів до оцінки персональної ефективності (методи GTD, ABC-аналіз, принцип Парето).
- Розробити математичну модель розрахунку рейтингу успішності користувача, що враховує не лише факт виконання, а й складність, пріоритетність та своєчасність завдань.
- Синтезувати алгоритм ранжування черги завдань на основі зважених критеріїв без використання нейромережевих підходів.
- Спроектувати архітектуру веб-системи та реалізувати модуль візуалізації аналітичних даних (дашборд).
- Провести експериментальне дослідження ефективності запропонованих алгоритмів на тестових наборах даних.

2. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВЕБ-СИСТЕМИ УПРАВЛІННЯ ЗАВДАННЯМИ

2.1 Технічна платформа та обґрунтування вибору засобів розробки

Сучасний стан розвитку веб-технологій диктує високі вимоги до інтерактивності, швидкодії та кросплатформенності програмних продуктів. При проєктуванні «Task Intelligence System» — інтелектуальної системи управління завданнями та оцінки ефективності персоналу — було прийнято рішення використовувати архітектуру односторінкового веб-застосунку (Single Page Application, SPA).

Для реалізації інтелектуальної веб-системи важливим етапом є обґрунтований вибір програмних засобів та технологій, які забезпечують необхідний рівень функціональності, продуктивності та зручності використання. Оскільки розроблювана система орієнтована на персональне використання та повинна працювати без складної серверної інфраструктури, доцільним є застосування клієнтських веб-технологій, що підтримуються сучасними браузерами.

Використання веб-платформи як основи для реалізації системи дозволяє забезпечити кросплатформність та доступність застосунку незалежно від операційної системи користувача. На відміну від настільних програм, веб-застосунок не потребує інсталяції та оновлення програмного забезпечення, що значно спрощує процес використання і підвищує зручність взаємодії з системою.

Порівняно з серверно-орієнтованими рішеннями, клієнтська архітектура з використанням браузерних технологій характеризується меншою складністю реалізації та експлуатації. Це є особливо актуальним для інтелектуальних систем персонального призначення, у яких основний обсяг обробки даних може виконуватися локально без необхідності постійної взаємодії з сервером. Такий підхід також зменшує ризики, пов’язані з безпекою передачі персональних даних.

Обрані програмні засоби дозволяють реалізувати інтерактивний інтерфейс користувача, динамічну обробку подій та аналічну логіку обчислення показників

ефективності. Це створює основу для впровадження інтелектуальної складової системи, яка полягає в автоматизованому аналізі даних про завдання та формуванні узагальнених результатів діяльності користувача.

Цей підхід дозволяє завантажувати необхідний код (HTML, CSS, JavaScript) одноразово, а подальша взаємодія з користувачем відбувається шляхом динамічного оновлення контенту без повного перезавантаження сторінки, що значно підвищує комфорт використання та швидкість відгуку системи. Для реалізації клієнтської частини системи було обрано класичний стек веб-технологій: HTML5, CSS3 та JavaScript (ES6+). Відмова від використання важковагових фреймворків (таких як Angular або React) на користь чистого JavaScript («Vanilla JS») обумовлена необхідністю забезпечення максимальної продуктивності, мінімізації залежностей та спрощення процедури розгортання системи.

2.1.1 Мова розмітки гіпертексту HTML5

В якості основи структурної розмітки системи використано стандарт HTML5. Він надає широкий набір семантичних тегів, які дозволяють чітко структурувати контент, покращуючи читабельність коду та доступність інтерфейсу (Accessibility). У розробленій системі структура головного файлу index.html базується на використанні блочних контейнерів, що розділяють інтерфейс на логічні зони: панель авторизації (login-container), основний робочий простір (app) та секційні блоки (section) для календаря, списку завдань та статистики. Використання HTML5 також дозволило імплементувати нативні елементи управління формами, такі як <input type="date"> для вибору дедлайну завдання, що забезпечує уніфікований інтерфейс календаря залежно від операційної системи користувача без необхідності підключення сторонніх бібліотек.

2.1.2 Каскадні таблиці стилів CSS3

Для візуального оформлення системи обрано стандарт CSS3, який дозволяє створювати складні анімації, градієнти та адаптивні макети. Ключовою вимогою до системи є її адаптивність — коректне відображення як на десктопних моніторах,

так і на мобільних пристроях. Для реалізації цієї вимоги використано технологію Media Queries. Зокрема, для екранів шириною менше 900 пікселів застосовано правило @media (max-width: 900px), яке трансформує сітку макета (.main-layout) з горизонтального відображення колонок у вертикальне, забезпечуючи зручність читання та навігації на смартфонах. Для побудови сітки інтерфейсу використано сучасні модулі верстки:

- Flexbox: Застосовано для вирівнювання елементів всередині карток завдань, кнопок та форм (класи .task-row, .task-actions). Це дозволяє динамічно розподіляти простір між елементами.
- CSS Grid: Використано для побудови складного макета календаря (.calendar-days), де дні місяця автоматично розташовуються у вигляді таблиці 7xN.

Особливу увагу приділено візуальній естетиці та інтерактивності. Використано CSS-анімації (@keyframes), наприклад, анімація fadeIn для плавної появи нових завдань та модальних вікон, а також ефекти hover для кнопок і карток, що забезпечує інтуїтивно зрозумілий зворотний зв'язок інтерфейсу на дії користувача.

2.1.3 Мова програмування JavaScript

Логічне ядро клієнтської частини системи реалізовано мовою програмування JavaScript. Вибір цієї мови обумовлений її домінуючим становищем у браузерному середовищі, що гарантує крос браузерну сумісність та коректне відображення інтерфейсу на всіх сучасних пристроях без необхідності встановлення додаткових плагінів. Архітектура скриптів побудована таким чином, щоб забезпечити миттєву реакцію системи на дії користувача (створення завдання, зміна статусу), мінімізуючи затримки при обробці даних.

У програмному коді проєкту активно використовуються можливості сучасного стандарту ECMAScript 6+ (ES6+), що дозволило зробити код більш структурованим, читабельним та безпечним. Основні застосовані механізми включають:

Стрілкові функції (Arrow Functions): Використовуються для лаконічного опису обробників подій та callback-функцій у методах перебору масивів. Це дозволило спростити синтаксис при роботі з колекціями даних, наприклад, при фільтрації списку завдань або обчисленні KPI (tasks.filter(t => t.status === 'done')).
Шаблонні рядки (Template Literals): Застосовуються для динамічної генерації HTML-розмітки (рендерингу) карток завдань. Використання інтерполяції змінних ${...} дозволяє інтегрувати дані про назву, дедлайн та пріоритет завдання безпосередньо в структуру DOM-елементів, уникаючи громіздкої конкатенації рядків, характерної для застарілих версій JS.

Блочна область видимості (let та const): Замість застарілого var використовуються змінні з блочною видимістю, що запобігає конфліктам імен змінних та підвищує надійність алгоритмів обробки даних, особливо в циклах та умовних конструкціях.

Деструктуризація об'єктів: Цей механізм дозволяє зручно витягувати окремі властивості з об'єктів завдань (наприклад, id, title, priority) для їх подальшої передачі у функції рендерингу або редагування.

Окрему увагу в логічному ядрі приділено роботі з даними та їх персистентності. Для збереження стану системи між сесіями використовується механізм LocalStorage, взаємодія з яким реалізована через вбудований об'єкт JSON:

Метод JSON.stringify() використовується для серіалізації масиву об'єктів-завдань перед записом у сховище.

Метод JSON.parse() забезпечує зворотне перетворення збереженого рядка в повноцінні JavaScript-об'єкти при завантаженні сторінки.

Також реалізовано модуль валідації вхідних даних на стороні клієнта. Перед створенням нового завдання скрипт перевіряє заповненість обов'язкових полів та коректність введених дат, що запобігає потраплянню помилкових даних у систему розрахунку ефективності. Логіка оновлення інтерфейсу побудована на принципах реактивності: будь-яка зміна в даних (наприклад, завершення завдання)

автоматично ініціює перерахунок статистики та оновлення відповідних елементів на дашборді без перезавантаження сторінки.

2.1.4 Технологія зберігання даних Web Storage API

Однією з ключових особливостей спроєктованої системи є відсутність серверної бази даних (такої як MySQL або PostgreSQL) на початковому етапі, що дозволяє системі працювати в автономному режимі. Замість цього використано механізм LocalStorage. LocalStorage — це властивість об'єкта window, що дозволяє зберігати пари ключ-значення у веб-браузері. Основні переваги вибору цієї технології для даної системи:

- Персистентність: Дані зберігаються навіть після закриття браузера або перезавантаження комп'ютера, що є критичним для системи управління завданнями.
- Швидкодія: Доступ до даних відбувається миттєво без мережевих затримок, характерних для запитів до віддаленого сервера.
- Простота структури: Дані зберігаються у вигляді JSON-рядків, що ідеально підходить для документо-орієнтованої моделі даних системи (колекції users та tasks). У системі використовуються два основні ключі сховища:
- **users** — для зберігання масиву об'єктів користувачів з їхніми ролями.
- **tasks** — для зберігання масиву завдань з усіма метаданими (заголовок, пріоритет, дедлайн, статус).

2.1.5 Інструментальні засоби розробки

В якості основного інструментального засобу для написання коду та управління файловою структурою проєкту було обрано інтегроване середовище розробки (IDE) Visual Studio Code (VS Code) від компанії Microsoft. Вибір саме цього середовища обґрунтований його кросплатформеністю, високою швидкодією та наявністю розгалуженої екосистеми розширень, що перетворює його на повноцінну робочу станцію для веб-розробки.

Ключовою перевагою VS Code у контексті розробки даної системи стала технологія IntelliSense, яка забезпечує інтелектуальне автодоповнення коду, підсвічування синтаксису JavaScript та автоматичне визначення типів даних. Це дозволило значно зменшити кількість синтаксичних помилок ще на етапі написання скриптів. Крім того, вбудований термінал дозволив виконувати команди командного рядка без необхідності перемикання між різними вікнами операційної системи.

Для підвищення ефективності розробки було використано ряд спеціалізованих розширень середовища:

- Prettier — Code formatter: Використовувався для автоматичного форматування коду згідно із загальноприйнятими стандартами (відступи, лапки, переноси рядків). Це забезпечило єдиний стиль оформлення коду у всіх модулях системи (HTML, CSS, JS), що є критично важливим для читабельності та підтримки проєкту.
- Live Server: Забезпечив створення локального сервера розробки з функцією «гарячого перезавантаження» (hot reload). Завдяки цьому будь-які зміни, внесені у код (наприклад, коригування формули розрахунку KPI або зміна кольору кнопки), миттєво відображалися у браузері без необхідності ручного оновлення сторінки.

Контроль версій та управління історією змін здійснювалися за допомогою розподіленої системи Git. Використання цієї системи дозволило вирішити кілька важливих завдань:

Безпека коду: Створення контрольних точок (комітів) після реалізації кожного логічного блоку (наприклад, «Додано функцію реєстрації», «Реалізовано алгоритм сортування завдань») дозволило застрахувати проєкт від критичних помилок. У разі некоректної роботи нововведень завжди існувала можливість швидкого відкату (rollback) до попередньої стабільної версії.

Розгалуження (Branching): Розробка нових функцій велася в окремих гілках (branches), що дозволяло експериментувати з архітектурою додатку, не порушуючи роботу основної версії продукту (master/main branch).

У поєднанні з хмарним репозиторієм (наприклад, GitHub), це забезпечило надійне збереження вихідного коду та можливість доступу до проєкту з різних пристроїв.

2.2 Проєктування архітектури системи

При створенні сучасних веб-орієнтованих систем управління завданнями та персональної ефективності ключовим етапом проєктування є вибір та обґрунтування архітектурного паттерну. Враховуючи вимоги до високої інтерактивності інтерфейсу та необхідність миттєвого відображення змін у статистиці користувача, для системи «Task Intelligence System» було спроєктовано архітектуру, що базується на принципах Single Page Application (SPA) з використанням клієнтського рендерингу (Client-Side Rendering — CSR).

Вибір на користь SPA обумовлений такими факторами:

- Покращений користувацький досвід (User Experience): На відміну від класичних багатосторінкових сайтів (MPA), де кожна дія призводить до повного перезавантаження сторінки, SPA завантажує HTML-каркас, стилі та скрипти лише один раз. Подальша взаємодія відбувається шляхом динамічного оновлення вмісту DOM-дерева. Це створює відчуття роботи з нативним десктопним додатком, що є критично важливим для інструменту повсякденного планування.
- Оптимізація мережевого трафіку: Взаємодія з сервером (або локальним сховищем) відбувається у фоновому режимі за допомогою асинхронних запитів. Система передає лише «корисні дані» у форматі JSON (JavaScript Object Notation), а не важку HTML-розмітку. Це значно прискорює відгук системи, особливо при розрахунку показників ефективності (KPI).

Внутрішня архітектура додатку побудована з використанням адаптованого шаблону MVC (Model-View-Controller), що дозволило чітко розмежувати логіку роботи системи:

- Model (Модель): Відповідає за структуру даних (масиви завдань, об’єкти статистики) та бізнес-логіку (формули розрахунку рейтингу, перевірка

дедлайнів). У розробленій системі роль моделі виконують структури даних, що зберігаються та валідуються перед записом.

- View (Вигляд): Відповідає за відображення інтерфейсу користувача. Завдяки клієнтському рендерингу (CSR), браузер самостійно генерує HTML-код карток завдань та графіків на основі даних, отриманих від Моделі.
- Controller (Контролер): Забезпечує обробку дій користувача (кліки, введення тексту, drag-and-drop). Контролер перехоплює події, викликає методи Моделі для зміни даних та ініціює оновлення Вигляду.

З огляду на вимоги до функціональності та умов використання веб-системи доцільно застосувати клієнтську архітектуру, орієнтовану на виконання основних операцій безпосередньо у браузері користувача. Такий підхід забезпечує швидкий доступ до функцій системи, мінімізує затримки при обробці подій та спрощує розгортання програмного продукту.

Архітектура розроблюваної веб-системи базується на принципах Single Page Application (SPA), що передбачає динамічне оновлення вмісту сторінки без її повного перезавантаження. Використання SPA дозволяє зберігати поточний стан застосунку, забезпечувати безперервну взаємодію користувача з системою та підвищувати зручність роботи з інтерфейсом, що є особливо важливим при управлінні великою кількістю завдань.

Логічна структура системи включає три основні рівні: рівень представлення, рівень прикладної логіки та рівень зберігання даних. Рівень представлення відповідає за формування інтерфейсу користувача та обробку подій взаємодії, прикладна логіка реалізує алгоритми управління завданнями та обчислення показників ефективності, а рівень зберігання даних забезпечує збереження інформації про користувачів і завдання у браузерному середовищі.

Взаємодія між компонентами системи реалізується за допомогою подієвого підходу, що дозволяє забезпечити слабку зв’язаність модулів та підвищити масштабованість програмного рішення. Зміни стану завдань, створення нових

записів або оновлення аналітичних показників обробляються у прикладному шарі та відображаються в інтерфейсі в режимі реального часу.

Застосування описаної архітектури створює передумови для реалізації інтелектуальної складової системи, оскільки дозволяє інтегрувати аналітичні алгоритми безпосередньо у логіку клієнтського застосунку. Це забезпечує оперативний розрахунок показників ефективності та формування узагальнених результатів діяльності користувача без залучення додаткових серверних ресурсів.

Такий підхід забезпечує модульність системи: логіка розрахунку ефективності відокремлена від коду, що відповідає за малювання кнопок. Це значно спрощує процес налагодження, тестування та подальшого масштабування системи (наприклад, додавання нових метрик аналізу без необхідності переписувати весь інтерфейс).

2.2.1 Загальна архітектура взаємодії компонентів

Система побудована за модульним принципом, де кожен модуль відповідає за окрему функціональну область. Архітектуру можна представити у вигляді трьох рівнів:

- Рівень представлення (View Layer): Відповідає за візуалізацію даних та взаємодію з користувачем. Реалізований через HTML-шаблони та CSS-стилі. Ключовими компонентами цього рівня є: Модуль авторизації (#login-container). Головний дашборд (.main-layout). Компоненти завдань та календаря.
- Рівень бізнес-логіки (Business Logic Layer): Реалізований на мові JavaScript. Цей рівень обробляє події користувача (кліки, введення даних), виконує валідацію, розраховує статистику ефективності та керує станом додатку (State Management).
- Рівень даних (Data Layer): Відповідає за довгострокове зберігання інформації. У даній системі роль бази даних виконує локальне сховище браузера (LocalStorage), яке працює через інтерфейс Storage API.

Такий підхід дозволяє розвантажити серверну частину (у випадку її підключення в майбутньому) та перенести обчислювальне навантаження на пристрій клієнта, що є оптимальним для систем персонального планування.

2.2.2 Модель життєвого циклу веб системи

Оскільки розроблена система функціонує за принципом Single Page Application (SPA), її життєвий цикл повністю керується браузером клієнта. Процес роботи додатку можна розділити на послідовність суворо визначених етапів, починаючи від запиту до сервера за статичними файлами і закінчуючи переходом системи у стан очікування дій користувача.

Життєвий цикл ініціалізується в момент завантаження кореневого файлу index.html. Алгоритм запуску системи виглядає наступним чином:

Етап 1. Завантаження ресурсів та побудова DOM

На цьому етапі браузер отримує HTML-каркас сторінки та починає побудову об'єктної моделі документа (DOM). Паралельно відбувається: Синхронне завантаження таблиць стилів style.css, що формують візуальне представлення (CSSOM). Завантаження та виконання основного скрипту script.js. Система очікує подію DOMContentLoaded, яка сигналізує про те, що дерево елементів повністю побудовано і готове до маніпуляцій з боку JavaScript.

Етап 2. Перевірка сесії та маршрутизація

Критично важливим етапом є визначення статусу користувача. При старті скрипт звертається до тимчасового сховища sessionStorage для пошуку об'єкта currentUser.

Сценарій «Авторизований»: Якщо об'єкт існує, система пропускає етап введення пароля і миттєво ініціює перехід до головного робочого простору (Dashboard). Це реалізовано для покращення UX, щоб користувачу не доводилося вводити логін при кожному оновленні сторінки.

Сценарій «Гість»: Якщо ключ відсутній або пошкоджений, система примусово відображає форму входу (login-container), блокуючи доступ до функціональних модулів.

Етап 3. Ініціалізація та валідація даних

Система виконує зчитування основних масивів даних (users, tasks) з постійного сховища localStorage. На цьому етапі реалізовано механізм «холодного старту»: Відбувається спроба десеріалізації JSON-рядків (JSON.parse()). Якщо дані відсутні (перший запуск системи на пристрої), ініціалізуються порожні масиви або дефолтні значення, що запобігає виникненню критичних помилок ReferenceError або TypeError у подальшому коді.

Етап 4. Динамічний рендеринг інтерфейсу

Функція prepareRoleView() аналізує роль поточного користувача (отриману на Етапі 2) та динамічно модифікує DOM-дерево: Для ролі 'Lead': Генеруються адміністративні панелі, кнопки перегляду звітів та розширені фільтри. Для ролі 'Member': Приховуються елементи управління чужими завданнями, інтерфейс фокусується виключно на особистих задачах. Цей етап завершується викликом функцій відмальовування списку завдань renderTasks() та оновленням статистики.

Етап 5. Реєстрація обробників подій

Фінальний етап ініціалізації. Система «навішує» слухачі подій (addEventListener) на інтерактивні елементи: кнопки створення завдань, поля введення, фільтри та модальні вікна. Після цього життєвий цикл переходить у стадію «Idle» (Очікування), реагуючи лише на дії користувача.

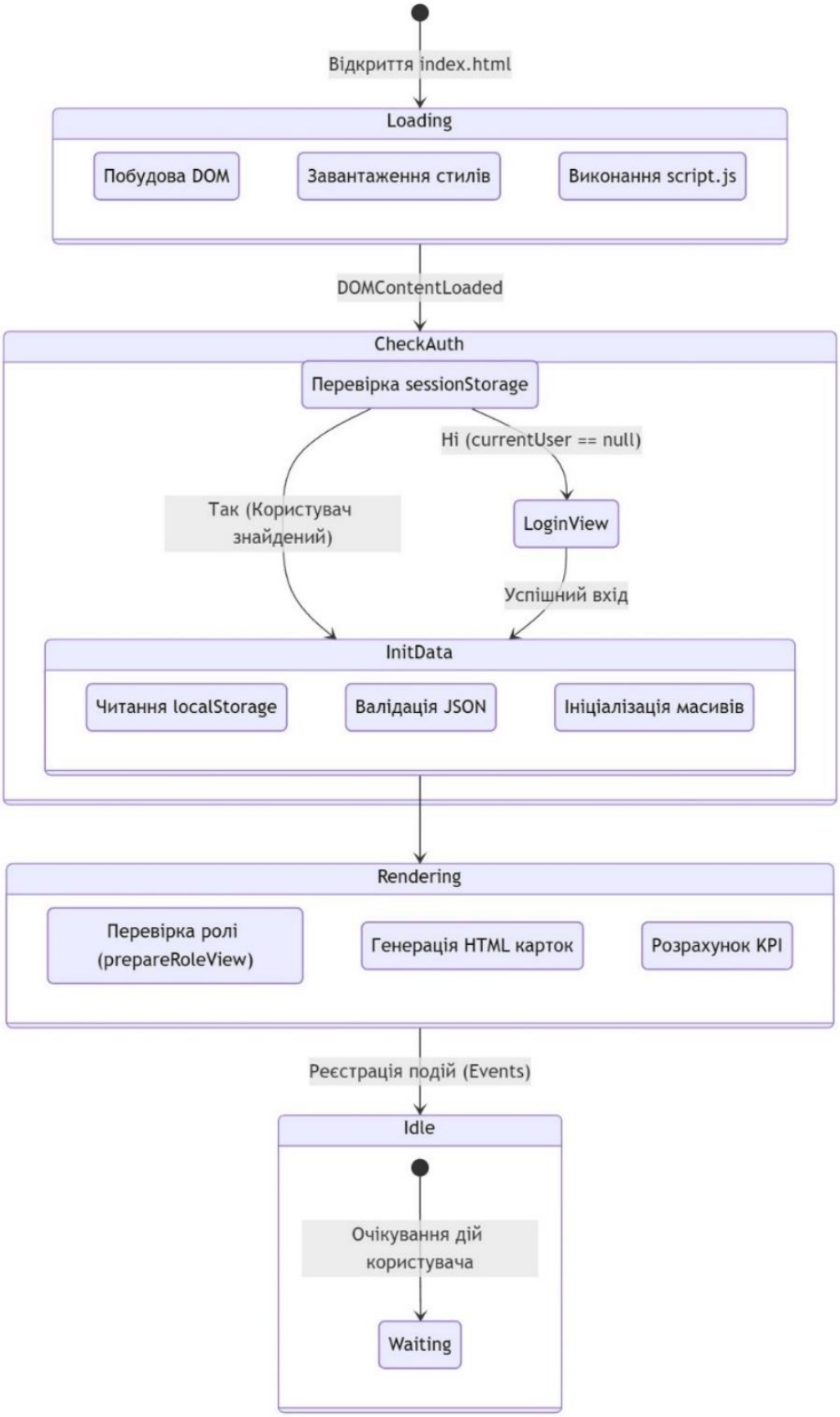

Рисунок 2.1 Модель життєвого циклу веб-системи

2.2.3 Проєктування логіки розмежування прав доступу

Важливою складовою архітектури є система контролю доступу на основі ролей (Role-Based Access Control). У системі виділено дві ролі з різними рівнями доступу:

Роль «Тім Лід» (lead):

- Має повний доступ до всіх модулів системи.
- Може створювати завдання та призначати їх будь-якому зареєстрованому користувачеві.
- Має доступ до модуля аналітики (#stats-panel) та розширених звітів ефективності.
- Має право видаляти завдання та змінювати їх статус примусово.

Роль «Працівник» (member):

- Має обмежений доступ
- Бачить лише завдання, призначені особисто йому.
- Може змінювати статус завдання тільки на «Виконано» або «Не виконано» (для власних задач).
- Не має доступу до панелі призначення виконавців (#assign-block) та загальної статистики.

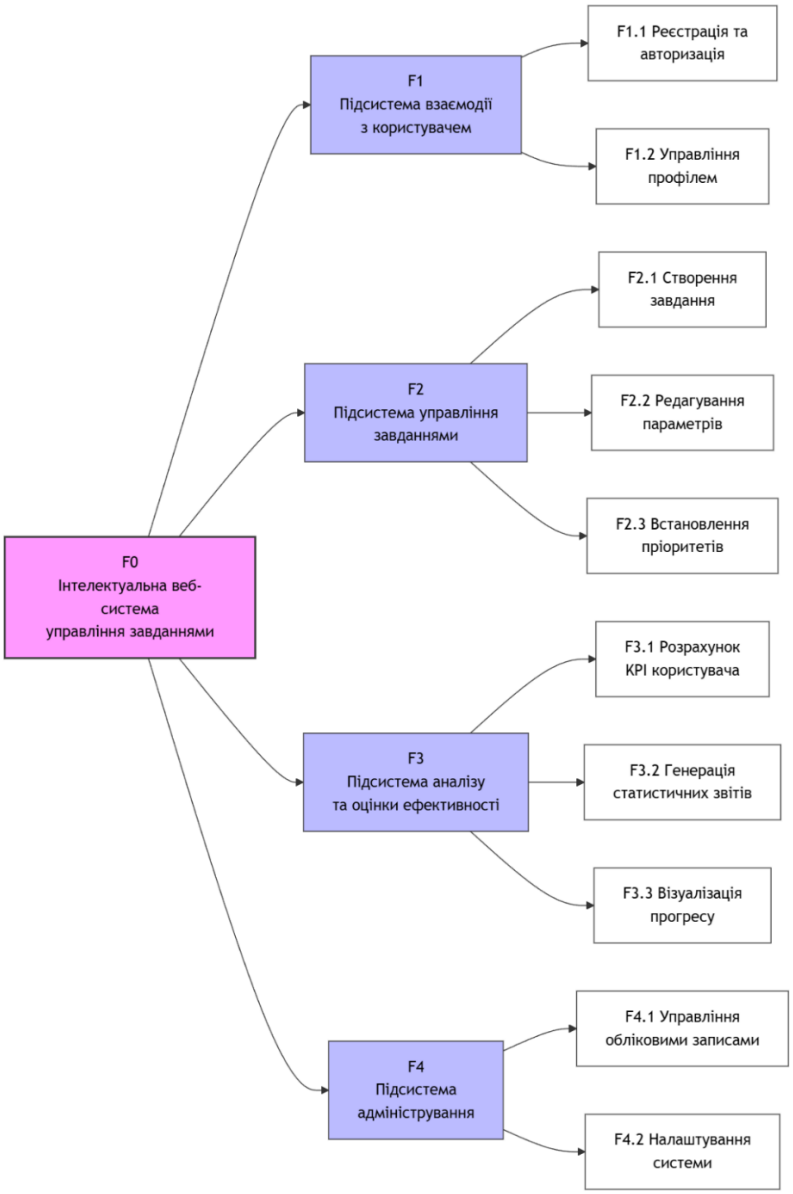

Рисунок 2.2 Логічна модель

Проєктування такої рольової моделі на етапі архітектури дозволяє забезпечити безпеку даних на рівні інтерфейсу та уникнути несанкціонованих дій з боку звичайних користувачів. Логіка перевірки ролей вбудована у кожну функцію рендерингу даних (renderTasks, renderStats), що гарантує цілісність відображення інформації

2.2.4 Взаємодія модулів системи

Система складається з чотирьох основних функціональних модулів, які взаємодіють між собою через глобальний стан (Global State):

Модуль **Auth**: Відповідає за вхід (login()) та вихід (logout()) із системи. Він оновлює глобальну змінну currentUser.

Модуль **Task Manager**: Відповідає за CRUD-операції (створення, читання, оновлення, видалення) над завданнями. Він безпосередньо взаємодіє з масивом tasks та викликає методи збереження saveTasks().

Модуль **Calendar**: Візуалізує завдання у вигляді календарної сітки. Цей модуль є залежним від даних Task Manager, оскільки він фільтрує завдання за датою дедлайну для відображення у відповідних комірках.

Модуль **Analytics**: (Тільки для Ліда) Агрегує дані про виконання завдань, розраховує коефіцієнти ефективності та будує звіти. Цей модуль виконує лише операції читання (Read-only) над масивом tasks.

Така модульна архітектура забезпечує низьку зв'язність коду (Low Coupling), що спрощує подальшу підтримку та масштабування системи, наприклад, додавання нових типів завдань або нових ролей.

2.2.5 Функціональне моделювання системи засобами UML

Для деталізації поведінки системи та взаємодії користувачів з інтерфейсом було розроблено діаграму прецедентів (Use Case Diagram) уніфікованою мовою моделювання UML. Визначено два типи акторів (Actors): "Працівник" (Member) та "Тім Лід" (Lead).

Актор "Працівник" має базовий набір прав доступу, необхідний для виконання персональної роботи. До основних прецедентів цього актора відносяться:

- Авторизація: Вхід у систему з вказанням унікального імені.
- Перегляд власних завдань: Доступ до списку задач, де поле assignedTo збігається з ім'ям користувача.
- Зміна статусу: Переведення завдання зі стану "New" у стан "Done" або "Failed".
- Робота з календарем: Перегляд дедлайнів у візуальній сітці.

Актор "Тім Лід" успадковує всі права "Працівника", але має розширені можливості адміністрування. Специфічні прецеденти для цієї ролі включають:

- Призначення завдань (Assign Task): Можливість вибору виконавця зі списку зареєстрованих користувачів при створенні завдання.
- Глобальний моніторинг: Перегляд завдань усіх користувачів системи з можливістю фільтрації.
- Видалення завдань: Право на безповоротне видалення будь-якого запису з бази даних.
- Генерація аналітичного звіту: Ініціювання розрахунку інтелектуального показника KPI та відправка результатів електронною поштою.

Життєвий цикл завдання (Task Lifecycle) змодельовано за допомогою діаграми станів (State Machine Diagram). Об'єкт "Завдання" може перебувати в одному з трьох станів:

- Active (New): Початковий стан після створення. Завдання відображається у центральному списку та календарі.
- Completed (Done): Кінцевий позитивний стан. Завдання переміщується у блок історії ("Виконані"). Враховується з позитивним васовим коефіцієнтом у статистиці.
- Failed: Кінцевий негативний стан (прострочено або скасовано). Враховується зі штрафним коефіцієнтом.

Таке моделювання дозволило чітко розмежувати зони відповідальності користувачів ще до етапу написання програмного коду.

2.3 Проєктування структури даних систем

Враховуючи архітектурні особливості системи як клієнтського SPA-застосунку (Single Page Application), для організації рівня збереження даних (persistence layer) було обрано документо-орієнтований підхід. Замість використання класичної реляційної бази даних (RDBMS) зі складними зв'язками,

система оперує структурованими JSON-колекціями, що зберігаються у локальному сховищі браузера (LocalStorage).

Для забезпечення коректної роботи веб-системи важливим є формування логічної структури даних, яка відображає основні сутності предметної області та зв’язки між ними. Чітка організація даних дозволяє спростити реалізацію прикладної логіки, забезпечити цілісність інформації та створити передумови для подальшого розширення функціональних можливостей системи.

У межах розроблюваної веб-системи доцільно використовувати логічну модель даних, орієнтовану на зберігання інформації про користувачів, їх ролі та персональні завдання. Основними атрибутами завдання є ідентифікатор, назва, рівень пріоритету, термін виконання, статус та прив’язка до конкретного користувача. Така структура дозволяє реалізувати механізми фільтрації, сортування та аналітичної обробки даних з урахуванням ключових параметрів.

Для зберігання даних у системі використовується механізм LocalStorage, який забезпечує постійне збереження інформації у браузері користувача. Застосування LocalStorage є доцільним для систем персонального призначення, оскільки дозволяє зберігати дані між сесіями без використання серверної частини та додаткових засобів автентифікації. Доступ до даних здійснюється у вигляді пар «ключ–значення», що спрощує реалізацію операцій читання та запису.

З метою забезпечення структурованого зберігання складних об’єктів дані у LocalStorage зберігаються у форматі JSON. Це дозволяє представляти інформацію у вигляді вкладених структур, що відповідають логічній моделі системи, та спрощує їх обробку у прикладній логіці застосунку.

Альтернативними підходами до зберігання даних у веб-системах є використання IndexedDB або зовнішніх серверних баз даних. Проте застосування таких рішень у межах даного проєкту є недоцільним з огляду на складність реалізації, надмірність функціоналу та відсутність потреби у синхронізації даних

між різними пристроями. Обраний підхід дозволяє досягти оптимального балансу між простотою реалізації та функціональністю системи.

Такий вибір технологічного стеку обумовлений ізоморфністю структур даних: об'єкти, що використовуються в програмному коді JavaScript, зберігаються в базі даних у майже незмінному вигляді. Це дозволяє:

- Уникнути надлишкової складності: Відпадає необхідність у використанні ORM (Object-Relational Mapping) систем для перетворення даних.
- Підвищити швидкодію: Операції читання та запису відбуваються миттєво без мережевих затримок, оскільки дані знаходяться безпосередньо на пристрої користувача.
- Денормалізація даних: Використання вкладених структур замість операцій об'єднання таблиць (JOIN), що є характерним для NoSQL-рішень.

Інформаційна модель системи складається з двох основних сутностей-агрегатів: «Користувач» (User) та «Завдання» (Task). Взаємозв'язок між ними реалізовано за принципом «один-до-багатьох» (One-to-Many), де один користувач може володіти множиною завдань.

Структура колекцій даних. Зберігання інформації реалізовано через ключ-значення (Key-Value), де значенням виступає серіалізований JSON-рядок. Нижче наведено спроєктовану структуру об'єктів системи:

1. Колекція users (Користувачі) Містить дані для автентифікації та налаштування профілю.

Фрагмент кода

JSON

```
{
  "id": "u_1709823",
  "username": "Student2025",
  "email": "student@ukr.net",
  "password": "***",
  "createdAt": "2025-10-21T10:00:00Z"
```

}

2. Колекція tasks (Завдання) Є основною робочою сутністю. Містить як описові поля, так і метрики для розрахунку KPI.

JSON

```
{
  "id": 101,
  "userId": "u_1709823",
  "title": "Написання розділу 2",
  "deadline": "2025-12-10",
  "priority": "high",
  "status": "done",
  "completedAt": "2025-12-09"
}
```

Механізм взаємодії з даними реалізовано через набір CRUD-операцій (Create, Read, Update, Delete), які інкапсульовані в окремому модулі сервісного шару додатку. Перед збереженням будь-яких змін система автоматично валідує цілісність структури JSON, щоб запобігти помилкам під час подальшого парсингу (deserialization).

2.3.1 Логічна структура сутності «Користувач»

Дані про зареєстрованих користувачів системи серіалізуються та зберігаються у ключі локального сховища users. Структура цієї колекції являє собою масив об'єктів, кожен з яких описує окремий обліковий запис. Проєктування цієї сутності базується на принципі мінімальної достатності атрибутів, необхідних для забезпечення функціонування механізмів авторизації та рольового розмежування прав доступу (RBAC — Role-Based Access Control).

Схема об'єкта User включає наступні поля, кожне з яких відіграє критичну роль у бізнес-логіці системи:

Атрибут name (Тип: String, Unique) Це поле виконує роль первинного ключа (Primary Key) у межах клієнтської бази даних.

- Призначення: Використовується для однозначної ідентифікації користувача при встановленні реляційних зв'язків із сутністю «Завдання» (поле owner у завданнях посилається на name користувача).
- Валідація: При реєстрації система перевіряє введене значення на унікальність у масиві, щоб уникнути колізій даних та дублювання профілів.

Атрибут role (Тип: String, Enumerated) Визначальний атрибут для підсистеми безпеки, що регламентує рівень доступу до функціонала. Реалізовано два рівні ієрархії:

- 'lead' (Тім Лід / Адміністратор): Користувач із цим маркером отримує розширені права. Інтерфейс для цієї ролі включає панелі глобальної аналітики, можливість перегляду ефективності інших працівників та інструменти для генерації зведених звітів.
- 'member' (Працівник / Виконавець): Базовий рівень доступу. Інтерфейс суворо обмежений переглядом лише власних завдань та особистої статистики (Personal Dashboard), що забезпечує конфіденційність індивідуальних показників ефективності.

3. Атрибут email (Тип: String, Optional) Додаткове службове поле, яке використовується підсистемою сповіщень.

- Призначення: Зберігає контактні дані для автоматизованої відправки PDF-звітів про ефективність роботи (KPI Report).
- Динаміка: Поле може бути незаповненим на етапі реєстрації та додаватися динамічно через налаштування профілю або адміністративну панель керівника.

Приклад реалізації JSON-структури:

Для наочності наведемо фрагмент коду, що демонструє збереження двох користувачів з різними ролями у форматі JSON: Фрагмент кода

JSON

```
[
  {
```

```
    "name": "Admin_Alex",
    "role": "lead",
    "email": "alex.lead@company.com",
    "registeredAt": "2025-10-21"
  },
  {
    "name": "Dev_Ivan",
    "role": "member",
    "email": null,
    "preferences": { "theme": "light" }
  }
]
```


Така структура даних дозволяє системі миттєво визначати права поточного користувача під час входу (login), рендерити відповідний варіант навігаційного меню (наприклад, приховувати кнопку «Адмінка» для ролі member) та фільтрувати масив завдань, відображаючи лише релевантні записи.

2.3.2 Логічна структура сутності «Завдання»

Сутність «Завдання» є центральним елементом системи і зберігається у ключі tasks. Структура об'єкта спроєктована таким чином, щоб забезпечити зберігання не лише описової частини, але й метаданих, необхідних для роботи алгоритму оцінки ефективності.

Схема об'єкта Task включає наступні атрибути:

Id : Унікальний числовий ідентифікатор, що генерується на основі поточної часової мітки (Date.now()). Це гарантує унікальність кожного завдання в межах системи.

title (String): Текстовий опис суті завдання.

deadline (String): Дата граничного терміну виконання у форматі ISO (YYYY-MM-DD). Використовується модулем календаря для відображення задачі у відповідній чарунці.

priority (String): Рівень пріоритетності завдання. Приймає значення зі фіксованого списку:
'**low**' (Низький);
'**medium**' (Середній)
'**high**' (Високий). Цей атрибут є критично важливим для роботи інтелектуального алгоритму оцінки, оскільки кожному рівню відповідає певний ваговий коефіцієнт.
assignedTo (String): Ім'я виконавця. Це поле реалізує логічний зв'язок "один-до-багатьох" між користувачем та завданнями.
createdBy (String): Ім'я автора завдання (для аудиту).
status (String): Поточний стан життєвого циклу завдання. Можливі стани:
'**new**' — завдання в роботі;
'**done**' — успішно виконано;
'**failed**' — провалено/прострочено.

2.3.3 Реалізація зв'язків та цілісності даних

У системі реалізовано неявні зв'язки між сутностями. Зв'язок між завданням та користувачем здійснюється через строкове поле assignedTo, яке повинно відповідати полю name у колекції користувачів. Приклад програмної реалізації структури даних наведено у коді нижче.

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

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

// Приклад формування об'єкта завдання перед збереженням

let task = {

   id: Date.now(),

   title: title

   deadline: deadline
```

```
    priority: priority,
    assignedTo: assignedTo,
    createdBy: currentUser.name,
    status: "new"
};
tasks.push(task);
saveTasks();
```

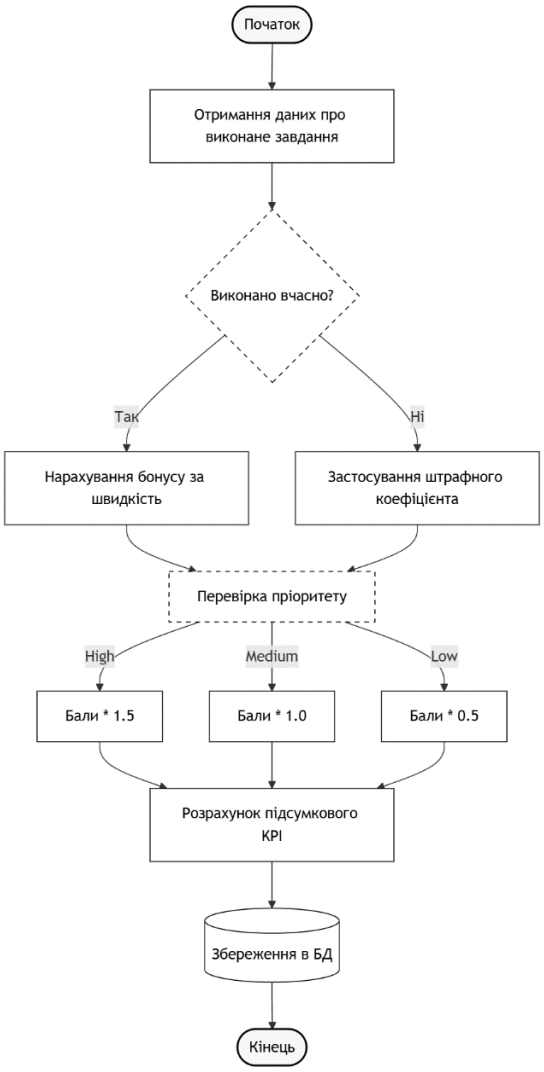

Рисунок 2.2 Блок схема алгоритму

Така структура даних є оптимальною для обраної архітектури, оскільки дозволяє виконувати фільтрацію завдань (наприклад, відображення завдань лише конкретного виконавця t.assignedTo === currentUser.name) без виконання складних операцій об'єднання таблиць (JOIN), характерних для SQL-баз даних.

2.3.4 Проєктування інтерфейсу користувача (UI/UX)

Під час проєктування інтерфейсу системи головним пріоритетом було забезпечення інтуїтивно зрозумілої взаємодії (Usability) та зниження когнітивного навантаження на користувача. Інтерфейс спроєктовано за принципом Dashboard (Інформаційна панель), де вся ключова інформація доступна на одному екрані.

2.3.5 Кольорова схема та візуальний стиль

Візуальна концепція системи розроблена з урахуванням сучасних ергономічних вимог до веб-інтерфейсів та базується на адаптованих принципах методології Material Design. Головною метою дизайну є зниження когнітивного навантаження на користувача та забезпечення інтуїтивно зрозумілої навігації. Для реалізації ефекту просторової глибини та акцентування уваги на інтерактивних елементах (картки завдань, модальні вікна) використано систему м'яких градієнтів та багаторівневих тіней (box-shadow), що дозволяє візуально відокремити контент від фону по осі Z.

Колористичне рішення проєкту сформовано на основі принципів кольорового кодування, де кожен відтінок несе чітке семантичне навантаження, забезпечуючи швидку ідентифікацію станів системи:

Фонове середовище (Background): Використано світлий градієнтний перехід (від #f5f8fc до #e8ecf8). Такий вибір палітри створює нейтральне, «повітряне» робоче середовище, яке не втомлює зір користувача при тривалій роботі, на відміну від абсолютно білого кольору, що часто створює надмірний контраст.

Акцентний колір (Primary Blue): Синій колір (#4a73f3) обрано як домінантний для інтерактивних елементів (CTA-кнопки «Додати завдання», активні посилання, перемикачі). У психології кольору цей відтінок асоціюється з продуктивністю,

фокусом та технологічністю, що відповідає тематиці системи управління завданнями.

Індикація статусів (Status Indication): Для миттєвої оцінки стану виконання завдань застосовано загальноприйняту світлофорну схему, що забезпечує інтуїтивне розуміння інтерфейсу:

Успіх: Насичений зелений (#10b981) маркує завершені завдання (статус «Done») та позитивну динаміку KPI.

Увага/Помилка: Червоний (#ef4444) сигналізує про прострочені дедлайни («Failed»), критично низький рейтинг ефективності або незворотні дії (видалення).

Візуалізація активності: Для виділення активних карток завдань та надання інтерфейсу сучасної естетики використано фіолетовий градієнт (#667eea). Це дозволяє візуально пріоритезувати поточні задачі на тлі вже виконаних або запланованих.

2.3.6 Компонування елементів

Візуальна структура веб-системи побудована за модульним принципом з використанням семантичних тегів HTML5 (section, div, h2). Головний екран програми (#app) прихований до моменту успішної авторизації і стає видимим лише після перевірки облікових даних, що реалізовано шляхом маніпуляції класом .hidden.

Архітектура сторінки розділена на дві логічні області: Верхня панель управління та Основний робочий простір.

Верхня частина інтерфейсу (.top-panel) слугує для відображення привітання користувача та кнопки виходу. Вона відокремлена від основного контенту тінями та відступами для створення ефекту «парячої» панелі.

Основний контент розміщено у контейнері .main-layout, який реалізує двоколонкову верстку для розділення операційної та аналітичної діяльності:

1. Ліва колонка (.left-column): Є основною робочою зоною. Тут розміщено:

- Модуль календаря (#calendar-panel) для візуалізації дат.
- Блок фільтрації та створення завдань

- Динамічний список завдань (#task-list).

2. Права колонка (.right-column): Виконує інформаційну функцію. Тут розміщено:

- Панель статистики (#stats-panel), яка за замовчуванням прихована і відображається лише для керівників.
- Список завершених або провалених задач (#completed-panel).

Фрагмент коду:

```
<div class="main-layout">
  <section class="left-column">
    <div id="calendar-panel" class="card hidden">...</div>
    <div id="task-creator" class="card">...</div>
    <div id="task-list"></div>
  </section>
  <section class="right-column">
    <div id="stats-panel" class="card hidden">...</div>
    <div id="completed-panel" class="card">...</div>
  </section>
</div>
```

Для стилізації використано CSS-властивості flexbox та градієнтні заливки, що задаються у класі body та .top-panel (файл style.css), забезпечуючи сучасний вигляд (Glassmorphism):

```
.top-panel {
  display: flex;
  justify-content: space-between;
  background: linear-gradient(135deg, #2d3748 0%, #374152 100%);
  box-shadow: 0 4px 12px rgba(0,0,0,0.1);
  border-radius: 12px;
}
```

Ключовою особливістю реалізації є клієнтський рендеринг, що залежить від ролі користувача (lead або member). У файлі script.js реалізовано логіку, яка перевіряє об'єкт currentUser і модифікує HTML-код "на льоту".

Система використовує шаблонні рядки (Template Literals) для генерації карток завдань. При цьому кнопки видалення завдань додаються у DOM-дерево тільки якщо користувач має роль «Тім Лід» і завдання має статус «нове».

Фрагмент коду:

```
const showDeleteBtn = currentUser.role === 'lead' && t.status === 'new';
html += `
  <div class="task ${priorityClass} ${statusClass}">
    <div class="task-header">
      <div class="task-title">${t.title}</div>
      ${showDeleteBtn ?
        `<button class="delete-btn" onclick="deleteTask(${t.id})">×</button>`
        : ''}
    </div>
    <div class="task-meta">
      <span>Дедлайн: ${t.deadline}</span>
      <span>Пріоритет: ${priorityLabels[t.priority]}</span>
    </div>
  </div>
`;
```

Такий підхід дозволяє використовувати єдиний HTML-шаблон для всіх користувачів, забезпечуючи безпеку інтерфейсу на рівні JavaScript-логіки. Крім того, панель статистики (#stats-panel) програмно приховується для звичайних працівників, щоб не перевантажувати їх інтерфейс зайвою інформацією.

2.3.7 Адаптивність

Для забезпечення кросплатформенності та коректного відображення системи на широкому спектрі пристроїв (від широкоформатних моніторів до смартфонів)

було реалізовано стратегію адаптивної верстки (Responsive Web Design). У файлі стилів style.css застосовано механізм CSS Media Queries, який дозволяє динамічно змінювати властивості DOM-елементів залежно від параметрів екрана користувача (ширини viewport).

Ключовим пороговим значенням (breakpoint) обрано ширину 900 пікселів. Це значення є оптимальним для розділення десктопного режиму та режиму планшета/смартфона. Логіка трансформації інтерфейсу реалізована наступним чином:

Трансформація Flex-контейнерів: При зменшенні ширини екрана нижче 900px, основний макет сторінки (.main-layout), побудований на технології Flexbox, змінює напрямок осей з горизонтального (row) на вертикальний (column). Це дозволяє уникнути горизонтального прокручування (скролу), яке є критичним недоліком для мобільних інтерфейсів, та вибудовує блоки контенту (сайдбар, список завдань, календар) в одну зручну для читання колонку.

Оптимізація під сенсорне керування (Touch UI): Ширина контейнерів входу та карток завдань автоматично збільшується до 90-100% ширини екрана. Це зроблено з метою покращення ергономіки взаємодії: збільшені елементи інтерфейсу легше натискати пальцем на сенсорних екранах, що мінімізує кількість помилкових кліків (так звана проблема "fat finger").

Нижче наведено фрагмент коду CSS, що демонструє реалізацію адаптивного правила:

CSS

```
@media (max-width: 900px) {
   .main-layout {
      flex-direction: column;
      gap: 16px;
   }
   .side-column {
      width: 100%;
```

```
        order: -1; /
    }
    .login-container {
        width: 90%;
        padding: 20px;
    }
}
```

Такий підхід до проєктування UI/UX гарантує, що система залишається функціональною та зручною незалежно від контексту використання: як для аналітичної роботи в офісі за великим монітором, так і для оперативного контролю виконання завдань "на ходу" за допомогою смартфона.

2.3.8 Психофізіологічні аспекти дизайну інтерфейсу

При розробці візуального стилю системи було застосовано принципи "спокійного дизайну" (Calm Technology), спрямовані на зниження когнітивного навантаження користувача при тривалій роботі.

Колористичне рішення: Базова палітра побудована на відтінках синього та сірого кольорів, що, згідно з дослідженнями ергономіки, сприяє концентрації уваги.

- Фон сторінки реалізовано через складний лінійний градієнт (linear-gradient(135deg, #f5f8fc, #e8ecf8)), який імітує ефект денного світла і зменшує контрастне навантаження на очі.
- Для акцентування уваги на інтерактивних елементах (картки завдань) використано ефект "глибини" за допомогою тіней (box-shadow: 0 8px 24px...).

Мікровзаємодії та анімація: Для забезпечення зворотного зв'язку системи реалізовано CSS-анімації. Зокрема, для фонових елементів використано анімацію float (плавання), яка створює динамічний, але ненав'язливий візуальний ряд. Ключові кадри анімації (keyframes) налаштовані на зміну позиції (transform: translate3d) з інтервалом у 20 секунд, що забезпечує плавність руху 60 кадрів на

секунду завдяки використанню апаратного прискорення графічного процесора (GPU).

Типографіка: Використання шрифту Manrope з гуманістичними гротескними рисами покращує читабельність цифрових даних (дат, статистики) на екранах з різною роздільною здатністю.

2.4 Функціонування веб-системи та взаємодія користувачів

Функціонування веб-системи базується на послідовності сценаріїв взаємодії користувача з інтерфейсом, які охоплюють процеси створення, редагування та аналізу персональних завдань. Кожна роль користувача має визначений набір доступних дій, що забезпечує впорядкованість роботи системи та контроль доступу до її функціональних можливостей.

Користувач після проходження процедури авторизації отримує доступ до персонального робочого середовища, де відображається перелік актуальних завдань із зазначенням пріоритетів, термінів виконання та поточного статусу. Створення нового завдання передбачає введення основних параметрів, які надалі використовуються для аналітичної обробки та формування показників ефективності.

У процесі роботи із завданнями система автоматично відстежує зміну їх стану та фіксує ключові події, зокрема завершення або прострочення виконання. На основі цих даних здійснюється розрахунок аналітичних показників, що дозволяють оцінити результативність діяльності користувача за певний період часу. Такий підхід забезпечує інтеграцію інтелектуальної складової безпосередньо у процес взаємодії з системою.

Окрему увагу приділено сценаріям перегляду аналітичної інформації. Користувач має можливість отримувати узагальнені відомості про виконані та невиконані завдання, аналізувати дотримання дедлайнів та оцінювати власну продуктивність на основі кількісних показників. Візуалізація результатів у зрозумілому вигляді сприяє підвищенню усвідомленості та мотивації користувача.

Таким чином, функціонування веб-системи побудоване на поєднанні зручних механізмів управління завданнями та аналітичних інструментів, що дозволяє забезпечити ефективну взаємодію користувачів з інтелектуальним програмним середовищем.

2.4.1 Специфікація колекції користувачів

Оскільки система спроєктована як Single Page Application (SPA) без використання традиційної серверної реляційної бази даних (RDBMS), критично важливим етапом проєктування стала розробка ефективної структури зберігання даних на стороні клієнта. Для забезпечення персистентності (збереження стану між сесіями) було обрано механізм LocalStorage, що визначило необхідність застосування документо-орієнтованого підходу (Document-Oriented Approach), характерного для NoSQL систем.

На відміну від табличної структури SQL, де дані нормалізуються та розбиваються на зв'язані таблиці, у розробленій системі дані зберігаються у вигляді колекцій JSON-документів. Це дозволяє зберігати складні ієрархічні структури (наприклад, масив завдань всередині об'єкта проєкту) без необхідності виконання ресурсомістких операцій об'єднання (JOIN).

Колекція users відповідає за зберігання облікових даних та налаштувань профілю. Вона реалізована як масив об'єктів, що серіалізується у рядок JSON перед записом у сховище браузера.

Структура документа користувача спроєктована з урахуванням мінімальної надлишковості та включає наступні атрибути:

- name (String, Unique): Унікальний ідентифікатор користувача в системі. Виконує роль первинного ключа (Primary Key). Використовується для авторизації та як зовнішній ключ (Foreign Key) у завданнях для поля assignedTo.
- role (Enum: 'lead' | 'member'): Визначальний атрибут для системи контролю доступу (RBAC).

Значення 'lead' надає адміністративні права (перегляд статистики, видалення будь-яких завдань, призначення виконавців). Значення 'member' обмежує права доступу лише власними завданнями.

- email (String, Optional): Службове поле для інтеграції з зовнішніми поштовими клієнтами (використовується функцією генерації звітів sendLeadStatsEmail).

2.4.2 Специфікація колекції завдань

Колекція tasks є основною сутністю системи, що агрегує всю інформацію про діяльність користувачів. Для забезпечення можливості розрахунку інтелектуальних метрик ефективності (KPI), структура документа була розширена специфічними полями.

Атрибути сутності «Завдання»:

- id (Number/Timestamp): Унікальний числовий ідентифікатор. Генерується автоматично на основі поточної часової мітки (Date.now()), що гарантує унікальність в межах системи та дозволяє хронологічне сортування.
- title (String): Змістовий опис завдання.deadline (Date String ISO 8601): Дата граничного терміну виконання у форматі YYYY-MM-DD. Використовується модулем календаря для візуалізації завантаженості.
- priority (Enum: 'low', 'medium', 'high'): Рівень пріоритетності. Це поле є ключовим для математичної моделі оцінки ефективності, оскільки кожному рівню відповідає певний ваговий коефіцієнт ($W_{low}=1, W_{med}=2, W_{high}=3$).
- assignedTo (String): Посилання на виконавця. Забезпечує зв'язок "Один до багатьох" (User -> Tasks).createdBy (String): Поле аудиту, що фіксує автора завдання (важливо для сценаріїв, коли Тім Лід призначає завдання підлеглому).
- status (Enum: 'new', 'done', 'failed'): Поточний стан життєвого циклу завдання. Зміна цього поля ініціює перерахунок статистики користувача.

Така структура даних є денормалізованою, що дозволяє отримати повну інформацію про завдання (включно з пріоритетом та виконавцем) за одну операцію читання, суттєво підвищуючи швидкодію інтерфейсу на клієнтських пристроях з обмеженими ресурсами.

3. РОЗРОБКА ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ УПРАВЛІННЯ ПЕРСОНАЛЬНИМИ ЗАВДАННЯМИ ТА ОЦІНКИ ЕФЕКТИВНОСТІ

У цьому розділі розглядається практична реалізація веб-системи, включаючи програмний код основних модулів, реалізацію алгоритмів обробки даних та побудову інтерфейсу користувача.

3.1 Реалізація механізмів ініціалізації та управління станом системи

Основою функціонування Single Page Application (SPA) є коректне управління станом застосунку (State Management). Оскільки система працює без серверної бази даних, стан ініціалізується шляхом зчитування збережених даних із локального сховища браузера (localStorage)

Для забезпечення коректної роботи системи важливим є чітке управління внутрішнім станом веб-застосунку, який визначає поточний контекст взаємодії користувача з функціоналом. Під станом системи у межах даного проєкту розуміється сукупність даних про активного користувача, перелік його завдань, обрані параметри фільтрації та сортування, а також проміжні значення, що використовуються для аналічної обробки.

Особливістю розробленої системи є відсутність серверної частини, що зумовлює необхідність реалізації механізмів ініціалізації та збереження стану виключно на стороні клієнта. Такий підхід спрощує розгортання застосунку, зменшує вимоги до інфраструктури та робить систему незалежною від зовнішніх сервісів.

Керування станом реалізується таким чином, щоб при кожному запуску веб-застосунку забезпечувалось відновлення актуальних даних користувача та цілісність логіки роботи. Це створює основу для стабільного функціонування всіх подальших модулів системи, зокрема механізмів управління завданнями та інтелектуальної оцінки ефективності.

3.1.1 Ініціалізація глобальних змінних та персистентність даних

Процес ініціалізації програмного забезпечення є критичним етапом життєвого циклу веб-додатка, оскільки саме він відповідає за підготовку робочого середовища та відновлення стану системи. У розробленій системі цей етап базується на взаємодії з механізмом локального сховища браузера (localStorage), що дозволяє забезпечити збереження даних між сесіями користувача без необхідності постійного звернення до віддаленого сервера.

Алгоритм ініціалізації розпочинається автоматично при завантаженні сторінки (подія load або виконання скрипту в кінці body). Система виконує запит до сховища для отримання даних за ключовими ідентифікаторами users (масив користувачів) та tasks (масив завдань). Оскільки localStorage зберігає дані виключно у рядковому форматі, для перетворення отриманих значень у робочі об'єкти JavaScript застосовується метод десеріалізації JSON.parse().

У практичній реалізації важливо враховувати, що дані у localStorage можуть бути відсутні (перший запуск), частково заповнені або мати некоректний формат (наприклад, після ручного очищення сховища чи зміни структури даних під час доопрацювання застосунку). Тому ініціалізація повинна передбачати механізм значень за замовчуванням, який гарантує працездатність системи навіть за відсутності попередньо збережених відомостей.

З цією метою після зчитування даних виконується відновлення початкового стану у вигляді порожніх колекцій для users та tasks, якщо відповідні ключі сховища не містять значень. Такий підхід дозволяє уникнути помилок типу null або undefined під час подальшої обробки даних (наприклад, під час перебору масиву або побудови інтерфейсу). У свою чергу, використання JSON-серіалізації забезпечує структуроване представлення об’єктів і дозволяє зберігати складні сутності у вигляді набору атрибутів.

Окремо в системі реалізовано допоміжні функції персистентності, які інкапсулюють операції запису до localStorage для масивів користувачів та завдань. Така декомпозиція спрощує супровід коду, забезпечує повторне використання

логіки збереження у різних сценаріях (додавання, редагування, видалення) та зменшує ризик ситуацій, коли зміни відображаються в інтерфейсі, але не фіксуються у сховищі.

```
// Ініціалізація стану із LocalStorage
let users = JSON.parse(localStorage.getItem("users")) || [];
let tasks = JSON.parse(localStorage.getItem("tasks")) || [];
let currentUser = null;
// Функції для персистентності даних
function saveUsers() {
   localStorage.setItem("users", JSON.stringify(users));
}
function saveTasks() {
   localStorage.setItem("tasks", JSON.stringify(tasks));
}
```

Такий підхід забезпечує надійне зберігання даних між сесіями роботи браузера.

3.1.2 Реалізація логіки авторизації та сесій

Підсистема авторизації є ключовим модулем безпеки та персоналізації веб-додатка, що відповідає за ідентифікацію користувача та надання йому відповідних прав доступу. У рамках даного проекту реалізовано клієнтську імітацію механізму сесій, яка дозволяє зберігати стан авторизації без використання серверних технологій (cookies або JWT-токенів).

Для забезпечення безперервності роботи користувача (User Experience) та уникнення необхідності повторного входу в систему при кожному оновленні сторінки (F5), було обрано використання об'єкта веб-сховища sessionStorage. На відміну від localStorage, дані в sessionStorage існують лише в межах поточної вкладки браузера і автоматично видаляються після її закриття, що повністю відповідає життєвому циклу класичної веб-сесії.

Функція login(), яка ініціює процес входу, виконує ряд послідовних операцій, що забезпечують коректну обробку даних користувача:

Збір та первинна валідація даних: Система зчитує значення, введені користувачем у форму авторизації. На цьому етапі відбувається базова перевірка на наявність даних (щоб уникнути створення користувачів з пустими іменами) та, за необхідності, санітизація вводу (наприклад, видалення зайвих пробілів).

Алгоритм "Перевірка або Реєстрація": Система здійснює пошук введеного імені в масиві існуючих користувачів (users). Реалізовано гібридний підхід:

Якщо користувача знайдено, система завантажує його профіль та налаштування.

Якщо збігів не виявлено, спрацьовує логіка автоматичної реєстрації: створюється новий об'єкт користувача з базовими правами та додається до глобального масиву. Це спрощує взаємодію з прототипом, об'єднуючи процеси реєстрації та входу.

Ініціалізація сесії: Після успішної ідентифікації оновлюється глобальна змінна currentUser. Критично важливим кроком є серіалізація цього об'єкта та запис його у sessionStorage. Саме це дозволяє системі "пам'ятати" користувача при навігації між сторінками або їх перезавантаженні.

Адаптація інтерфейсу (Role-Based UI): Фінальним етапом є виклик функції prepareRoleView(). Цей метод відповідає за динамічну перебудову DOM-дерева сторінки. Залежно від ролі авторизованого користувача (наприклад, "Адміністратор" або "Користувач"), певні елементи керування стають видимими або приховуються, забезпечуючи розмежування прав доступу на рівні інтерфейсу. Програмна реалізація описаного алгоритму входу в систему наведена у коді нижче:

```
function login() {
  let name = document.getElementById("username").value;
  let role = document.getElementById("role").value;
  if (!name) return alert("Введіть ім'я!");
  currentUser = { name, role };
```

```
    // Перевірка існування користувача
    let exists = users.find(u => u.name === name);
    if (!exists) {
        users.push(currentUser); // Реєстрація нового
        saveUsers();
    } else if (exists.role !== role) {
        exists.role = role; // Оновлення ролі
        saveUsers();
    }
    // Перехід до робочого простору
    document.getElementById("login-container").classList.add("hidden");
    document.getElementById("app").classList.remove("hidden");
    prepareRoleView();
    renderTasks();
}
```

Цей механізм дозволяє динамічно керувати списком користувачів без необхідності створення окремої панелі адміністратора для реєстрації.

3.2 Програмна реалізація функціоналу управління завданнями

Модуль управління завданнями (Task Manager) виступає центральним функціональним вузлом розробленої веб-системи. Він реалізує повний цикл обробки даних згідно з парадигмою CRUD (Create, Read, Update, Delete), забезпечуючи створення, відображення, модифікацію статусу виконання та видалення завдань. Архітектура модуля спроєктована таким чином, щоб забезпечити миттєвий відгук інтерфейсу та надійну синхронізацію даних із локальним сховищем.

Процес ініціалізації нового завдання керується функцією addTask(). Ця функція виконує роль контролера, який агрегує вхідні дані від користувача, валідує їх та формує нову сутність у системі. Алгоритм роботи функції можна розділити на декілька етапів:

Збір та валідація даних: На першому етапі скрипт отримує доступ до значень елементів форми через DOM-дерево (поля назви завдання, дедлайну, пріоритету). Перед обробкою виконується перевірка на заповненість обов'язкових полів, що запобігає створенню "пустих" завдань та забезпечує цілісність даних.

Генерація унікального ідентифікатора: Для однозначної ідентифікації кожного завдання в межах системи використовується підхід генерації ID на основі часової мітки (timestamp). Метод Date.now() повертає кількість мілісекунд, що пройшли з 1 січня 1970 року. Таке число є унікальним для моменту створення, що дозволяє точно звертатися до конкретного об'єкта при подальших операціях редагування або видалення без ризику колізій.

Рольова логіка призначення виконавця (Assignment Logic): Важливою особливістю бізнес-логіки системи є адаптивний алгоритм визначення поля assignedTo (виконавець). Система автоматично перевіряє роль поточного авторизованого користувача (currentUser.role):

- Сценарій для ролі "Lead" (Тім Лід): Користувачі з розширеними правами мають можливість делегувати завдання. У цьому випадку значення виконавця зчитується з випадаючого списку (select), який динамічно заповнюється іменами всіх зареєстрованих у системі користувачів.
- Сценарій для ролі "Member" (Працівник): Звичайні користувачі обмежені у правах делегування. Для них система автоматично встановлює виконавцем самого автора завдання. Це реалізується шляхом прямого присвоєння імені поточного користувача (currentUser.name) у поле виконавця, ігноруючи вибір у інтерфейсі або приховуючи його.

Формування та збереження об'єкта: На основі зібраних даних створюється об'єкт-літерал завдання, який включає всі вищезазначені властивості, а також початковий статус (за замовчуванням — "pending" або "to do"). Новий об'єкт додається до глобального масиву tasks, після чого викликається функція saveTasks() для фіксації змін у localStorage та функція renderTasks() для миттєвого оновлення візуального представлення списку завдань.

Такий підхід забезпечує гнучкість системи та чітке розмежування відповідальності залежно від ролі користувача в команді.

3.2.1 Алгоритм створення нового завдання

Функція addTask() зчитує дані з полів введення (назва, дедлайн, пріоритет). Важливою особливістю є автоматичне визначення виконавця (assignedTo): Якщо користувач має роль lead (Тім Лід), він може обрати виконавця зі списку select. Якщо роль member (Працівник), завдання автоматично призначається самому собі. Кожне завдання отримує унікальний ідентифікатор id на основі Date.now(), що дозволяє точно ідентифікувати об'єкт при редагуванні або видаленні.

```
function addTask() {
   let title = document.getElementById("task-title").value;
   let deadline = document.getElementById("task-deadline").value;
   let task = {
      id: Date.now(),
      title,
      deadline,
      priority, // 'low', 'medium', 'high'
      assignedTo,
      createdBy: currentUser.name,
      status: "new"
   };
   tasks.push(task);
   saveTasks();
   renderTasks();
}
```

3.2.2 Візуалізація та фільтрація списку завдань

Функція renderTasks() відповідає за побудову DOM-дерева списку завдань. Вона виконує фільтрацію масиву tasks на основі ролі поточного користувача: Тім

Лід бачить всі завдання. Працівник бачить лише завдання, де assignedTo === currentUser.name.

Для покращення сприйняття реалізовано групування завдань за датою дедлайну. Спочатку створюється проміжний об'єкт tasksByDate, ключами якого є дати, а значеннями — масиви завдань. Потім відбувається ітерація по відсортованим датам і генерація HTML-коду.

```
const tasksByDate = {};
activeTasks.forEach(({ task, index }) => {
   if (!tasksByDate[task.deadline]) {
      tasksByDate[task.deadline] = [];
   }
   tasksByDate[task.deadline].push({ task, index });
});
sortedDates.forEach(date => {
   let dateSectionHtml = `<div class="task-date-section">`;
   dateSectionHtml += `<div class="task-date-header">${date}</div>`;
});
```

Така структура відображення дозволяє користувачеві швидко оцінити обсяг робіт на конкретний день.

3.3 Розробка та реалізація алгоритму інтелектуальної оцінки ефективності

Ключовою інновацією системи є модуль аналітики, доступний для ролі Тім Ліда. Він базується на розробленому математичному алгоритмі, який враховує не тільки кількість виконаних завдань, але й їхню складність (пріоритет).

3.3.1 Математична модель оцінки

Алгоритм computeLeadStats() використовує систему ваг:

- Низький пріоритет (low) = 1 бал.

- Середній пріоритет (medium) = 2 бали.
- Високий пріоритет (high) = 3 бали.

Загальна ефективність (E) розраховується як відношення зваженої суми виконаних завдань до загальної суми, з урахуванням штрафу за провалені завдання (коефіцієнт 0.5).

```
function computeLeadStats() {
   const weights = { low: 1, medium: 2, high: 3 };
   // ..
   let doneWeight = (done.low * weights.low) +
                    (done.medium * weights.medium) +
                    (done.high * weights.high);
   let failedWeight = (failed.low * weights.low) +
                      (failed.medium * weights.medium) +
                      (failed.high * weights.high);
   let score = 0;
   if (totalWeight > 0) {
      score = ((doneWeight - 0.5 * failedWeight) / totalWeight) * 100;
      score = Math.round(Math.max(0, Math.min(100, score)));
   }
}
```

Цей алгоритм стимулює працівників не лише виконувати завдання, але й уникати їх протермінування, особливо для задач з високим пріоритетом.

3.3.2 Генерація звітів та експорт даних

Результати розрахунків відображаються у модальному вікні, яке створюється динамічно через маніпуляцію DOM (функція openLeadStats). Для забезпечення комунікації реалізовано функцію sendLeadStatsEmail, яка формує текстовий звіт і відкриває поштовий клієнт за допомогою протоколу mailto:.

```
function sendLeadStatsEmail(stats) {
   let body = `Оцінка ефективності — ${new Date().toLocaleString()}\n\n`;
```

```
    stats.forEach(s => {
        body += `Користувач: ${s.name}\n`;
        body += `Успішність: ${s.completionRate}% · Оцінка: ${s.score}%\n`;
    });
    const mailto = `mailto:...?body=${encodeURIComponent(body)}`;
    window.location.href = mailto;
}'
```

3.4 Реалізація адаптивного інтерфейсу та календаря

Для зручності планування в системі реалізовано модуль календаря, який візуалізує завантаженість по днях місяця.

3.4.1 Генерація календарної сітки

Функція renderCalendar() динамічно створює сітку днів для поточного місяця. Вона розраховує зміщення першого дня місяця (щоб понеділок/неділя були на правильних місцях) та кількість днів. Для кожного дня виконується пошук активних завдань. Якщо завдань більше двох, відображається індикатор "+N", щоб не перевантажувати комірку.

```
        for (let day = 1; day <= daysInMonth; day++) {
            const dateStr = `${year}-${month}-${day}`;
            // Фільтрація завдань на конкретну дату
            let dayTasks = tasks.filter(t => t.deadline === dateStr && t.status === 'new');
            calendarHtml += `
                <div class="calendar-day" onclick="showDayTasks('${dateStr}')">
                    <div class="cal-day-number">${day}</div>
                    <div class="cal-day-tasks">...</div>
```

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

3.4.2 Адаптивність та стилізація

Зовнішній вигляд системи реалізовано за допомогою CSS3. Використано технологію CSS Grid для календаря, що дозволяє автоматично адаптувати розмір комірок. Для мобільних пристроїв (ширина екрану < 900px) застосовано медіа-запити, які змінюють напрямок розташування основних блоків (flex-direction: column).

```
.calendar-days {
    display: grid;
    grid-template-columns: repeat(7, 1fr
    gap: 6px;
}
@media (max-width: 900px) {
    .main-layout {
        flex-direction: column;
    }
    #login-container {
        width: 90%;
    }
```

Використання анімацій (transition: all 0.3s ease) для кнопок та карток завдань робить інтерфейс більш чутливим та приємним для користувача.

3.5 Тестування та верифікація програмного продукту

Для підтвердження працездатності розробленої системи було проведено комплексне функціональне тестування методом "чорної скриньки" (Black Box Testing). Тестування охоплювало перевірку рольової моделі, валідацію вхідних даних та коректність роботи математичних алгоритмів. Результати тестування ключових сценаріїв наведено у таблиці 3.1.

Таблиця 3.1 — Протокол функціонального тестування веб-системи

№	Назва сценарію	Вхідні дані	Очікувана поведінка	Фактичний результат	Статус
1	**Авторизація (Валідація)**	Поле "Ім'я" залишене порожнім.	Система блокує вхід, виводить попередження alert("Введіть ім'я!").	З’явилося спливаюче вікно з помилкою. Вхід не виконано.	**Passed**
2	**Авторизація (Роль: Працівник)**	Ім'я: "User1", Роль: "Member".	Приховуються панелі адміністрування та статистики.	Інтерфейс завантажено у спрощеному режимі. Блок assign-block приховано.	**Passed**
3	**Створення завдання**	Title: "Test Task", Priority: "High", Deadline: "2025-12-31".	Завдання додається в масив tasks і зберігається в localStorage.	Завдання з'явилося у списку. Після перезавантаження сторінки дані збереглися.	**Passed**
4	**Обмеження прав доступу**	Користувач з роллю "Member" намагається видалити завдання.	Кнопка видалення ("X") відсутня в інтерфейсі картки завдання.	Елемент DOM delete-btn не згенеровано функцією renderTasks.	**Passed**
5	**Розрахунок KPI (Сценарій А)**	1 завдання High (Done), 1 завдання Low (Done).	Score = (3+1)/(3+1) * 100% = 100%.	Система відображає: "Успішність: 100%".	**Passed**

Продовження таблиці 3.1

6	**Розрахунок KPI (Сценарій Б)**	1 завдання High (Failed).	Score = (0 - 0.5*3)/3 * 100% = -50% -> norm -> 0%.	Система відображає: "Успішність: 0%".	**Passed**
7	**Генерація звіту**	Натискання кнопки "Надіслати на пошту".	Відкриття поштового клієнта з сформованим тілом листа.	Клієнт відкрився, текст містить коректну статистику по всім юзерам.	**Passed**
8	**Робота календаря**	Перехід на наступний місяць (Стрілка вправо).	Перемальовування сітки днів, зміщення днів тижня.	Календар оновився коректно, 1 число місяця відповідає правильному дню тижня.	**Passed**

3.6 Розширений протокол функціонального тестування

Для перевірки відповідності розробленої системи технічним вимогам було проведено комплексне тестування методом «чорної скриньки».

Тестування проводилося у браузері Google Chrome (версія 120.0) та Safari (версія 17.1).

Результати випробувань зафіксовано у Протоколі №1 (Таблиця 3.2).

Таблиця 3.2 — Протокол тестування системи

№	**Об'єкт перевірки**	**Сценарій тестування (Test Case)**	**Очікувана поведінка**	**Фактичний результат**	**Статус**
1	**Авторизація**				
1.1	Валідація	Вхід з пустим полем імені.	Блокування входу, `alert`-повідомлення.	Вхід заблоковано, попередження показано.	**OK**

Продовжження таблиці 3.2

1.2	Роль «Member»	Вхід: «User1» (роль: Працівник).	Dashboard без адмін-панелей («Статистика», «Призначення»).	Інтерфейс завантажено без адмін-блоків.	**OK**
1.3	Роль «Lead»	Вхід: «Admin» (роль: Тім Лід).	Доступ до всіх модулів, календарю та статистики.	Відображено повний функціонал.	**OK**
2	**Завдання**				
2.1	Створення	Ввід: "Звіт", Дедлайн: "2025-12-31", Пріоритет: High.	Запис додано в список та `LocalStorage`. Поля очищено.	Завдання створено та збережено.	**OK**
2.2	Призначення (Lead)	Лід обирає виконавця «User1» зі списку.	У полі `assignedTo` записано «User1».	Виконавця призначено коректно.	**OK**
2.3	Призначення (Member)	Працівник створює завдання (вибір виконавця приховано).	Автоматичне призначення автору (`currentUser`).	Завдання закріплено за автором.	**OK**
2.4	Видалення (Lead)	Лід натискає «X» на завданні.	Підтвердження `confirm` -> Видалення.	Завдання видалено безповоротно.	**OK**

Продовжження таблиці 3.2

2.5	Видалення (Member)	Працівник шукає опцію видалення.	Кнопка «X» відсутня в DOM.	Кнопка видалення не рендеритьс я.	**OK**
3	**Логіка**				
3.1	Календар	Завдання з датою "2025-12-31".	Відображен ня у клітинці 31-го числа.	Завдання присутнє в сітці.	**OK**
3.2	KPI (Тест 1)	1 задача High (Done), 1 Low (Failed).	$(3 - 0.5 \cdot 1) / 4 \cdot 100\% \approx 63\%$.	Система показує: «Успішніст ь: 63%».	**OK**
3.3	KPI (Тест 2)	Виконано 0 завдань.	Обробка ділення на нуль -> 0%.	Система показує: «Успішніст ь: 0%».	**OK**
3.4	Збереження	Перезавантаження сторінки (F5).	Відновлення стану з `LocalStorag e`.	Дані не втрачено після оновлення.	**OK**
3.5	Експорт	Клік «Надіслати на пошту».	Відкриття поштового клієнта з даними.	Outlook відкриваєть ся з текстом звіту.	**OK**

ВИСНОВОК

У дипломній роботі вирішено актуальне науково-прикладне завдання створення інтелектуальної веб-системи для управління персональними завданнями користувача та оцінки ефективності їх виконання. В ході виконання роботи було проведено повний цикл розробки: від детального аналізу предметної області та існуючих підходів до тайм-менеджменту до програмної реалізації та тестування готового веб-ресурсу. Дослідження існуючих аналогів засвідчило, що більшість популярних планувальників орієнтовані насамперед на фіксацію списків справ, тоді як функція глибокого аналізу продуктивності та надання інтелектуального зворотного зв’язку в них реалізована недостатньо або відсутня, що й обумовило доцільність розробки власної системи.

Результатом етапу проєктування стала розробка гнучкої архітектури програмного забезпечення, здатної обробляти різнотипні дані про активність користувача. Було спроєктовано структуру бази даних, яка дозволяє ефективно зберігати інформацію про завдання, їх пріоритетність, складність та часові витрати. Вагомим здобутком цього етапу стала розробка алгоритмічної моделі оцінки ефективності, яка, на відміну від простих лічильників виконаних задач, враховує комплексні показники: дотримання дедлайнів, співвідношення запланованого та витраченого часу, а також рівень складності виконуваної роботи.

Практична цінність роботи полягає у створенні повнофункціональної інтелектуальної веб-системи із застосуванням сучасних технологій веб-розробки. Реалізовано зручний користувацький інтерфейс, що мінімізує час на внесення даних, та впроваджено модуль аналітики, який візуалізує прогрес користувача за допомогою динамічних графіків. Система не лише автоматизує процес обліку завдань, але й виступає інструментом самоконтролю, надаючи користувачеві об’єктивну оцінку його поточної продуктивності на основі накопиченої статистики.

Проведене тестування програмного продукту підтвердило працездатність усіх його модулів, коректність розрахунків коефіцієнтів ефективності та надійність системи захисту даних користувачів. Розроблена веб-система є завершеним рішенням, готовим до використання, яке дозволяє користувачам підвищити рівень особистої організованості, виявити проблемні аспекти у плануванні власного часу та оптимізувати робочі процеси на основі отриманих аналітичних даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про захист інформації в інформаційно-телекомунікаційних системах [Текст] : Закон України від 05.07.1994 р. № 80/94-ВР // Відомості Верховної Ради України. – 1994. – № 31. – Ст. 286.
2. Про авторське право і суміжні права [Текст] : Закон України від 23.12.1993 р. № 3792-XII // Відомості Верховної Ради України. – 1994. – № 13. – Ст. 64.
3. ДСТУ ISO/IEC 25010:2016. Інженерія систем і програмного забезпечення. Вимоги до якості систем і програмного забезпечення та оцінювання (SQuaRE). Моделі якості систем і програмного забезпечення (ISO/IEC 25010:2011, IDT). – К. : ДП «УкрНДНЦ», 2016. – 34 с.
4. ДСТУ ISO/IEC 12207:2016. Інженерія систем і програмного забезпечення. Процеси життєвого циклу програмного забезпечення. – К. : ДП «УкрНДНЦ», 2016. – 120 с.
5. Аллен Д. Як упорядкувати справи. Мистецтво продуктивності без стресу / Д. Аллен. – К. : КМ-БУКС, 2017. – 392 с.
6. Мартін Р. Чиста архітектура. Мистецтво розробки програмного забезпечення / Р. Мартін. – Фабула, 2019. – 368 с.
7. Фленаган Д. JavaScript. Повне керівництво / Д. Фленаган. – К. : Вільямс, 2021. – 720 с.
8. Larsen J. The Complete Guide to Modern JavaScript / J. Larsen. – New York : O'Reilly Media, 2020. – 450 p.
9. Соммервіль І. Інженерія програмного забезпечення / І. Соммервіль. – 10-те вид. – М. : Вільямс, 2018. – 850 с.
10. Купер А. Інтерфейс. Основи проєктування взаємодії / А. Купер, Р. Рейман. – СПб. : Символ-Плюс, 2018. – 600 с.
11. Макконелл С. Досконалий код: Майстер-клас / С. Макконелл. – М. : Русская редакция, 2017. – 896 с.
12. Фаулер М. Рефакторинг. Поліпшення проекту існуючого коду / М. Фаулер. – М. : Вільямс, 2019. – 448 с.

13. Simpson K. You Don't Know JS: Scope & Closures / K. Simpson. – O'Reilly Media, 2020. – 180 p.
14. Resig J. Secrets of the JavaScript Ninja / J. Resig, B. Bibeault. – Manning Publications, 2016. – 464 p.
15. Duckett J. JavaScript and JQuery: Interactive Front-End Web Development / J. Duckett. – Wiley, 2014. – 640 p.
16. Haverbeke M. Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming / M. Haverbeke. – No Starch Press, 2018. – 472 p.
17. Cirillo F. The Pomodoro Technique: The Acclaimed Time-Management System That Has Transformed How We Work / F. Cirillo. – Currency, 2018. – 160 p.
18. Covey S. R. The 7 Habits of Highly Effective People / S. R. Covey. – Simon & Schuster, 2020. – 464 p.
19. Tracy B. Eat That Frog!: 21 Great Ways to Stop Procrastinating and Get More Done in Less Time / B. Tracy. – Berrett-Koehler Publishers, 2017. – 144 p.
20. Rubin K. S. Essential Scrum: A Practical Guide to the Most Popular Agile Process / K. S. Rubin. – Addison-Wesley, 2012. – 500 p.
21. Гамма Е. Прийоми об'єктно-орієнтованого проектування. Патерни проектування / Е. Гамма, Р. Хелм. – К. : ВІЛЬЯМС, 2015. – 368 с.
22. Конноллі Т. Бази даних. Проектування, реалізація і супровід / Т. Конноллі, К. Бегг. – М. : Вільямс, 2017. – 1440 с.
23. MDN Web Docs. Web Storage API [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API.
24. MDN Web Docs. Document Object Model (DOM) [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model.
25. ISO/IEC 27001:2013 Information technology — Security techniques — Information security management systems — Requirements.

26. OWASP Top 10:2021. The Ten Most Critical Web Application Security Risks [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-top-ten/.
27. NIST SP 800-53. Security and Privacy Controls for Information Systems and Organizations. – National Institute of Standards and Technology, 2020.
28. Google Material Design Guidelines [Електронний ресурс]. – Режим доступу: https://m3.material.io/.
29. CSS Grid Layout Module Level 2. W3C Candidate Recommendation [Електронний ресурс]. – Режим доступу: https://www.w3.org/TR/css-grid-2/.
30. ECMAScript® 2023 Language Specification [Електронний ресурс]. – Режим доступу: https://tc39.es/ecma262/.
31. W3C Web Accessibility Initiative (WAI). Introduction to Web Accessibility [Електронний ресурс]. – Режим доступу: https://www.w3.org/WAI/fundamentals/.
32. Freeman E. Head First Design Patterns / E. Freeman, E. Robson. – O'Reilly Media, 2020. – 672 p.
33. Banks A. React Learning / A. Banks, E. Porcello. – O'Reilly Media, 2017. – 200 p.
34. Brown E. Web Development with Node and Express / E. Brown. – O'Reilly Media, 2019. – 320 p.
35. Kleppmann M. Designing Data-Intensive Applications / M. Kleppmann. – O'Reilly Media, 2017. – 616 p.
36. Nielsen J. Usability Engineering / J. Nielsen. – Morgan Kaufmann, 1993. – 362 p.
37. Norman D. The Design of Everyday Things / D. Norman. – Basic Books, 2013. – 368 p.
38. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability / S. Krug. – New Riders, 2014. – 216 p.

39. Zakas N. C. Professional JavaScript for Web Developers / N. C. Zakas. – Wrox, 2012. – 960 p.
40. Crockford D. JavaScript: The Good Parts / D. Crockford. – O'Reilly Media, 2008. – 176 p.
41. Osmani A. Learning JavaScript Design Patterns / A. Osmani. – O'Reilly Media, 2012. – 254 p.
42. Fowler M. Patterns of Enterprise Application Architecture / M. Fowler. – Addison-Wesley, 2002. – 560 p.
43. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software / E. Evans. – Addison-Wesley, 2003. – 560 p.
44. Beck K. Test Driven Development: By Example / K. Beck. – Addison-Wesley, 2002. – 240 p.
45. Humble J. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation / J. Humble, D. Farley. – Addison-Wesley, 2010. – 512 p.
46. Kim G. The DevOps Handbook / G. Kim, J. Humble, P. Debois. – IT Revolution Press, 2016. – 480 p.
47. Newman S. Building Microservices / S. Newman. – O'Reilly Media, 2015. – 280 p.
48. Richards M. Fundamentals of Software Architecture / M. Richards, N. Ford. – O'Reilly Media, 2020. – 432 p.
49. Burns B. Kubernetes: Up and Running / B. Burns, J. Beda, K. Hightower. – O'Reilly Media, 2019. – 268 p.
50. Chacon S. Pro Git / S. Chacon, B. Straub. – Apress, 2014. – 456 p.

ДОДАТОК А

Лістинги програмного коду

А.1. Файл index.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Task Intelligence System</title>
    <link
href="https://fonts.googleapis.com/css2?family=Manrope:wght@400;600;700&display=swap"
rel="stylesheet">
    <link rel="stylesheet" href="style.css">
</head>
<body>

<div id="login-container">
    <h2>Вхід в систему</h2>

    <label>Введіть ім'я:</label>
    <input id="username" type="text">

    <label>Оберіть роль:</label>
    <select id="role">
        <option value="member">Працівник</option>
        <option value="lead">Тім Лід</option>
    </select>

    <button onclick="login()">Увійти</button>
</div>


<div id="app" class="hidden">

    <div class="top-panel">
        <h2 id="welcome"></h2>
        <button class="logout-btn" onclick="logout()">Вийти</button>
    </div>

    <div class="main-layout">
        <div class="main-layout">
            <section class="left-column">
                <div id="calendar-panel" class="card hidden">
                    <h3>Календар</h3>
                    <div id="calendar-header"></div>
                    <div id="calendar-grid"></div>
                </div>
                <div id="calendar-day-tasks" class="card hidden">
                    <h3 id="calendar-day-title">Задачі на день</h3>
                    <div id="calendar-day-list"></div>
                </div>
            </section>
            <section class="center-column">
                <div id="task-creator" class="card hidden">
                    <h3>Створити задачу</h3>
                    <input id="task-title" placeholder="Назва задачі">
                    <div class="task-row">
                        <input id="task-deadline" type="date">
                        <select id="task-priority">
                            <option value="low">Низький</option>
                            <option value="medium">Середній</option>
                            <option value="high">Високий</option>
                        </select>
                    </div>
                    <div id="assign-block" class="hidden">
```

```
                        <label>Призначити на:</label>
                        <select id="assign-user"></select>
                    </div>
                    <button onclick="addTask()">Додати задачу</button>
                </div>
                <div class="card">
                    <div class="task-list-header">
                        <h3>Список задач (потрібно виконати)</h3>
                        <div class="task-filters">
                            <label>Пріоритет:</label>
                            <select id="priority-filter" onchange="renderTasks()">
                                <option value="all">Всі</option>
                                <option value="high">Високий</option>
                                <option value="medium">Середній</option>
                                <option value="low">Низький</option>
                            </select>
                        </div>
                    </div>
                    <div id="task-list"></div>
                </div>
            </section>
            <section class="right-column">
                <div id="stats-panel" class="card hidden">
                    <h3>Статистика по людям</h3>
                    <div id="stats-list"></div>
                    <div style="margin-top:12px">
                        <button id="lead-stats-btn" onclick="openLeadStats()">Оцінка
ефективності</button>
                    </div>
                    <div id="user-detail" class="user-detail hidden"></div>
                </div>
                <div id="completed-panel" class="card">
                    <h3>Виконані / Не виконані задачі</h3>
                    <div id="completed-list"></div>
                </div>
            </section>
        </div>
    </div>

</div>

<script src="script.js"></script>
</body>
</html>
</body>
</html>
```

A.2. Файл style.css

```
body {
    font-family: 'Manrope', Arial, sans-serif;
    background: linear-gradient(135deg, #f5f8fc 0%, #e8eef8 25%, #e0e7f3 50%, #dfe7f3 75%,
#e8ecf8 100%);
    margin: 0;
    padding: 0;
    min-height: 100vh;
    letter-spacing: 0.3px;
    color: #1a1d29;
    line-height: 1.6;
    position: relative;
    overflow-x: hidden;
    background-attachment: fixed;
}

/* subtle animated light blobs behind content (GPU-accelerated) */
body::before {
    content: '';
```

```
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background:
        radial-gradient(circle at 20% 50%, rgba(74, 115, 243, 0.08) 0%, transparent 50%),
        radial-gradient(circle at 80% 80%, rgba(102, 126, 234, 0.06) 0%, transparent 50%),
        radial-gradient(circle at 40% 20%, rgba(118, 75, 162, 0.05) 0%, transparent 50%);
    pointer-events: none;
    z-index: 0; /* keep in normal stacking, put #app above */
    will-change: transform;
    transform: translate3d(0,0,0);
    animation: float 20s linear infinite;
}

/* ensure main app is above the animated layer */
#app { position: relative; z-index: 1; }

@keyframes float {
    0%, 100% { transform: translate3d(0, 0, 0); }
    25%      { transform: translate3d(-14px, -8px, 0); }
    50%      { transform: translate3d(-28px, -16px, 0); }
    75%      { transform: translate3d(-14px, -8px, 0); }
}

/* respect user's reduced motion preference */
@media (prefers-reduced-motion: reduce) {
    body::before { animation: none; }
}

* {
    box-sizing: border-box;
}

h1, h2, h3, h4, h5, h6 {
    margin: 0;
    line-height: 1.2;
}

p {
    margin: 0;
}

.hidden { display: none; }

#app {
    animation: fadeIn 0.5s ease-out;
}

#welcome {
    font-size: 24px;
    font-weight: 700;
    background: linear-gradient(135deg, #f4f7ff 0%, #f6f8ff 100%);
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
    background-clip: text;
    letter-spacing: -0.5px;
}

#login-container {
    max-width: 920px;
    margin: 20px auto;
    padding: 32px;
    background: linear-gradient(135deg, #ffffff 0%, #f8fafc 100%);
    border-radius: 20px;
    box-shadow: 0 20px 60px rgba(0,0,0,0.08), 0 2px 12px rgba(0,0,0,0.04);
    border: 1px solid rgba(63, 73, 99, 0.06);
    position: relative;
    overflow: hidden;
```

```
}

#login-container::before {
    content: '';
    position: absolute;
    top: -50%;
    right: -50%;
    width: 400px;
    height: 400px;
    background: radial-gradient(circle, rgba(74, 115, 243, 0.08) 0%, transparent 70%);
    border-radius: 50%;
    pointer-events: none;
}

/* Cards use dark background and white text per request */
.card {
    max-width: 920px;
    margin: 20px auto;
    padding: 28px;
    background: linear-gradient(135deg, #2d3748 0%, #374152 100%);
    color: #ffffff;
    border-radius: 18px;
    box-shadow: 0 20px 50px rgba(0,0,0,0.15), 0 4px 15px rgba(0,0,0,0.08);
    border: 1px solid rgba(255, 255, 255, 0.05);
    position: relative;
    overflow: hidden;
}

.card::before {
    content: '';
    position: absolute;
    top: 0;
    right: 0;
    width: 200px;
    height: 200px;
    background: radial-gradient(circle, rgba(74, 115, 243, 0.1) 0%, transparent 70%);
    border-radius: 50%;
    pointer-events: none;
}

.card h3 {
    margin-top: 0;
    font-size: 24px;
    font-weight: 700;
    letter-spacing: -0.5px;
    position: relative;
    z-index: 1;
    margin-bottom: 20px;
}

#login-container h2 {
    text-align: center;
    color: #1a1d29;
    font-size: 28px;
    margin-bottom: 24px;
    background: linear-gradient(135deg, #4a73f3 0%, #667eea 100%);
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
    background-clip: text;
}

#task-creator h3 {
    color: #ffffff;
    text-align: center;
}

#task-creator {
    display: flex;
    flex-direction: column;
}
```

```
.task-row {
    display: flex;
    gap: 12px;
    width: 100%;
    margin: 10px 0;
}

.task-row input,
.task-row select {
    flex: 1;
    margin: 0;
    width: auto;
}

#assign-block {
    display: flex;
    flex-direction: column !important;
    gap: 0 !important;
}

/* Inputs inside dark cards should be readable */
.card input, .card select {
    background: #ffffff;
    color: #1a1d29;
    border-radius: 8px;
    padding: 12px 14px;
    border: 2px solid rgba(0,0,0,0.06);
    font-size: 15px;
    font-family: 'Manrope', Arial, sans-serif;
    transition: all 0.3s ease;
    position: relative;
    z-index: 1;
}

label {
    display: block;
    margin: 12px 0 6px 0;
    font-weight: 600;
    font-size: 14px;
    color: #1a1d29;
    letter-spacing: 0.2px;
}

.card label {
    color: #ffffff;
    margin-top: 14px;
    margin-bottom: 8px;
}

.card input:focus, .card select:focus {
    border-color: #4a73f3;
    box-shadow: 0 0 0 3px rgba(74, 115, 243, 0.1);
    background: #ffffff;
}

/* Constrain overall app width to full screen */
.main-layout { width: 100vw; margin: 0; padding: 20px 0; }

input, select, button {
    width: 100%;
    padding: 12px 16px;
    margin: 10px 0;
    font-family: 'Manrope', Arial, sans-serif;
    transition: all 0.3s ease;
}

input, select {
    border: 2px solid rgba(0,0,0,0.06);
    border-radius: 8px;
```

```
    font-size: 15px;
    background: white;
    color: #1a1d29;
}

input:hover, select:hover {
    border-color: rgba(74, 115, 243, 0.3);
}

button {
    cursor: pointer;
    background: linear-gradient(135deg, #4a73f3 0%, #3d5ec9 100%);
    border: none;
    color: white;
    border-radius: 8px;
    font-weight: 600;
    font-size: 15px;
    letter-spacing: 0.3px;
    transition: all 0.3s ease;
    position: relative;
    overflow: hidden;
}

button::before {
    content: '';
    position: absolute;
    top: 0;
    left: -100%;
    width: 100%;
    height: 100%;
    background: linear-gradient(90deg, transparent, rgba(255,255,255,0.3), transparent);
    transition: left 0.5s ease;
}

button:hover::before {
    left: 100%;
}

button:hover {
    transform: translateY(-2px);
    box-shadow: 0 12px 32px rgba(74, 115, 243, 0.4);
}

button:active {
    transform: translateY(0);
    box-shadow: 0 6px 16px rgba(74, 115, 243, 0.3);
}

.task-list-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 16px;
    flex-wrap: wrap;
    gap: 12px;
}

.task-list-header h3 {
    margin: 0;
}

.task-filters {
    display: flex;
    align-items: center;
    gap: 8px;
}

.task-filters label {
    font-size: 14px;
    color: rgba(255, 255, 255, 0.9);
```

```
    margin: 0;
}

.task-filters select {
    padding: 6px 12px;
    border-radius: 6px;
    border: 1px solid rgba(255, 255, 255, 0.2);
    background: #ffffff;
    color: #1a1d29;
    font-size: 14px;
    cursor: pointer;
    transition: all 0.2s ease;
}

.task-filters select option {
    color: #1a1d29;
    background: #ffffff;
}

.task-filters select:hover {
    background: rgba(255, 255, 255, 0.15);
    border-color: rgba(74, 115, 243, 0.5);
}

.task-filters select:focus {
    outline: none;
    border-color: #4a73f3;
    box-shadow: 0 0 0 2px rgba(74, 115, 243, 0.2);
}

#task-list {
    display: flex;
    flex-direction: column;
    gap: 16px;
    padding: 8px;
    width: 100%;
}

.task {
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    color: #ffffff;
    border-radius: 12px;
    padding: 12px; /* reduced padding to make cards smaller */
    box-shadow: 0 8px 24px rgba(102, 126, 234, 0.22), 0 1px 6px rgba(0,0,0,0.06);
    display: flex;
    flex-direction: column;
    justify-content: space-between;
    min-height: 110px; /* smaller height so more cards fit vertically */
    box-sizing: border-box;
    word-wrap: break-word;
    overflow-wrap: break-word;
    margin-bottom: 8px;
    transition: all 0.22s ease;
    border: 1px solid rgba(255, 255, 255, 0.08);
    position: relative;
    overflow: hidden; /* clip animated highlights inside card */
    animation: fadeIn 0.34s ease-out;
}

.task::before {
    content: '';
    position: absolute;
    top: 0;
    left: -100%;
    width: 100%;
    height: 100%;
    background: linear-gradient(90deg, transparent, rgba(255,255,255,0.2), transparent);
    transition: left 0.5s ease;
}
```

```
.task:hover::before {
    left: 100%;
}

.task:hover {
    transform: translateY(-4px);
    box-shadow: 0 15px 40px rgba(102, 126, 234, 0.35), 0 4px 12px rgba(0,0,0,0.12);
}

.task::after {
    content: '';
    position: absolute;
    top: -50%;
    right: -50%;
    width: 100px;
    height: 100px;
    background: radial-gradient(circle, rgba(255,255,255,0.1) 0%, transparent 70%);
    border-radius: 50%;
    pointer-events: none;
}

.task:nth-child(1) { animation-delay: 0ms; }
.task:nth-child(2) { animation-delay: 50ms; }
.task:nth-child(3) { animation-delay: 100ms; }
.task:nth-child(4) { animation-delay: 150ms; }
.task:nth-child(n+5) { animation-delay: 200ms; }

/* Task date section: wrapper containing header + group */
.task-date-section {
    display: flex;
    flex-direction: column;
    margin-bottom: 20px;
    width: 100%;
}

/* Task date header label (small label in top-left) */
.task-date-header {
    font-size: 12px;
    font-weight: 700;
    color: #ffffff;
    background: linear-gradient(135deg, #4a73f3 0%, #3d5ec9 100%);
    padding: 4px 8px;
    border-radius: 6px;
    letter-spacing: 0.3px;
    display: inline-block;
    margin-bottom: 8px;
    margin-left: 0;
    margin-top: 0;
    box-shadow: 0 2px 6px rgba(74, 115, 243, 0.25);
    width: fit-content;
}

/* Wrapper for each date group - creates row layout for tasks under date label */
.task-date-group {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(220px, 1fr));
    gap: 12px;
    width: 100%;
}



/* New task layout: centered title/meta and buttons at the bottom */
.task-body {
    text-align: center;
    position: relative;
    z-index: 1;
}

.task-title {
```

```
    font-weight: 700;
    margin-bottom: 6px;
    font-size: 16px;
    letter-spacing: -0.2px;
    text-shadow: 0 1px 6px rgba(0,0,0,0.08);
}
.task-meta {
    font-size: 13px;
    color: rgba(255, 255, 255, 0.86);
    margin-bottom: 6px;
    line-height: 1.3;
}

.task-actions {
    display: flex;
    gap: 8px;
    margin-top: 10px;
}

.task-actions button {
    flex: 1;
    background: rgba(255, 255, 255, 0.14);
    color: #ffffff;
    border: 1px solid rgba(255, 255, 255, 0.22);
    padding: 8px;
    border-radius: 8px;
    cursor: pointer;
    transition: all 0.22s ease;
    font-weight: 600;
    font-size: 12px;
    position: relative;
    overflow: hidden;
}

.task-actions button::before {
    content: '';
    position: absolute;
    top: 0;
    left: -100%;
    width: 100%;
    height: 100%;
    background: linear-gradient(90deg, transparent, rgba(255,255,255,0.2), transparent);
    transition: left 0.4s ease;
}

.task-actions button:hover::before {
    left: 100%;
}

.task-actions button:hover {
    background: rgba(255, 255, 255, 0.25);
    transform: translateY(-2px);
    box-shadow: 0 6px 16px rgba(0,0,0,0.15);
    border-color: rgba(255, 255, 255, 0.35);
}

.task-actions button:active {
    transform: translateY(0);
}

.done {
    color: #10b981;
    font-weight: 700;
    background: rgba(16, 185, 129, 0.1);
    padding: 4px 10px;
    border-radius: 6px;
    display: inline-block;
    font-size: 13px;
}
```

```
.status-new {
    color: #cbb8f8;
    font-weight: 700;
    background: rgba(139, 92, 246, 0.1);
    padding: 4px 10px;
    border-radius: 6px;
    display: inline-block;
    font-size: 13px;
}

.failed {
    color: #ef4444;
    font-weight: 700;
    background: rgba(255, 226, 226, 0.562);
    padding: 4px 10px;
    border-radius: 6px;
    display: inline-block;
    font-size: 13px;
}



.status-btn {
    background: linear-gradient(135deg, #10b981 0%, #059669 100%) !important;
    padding: 10px 12px !important;
    border-radius: 8px !important;
    border: 1px solid rgba(0,0,0,0.1) !important;
    color: white !important;
    transition: all 0.3s ease !important;
    font-weight: 600 !important;
    font-size: 14px !important;
    position: relative !important;
    overflow: hidden !important;
}

.status-btn::before {
    content: '';
    position: absolute;
    top: 50%;
    left: 50%;
    width: 0;
    height: 0;
    background: rgba(255,255,255,0.3);
    border-radius: 50%;
    transform: translate(-50%, -50%);
    transition: width 0.5s ease, height 0.5s ease;
}

.status-btn:hover::before {
    width: 300px;
    height: 300px;
}

.status-btn:hover {
    background: linear-gradient(135deg, #059669 0%, #047857 100%) !important;
    transform: translateY(-2px) !important;
    box-shadow: 0 8px 20px rgba(16, 185, 129, 0.4) !important;
}

.undo-btn {
    background: linear-gradient(135deg, #6b7280 0%, #4b5563 100%);
    color: white;
    padding: 10px 12px;
    border-radius: 8px;
    border: 1px solid rgba(255,255,255,0.1);
    transition: all 0.3s ease;
    font-weight: 600;
    position: relative;
    overflow: hidden;
}
```

```
.undo-btn::before {
    content: '';
    position: absolute;
    top: 50%;
    left: 50%;
    width: 0;
    height: 0;
    background: rgba(255,255,255,0.2);
    border-radius: 50%;
    transform: translate(-50%, -50%);
    transition: width 0.5s ease, height 0.5s ease;
}

.undo-btn:hover::before {
    width: 300px;
    height: 300px;
}

.undo-btn:hover {
    background: linear-gradient(135deg, #4b5563 0%, #374152 100%);
    color: white;
    transform: translateY(-2px);
    box-shadow: 0 8px 20px rgba(0,0,0,0.25);
}


.stat-row {
    display: flex;
    flex-direction: column;
    padding: 16px 12px;
    border-bottom: 1px solid rgba(255,255,255,0.08);
    gap: 8px;
    transition: all 0.3s ease;
    border-radius: 10px;
    position: relative;
}

.stat-row:last-child { border-bottom: none; }

.stat-name {
    font-weight: 700;
    font-size: 16px;
    background: linear-gradient(135deg, #ffffff 0%, rgba(255,255,255,0.9) 100%);
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
    background-clip: text;
}

.stat-values {
    color: rgba(255,255,255,0.88);
    font-size: 14px;
    line-height: 1.6;
}

/* Clickable stat row */
.stat-click {
    cursor: pointer;
    padding-left: 0;
    transition: all 0.3s ease;
    border-radius: 10px;
}

.stat-click:hover {
    background: linear-gradient(135deg, rgba(74, 115, 243, 0.2) 0%, rgba(102, 126, 234,
0.15) 100%);
    padding-left: 12px;
    border-left: 4px solid #4a73f3;
    box-shadow: inset 0 0 20px rgba(74, 115, 243, 0.1);
}
```

```
/* user detail panel */
.user-detail {
    margin-top: 16px;
    padding: 18px;
    background: linear-gradient(135deg, rgba(74, 115, 243, 0.12) 0%, rgba(118, 75, 162,
0.1) 100%);
    border-radius: 14px;
    border: 1px solid rgba(74, 115, 243, 0.2);
    box-shadow: 0 8px 24px rgba(74, 115, 243, 0.1);
    backdrop-filter: blur(10px);
}

.user-detail h4 {
    margin: 0 0 16px 0;
    font-size: 17px;
    font-weight: 700;
    color: #ffffff;
    background: linear-gradient(135deg, #4a73f3 0%, #764ba2 100%);
    -webkit-background-clip: text;
    -webkit-text-fill-color: transparent;
    background-clip: text;
}

/* small variant for tasks inside detail panel */
.task.small {
    min-height: 70px;
    padding: 12px;
    background: linear-gradient(135deg, rgba(255,255,255,0.1) 0%, rgba(255,255,255,0.06)
100%);
    margin-bottom: 10px;
    border-radius: 10px;
    border: 1px solid rgba(255, 255, 255, 0.15);
    transition: all 0.3s ease;
}

.task.small:hover {
    background: linear-gradient(135deg, rgba(255,255,255,0.15) 0%, rgba(255,255,255,0.1)
100%);
    transform: translateX(4px);
    box-shadow: 0 4px 12px rgba(0,0,0,0.15);
}

.fail-btn {
    background: linear-gradient(135deg, #ef4444 0%, #dc2626 100%);
    color: white;
    padding: 10px 12px;
    border-radius: 8px;
    border: 1px solid rgba(0,0,0,0.1);
    transition: all 0.3s ease;
    font-weight: 600;
    position: relative;
    overflow: hidden;
}

.fail-btn::before {
    content: '';
    position: absolute;
    top: 50%;
    left: 50%;
    width: 0;
    height: 0;
    background: rgba(255,255,255,0.3);
    border-radius: 50%;
    transform: translate(-50%, -50%);
    transition: width 0.5s ease, height 0.5s ease;
}

.fail-btn:hover::before {
    width: 300px;
```

```
    height: 300px;
}

.fail-btn:hover {
    background: linear-gradient(135deg, #dc2626 0%, #b91c1c 100%);
    color: white;
    transform: translateY(-2px);
    box-shadow: 0 8px 20px rgba(239, 68, 68, 0.4);
}

/* Delete button for task card */
.delete-btn {
    position: absolute;
    top: 10px; /* slightly tighter to match smaller size */
    right: 10px;
    width: 24px;
    height: 24px;
    padding: 0 !important;
    margin: 0 !important;
    background: linear-gradient(135deg, #ef4444 0%, #dc2626 100%);
    color: white;
    border: none;
    border-radius: 50%;
    cursor: pointer;
    font-size: 12px;
    font-weight: 700;
    transition: all 0.22s ease;
    display: flex;
    align-items: center;
    justify-content: center;
    z-index: 10;
    line-height: 1;
}

.delete-btn:hover {
    background: linear-gradient(135deg, #dc2626 0%, #b91c1c 100%);
    transform: scale(1.06) rotate(90deg);
    box-shadow: 0 6px 14px rgba(239, 68, 68, 0.45);
}

.delete-btn:active {
    transform: scale(0.94) rotate(90deg);
}

/* Calendar styles */
.calendar-nav {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 12px;
    gap: 12px;
}

.calendar-title {
    font-size: 16px;
    font-weight: 700;
    color: #ffffff;
    text-align: center;
    letter-spacing: 0.3px;
    flex: 1;
}

.calendar-nav-btn {
    background: linear-gradient(135deg, rgba(74, 115, 243, 0.3) 0%, rgba(102, 126, 234,
0.25) 100%);
    border: 1px solid rgba(74, 115, 243, 0.4);
    color: #ffffff;
    width: 36px;
    height: 36px;
    border-radius: 8px;
```

```
    cursor: pointer;
    font-size: 18px;
    font-weight: 700;
    transition: all 0.2s ease;
    display: flex;
    align-items: center;
    justify-content: center;
    padding: 0;
}

.calendar-nav-btn:hover {
    background: linear-gradient(135deg, rgba(74, 115, 243, 0.4) 0%, rgba(102, 126, 234,
0.35) 100%);
    border-color: rgba(74, 115, 243, 0.6);
    transform: scale(1.05);
}

.calendar-nav-btn:active {
    transform: scale(0.95);
}

.calendar-weekdays {
    display: grid;
    grid-template-columns: repeat(7, 1fr);
    gap: 6px;
    margin-bottom: 8px;
}

.calendar-weekday {
    font-size: 12px;
    font-weight: 700;
    color: rgba(255, 255, 255, 0.7);
    text-align: center;
    padding: 4px 0;
}

.calendar-days {
    display: grid;
    grid-template-columns: repeat(7, 1fr);
    gap: 6px;
}

.calendar-day {
    background: linear-gradient(135deg, rgba(102, 126, 234, 0.2) 0%, rgba(118, 75, 162,
0.15) 100%);
    border: 1px solid rgba(102, 126, 234, 0.25);
    border-radius: 8px;
    padding: 6px;
    height: 80px;
    max-height: 80px;
    display: flex;
    flex-direction: column;
    font-size: 11px;
    color: #ffffff;
    position: relative;
    transition: all 0.2s ease;
    cursor: pointer;
    overflow: hidden;
}

.calendar-day:not(.empty):hover {
    background: linear-gradient(135deg, rgba(102, 126, 234, 0.3) 0%, rgba(118, 75, 162,
0.25) 100%);
    box-shadow: 0 4px 12px rgba(102, 126, 234, 0.2);
    transform: translateY(-2px);
}

.calendar-day.empty {
    background: transparent;
    border: none;
```

```
}

.calendar-day.weekend {
    background: linear-gradient(135deg, rgba(239, 68, 68, 0.15) 0%, rgba(220, 38, 38, 0.1)
100%);
    border: 1px solid rgba(239, 68, 68, 0.2);
}

.calendar-day.weekend:not(.empty):hover {
    background: linear-gradient(135deg, rgba(239, 68, 68, 0.25) 0%, rgba(220, 38, 38, 0.2)
100%);
}

.calendar-day.today {
    background: linear-gradient(135deg, rgba(74, 115, 243, 0.4) 0%, rgba(102, 126, 234,
0.3) 100%);
    border: 2px solid #4a73f3;
    box-shadow: 0 0 12px rgba(74, 115, 243, 0.3);
}

.cal-day-number {
    font-weight: 700;
    font-size: 12px;
    margin-bottom: 4px;
    color: #ffffff;
}

.cal-day-tasks {
    display: flex;
    flex-direction: column;
    gap: 2px;
    flex: 1;
    overflow: hidden;
}

.cal-task-item {
    background: rgba(255, 255, 255, 0.15);
    padding: 2px 4px;
    border-radius: 3px;
    font-size: 10px;
    color: #e6eefc;
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
    max-width: 100%;
    display: block;
}

.cal-task-more {
    font-size: 10px;
    color: #a0b9ff;
    font-weight: 600;
    padding: 2px 4px;
}

.cal-no-tasks {
    flex: 1;
}

/* Calendar day tasks panel */
#calendar-day-tasks {
    margin-top: 15px;
}

#calendar-day-list .task {
    margin-bottom: 10px;
}

/* Layout for main + sidebar */
.main-layout {
```

```
    display: grid;
    grid-template-columns: 1fr 2fr 1fr;
    gap: 15px;
    align-items: flex-start;
    padding: 0px 20px 20px 20px;
    box-sizing: border-box;
    animation: fadeIn 0.5s ease;
    width: 100vw;
    max-width: 100vw;
    margin: 0;
}


.left-column {
    min-width: 0;
}
.center-column {
    min-width: 0;
    width: 100%;
}
.right-column {
    min-width: 0;
    padding-right: 50px;
}

/* Limit completed tasks list height and enable scrolling */
#completed-list {
    max-height: 360px;
    overflow-y: auto;
    overflow-x: hidden;
    padding-right: 6px;
}

@media (max-width: 900px) {
    .main-layout {
        flex-direction: column;
        gap: 16px;
    }

    .side-column {
        width: 100%;
    }

    #login-container {
        width: 90%;
        max-width: 420px;
    }

    .task {
        min-height: 120px;
    }
}

/* Ensure inputs/buttons don't overflow */
input:focus, select:focus {
    outline: none;
    box-shadow: 0 0 0 4px rgba(74, 115, 243, 0.15);
    border-color: #4a73f3;
    border-width: 2px;
}

/* Center the login form on the viewport */
#login-container {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    width: 420px;
    animation: slideUp 0.6s cubic-bezier(0.34, 1.56, 0.64, 1);
}
```

```
@keyframes slideUp {
    from {
        opacity: 0;
        transform: translate(-50%, -40%);
        filter: blur(20px);
    }
    to {
        opacity: 1;
        transform: translate(-50%, -50%);
        filter: blur(0);
    }
}

/* Make logout button small and inline in the top panel */
.top-panel .logout-btn {
    display: inline-block;
    padding: 10px 24px;
    font-size: 14px;
    background: linear-gradient(135deg, #ef4444 0%, #dc2626 100%);
    color: #fff;
    border-radius: 8px;
    border: none;
    cursor: pointer;
    align-self: center;
    width: auto !important;
    max-width: 240px;
    font-weight: 600;
    transition: all 0.3s ease;
}

.top-panel .logout-btn:hover {
    transform: translateY(-2px);
    box-shadow: 0 8px 20px rgba(239, 68, 68, 0.4);
}

/* Lead stats button (in stats panel) */
#lead-stats-btn {
    width: 100%;
    padding: 10px 12px;
    font-size: 14px;
    border-radius: 8px;
    background: linear-gradient(135deg, #6b8cff 0%, #546ef0 100%);
    color: white;
    border: none;
    cursor: pointer;
    font-weight: 700;
    transition: all 0.24s ease;
}

#lead-stats-btn:hover {
    transform: translateY(-2px);
    box-shadow: 0 10px 30px rgba(84,110,240,0.28);
}

.lead-stats-expl {
    font-size: 13px;
    color: #e6eefc;
    background: linear-gradient(180deg, rgba(84,110,240,0.08), rgba(84,110,240,0.04));
    padding: 10px 12px;
    border-radius: 8px;
    border: 1px solid rgba(84,110,240,0.06);
    line-height: 1.35;
}

/* Modal for lead stats */
.modal-backdrop {
    position: fixed;
    top: 0;
    left: 0;
```

```
    width: 100%;
    height: 100%;
    background: rgba(0,0,0,0.45);
    display: flex;
    align-items: center;
    justify-content: center;
    z-index: 2000;
}

.modal {
    background: linear-gradient(180deg, #ffffff 0%, #f3f6ff 100%);
    border-radius: 12px;
    width: 92%;
    max-width: 920px;
    padding: 18px;
    box-shadow: 0 30px 80px rgba(21, 41, 82, 0.35);
    border: 1px solid rgba(84,110,240,0.12);
}

.modal h3 { margin-top: 0; }

.modal .close-btn {
    position: absolute;
    top: 12px;
    right: 12px;
    width: 28px;
    height: 28px;
    background: linear-gradient(135deg, #ef4444 0%, #dc2626 100%);
    border: none;
    color: #fff;
    font-size: 12px;
    font-weight: 700;
    border-radius: 50%;
    display: flex;
    align-items: center;
    justify-content: center;
    cursor: pointer;
    box-shadow: 0 6px 16px rgba(220,36,36,0.28);
    transition: transform 0.18s ease, box-shadow 0.18s ease;
    z-index: 2100;
}

.modal .close-btn:hover {
    transform: scale(1.08);
    box-shadow: 0 10px 22px rgba(220,36,36,0.34);
}

.lead-stats-table { width: 100%; border-collapse: collapse; margin-top: 12px; }
.lead-stats-table th, .lead-stats-table td { padding: 8px 10px; text-align: left; border-
bottom: 1px solid rgba(0,0,0,0.06); }
.lead-stats-table th { font-weight: 700; font-size: 13px; color: #1a2340; }
.lead-stats-table td { font-size: 13px; color: #1f2a44; }

.modal-actions { display:flex; gap:8px; margin-top:12px; }
.modal-actions button { padding: 10px 12px; border-radius:8px; font-weight:700;
cursor:pointer }
.send-email-btn { background: linear-gradient(135deg,#10b981 0%,#059669 100%); color:white;
border:none }
.close-modal-btn { background: linear-gradient(135deg,#6b7280 0%,#4b5563 100%);
color:white; border:none }

/* Ensure specific form buttons remain full width (keep task action buttons flexible) */
#task-creator button, #login-container button, .card .fullwidth { width: 100%; }

.top-panel {
    display: flex;
    justify-content: space-between;
    align-items: center;
    width: calc(100vw - 50px);
    max-width: calc(100vw - 90px);
```

```
    padding: 20px 28px 20px 28px;
    margin: 20px 40px 0px 40px;
    background: linear-gradient(135deg, #2d3748 0%, #374152 100%);
    border: 1px solid rgba(74, 115, 243, 0.3);
    border-radius: 12px;
    box-shadow: 0 4px 12px rgba(0,0,0,0.1);
    animation: slideDown 0.5s ease;
}

@keyframes slideDown {
    from {
        opacity: 0;
        transform: translateY(-10px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}

/* Animations for card loading */
@keyframes fadeIn {
    from {
        opacity: 0;
        transform: scale(0.95);
    }
    to {
        opacity: 1;
        transform: scale(1);
    }
}

/* Scrollbar styling */
::-webkit-scrollbar {
    width: 10px;
    height: 10px;
}

::-webkit-scrollbar-track {
    background: rgba(0,0,0,0.05);
}

::-webkit-scrollbar-thumb {
    background: linear-gradient(135deg, #4a73f3 0%, #667eea 100%);
    border-radius: 10px;
}

::-webkit-scrollbar-thumb:hover {
    background: linear-gradient(135deg, #3d5ec9 0%, #5a6bcb 100%);
}
```

A.3. Файл script.js

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

// Map internal role values to localized labels
function roleLabel(role) {
    if (role === 'lead') return 'Тім Лід';
    if (role === 'member') return 'Працівник';
    return role;
}

// Helpers for persistence
function saveUsers() {
    localStorage.setItem("users", JSON.stringify(users));
}
```

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

// ---------------------- LOGIN ----------------------

function login() {
    let name = document.getElementById("username").value;
    let role = document.getElementById("role").value;

    if (!name) return alert("Введіть ім'я!");

    currentUser = { name, role };

    // Add to users if not present (unique by name)
    let exists = users.find(u => u.name === name);
    if (!exists) {
        users.push(currentUser);
        saveUsers();
    } else if (exists.role !== role) {
        // update role if changed
        exists.role = role;
        saveUsers();
    }

    document.getElementById("login-container").classList.add("hidden");
    document.getElementById("app").classList.remove("hidden");

    document.getElementById("welcome").innerText =
        `Вітаю, ${name}! Ваша роль: ${roleLabel(role)}`;

    prepareRoleView();
    renderTasks();
}

// ---------------------- LOGOUT ----------------------
function logout() {
    currentUser = null;
    location.reload();
}

// ---------------------- ROLE LOGIC ----------------------
function prepareRoleView() {
    let creator = document.getElementById("task-creator");
    let assignBlock = document.getElementById("assign-block");
    if (currentUser.role === "lead") {
        creator.classList.remove("hidden");
        assignBlock.classList.remove("hidden");
        assignBlock.style.display = '';

        // заповнити список учасників
        let select = document.getElementById("assign-user");
        select.innerHTML = "";
        users.forEach(u => {
            // allow assigning to anyone except a lead
            if (u.name && u.role !== 'lead')
                select.innerHTML += `<option value="${u.name}">${u.name} —
${roleLabel(u.role)}</option>`;
        });
        // ensure select is enabled for leads
        if (select) select.disabled = false;

        // show calendar and stats for lead
        document.getElementById("calendar-panel").classList.remove("hidden");
        renderCalendar();

        document.getElementById("stats-panel").classList.remove("hidden");
        renderStats();

    } else if (currentUser.role === "member") {
```

```
        creator.classList.remove("hidden");
        // hide and disable assign controls for members so they cannot assign tasks to
others
        assignBlock.classList.add("hidden");
        assignBlock.style.display = 'none';
        let select = document.getElementById("assign-user");
        if (select) {
            select.disabled = true;
            select.innerHTML = ""; // clear any options to avoid confusion
        }

        // show calendar for members too
        document.getElementById("calendar-panel").classList.remove("hidden");
        renderCalendar();

        document.getElementById("stats-panel").classList.add("hidden");
    }
}

// ---------------------- ADD TASK ----------------------
function addTask() {
    let title = document.getElementById("task-title").value;
    let deadline = document.getElementById("task-deadline").value;
    let priority = document.getElementById("task-priority").value;

    if (!title || !deadline) return alert("Заповніть всі поля!");

    // Members can only assign tasks to themselves. Leads may choose assignee from the
select.
    let assignedTo = currentUser.name;
    if (currentUser.role === "lead") {
        const sel = document.getElementById("assign-user");
        if (sel && sel.value) assignedTo = sel.value;
    }

    let task = {
        id: Date.now(),
        title,
        deadline,
        priority,
        assignedTo,
        createdBy: currentUser.name,
        status: "new"
    };

    tasks.push(task);
    saveTasks();

    // clear inputs
    document.getElementById("task-title").value = "";
    document.getElementById("task-deadline").value = "";
    document.getElementById("task-priority").value = "low";

    // update assign list (in case a new user was added) and views
    prepareRoleView();
    renderTasks();
}

function renderTasks() {
    let list = document.getElementById("task-list");
    // Render active (not-done) tasks in center, and completed in sidebar
    list.innerHTML = "";
    let completedList = document.getElementById('completed-list');
    completedList.innerHTML = "";

    // Get priority filter value
    const priorityFilter = document.getElementById('priority-filter')?.value || 'all';

    // Filter and group active tasks by deadline
    const activeTasks = [];
```

```
    const completedTasks = [];

    tasks.forEach((task, index) => {
        // visibility: lead sees all, members see only their own
        let visibleToUser = currentUser.role === 'lead' || task.assignedTo ===
currentUser.name;
        if (!visibleToUser) return;

        // if statsFilterUser is set (lead clicked a user), only show tasks for that user
in the main area
        if (statsFilterUser && currentUser.role === 'lead' && task.assignedTo !==
statsFilterUser) return;

        // Apply priority filter
        if (priorityFilter !== 'all' && task.priority !== priorityFilter) return;

        if (task.status === 'new') {
            activeTasks.push({ task, index });
        } else {
            completedTasks.push({ task, index });
        }
    });

    // Sort active tasks by deadline
    activeTasks.sort((a, b) => new Date(a.task.deadline) - new Date(b.task.deadline));

    // Group active tasks by deadline
    const tasksByDate = {};
    activeTasks.forEach(({ task, index }) => {
        if (!tasksByDate[task.deadline]) {
            tasksByDate[task.deadline] = [];
        }
        tasksByDate[task.deadline].push({ task, index });
    });

    // Render tasks grouped by date
    const sortedDates = Object.keys(tasksByDate).sort();
    sortedDates.forEach(date => {
        // Create a wrapper for this date section (date label + tasks row)
        let dateSectionHtml = `<div class="task-date-section">`;

        // Add date header label
        dateSectionHtml += `<div class="task-date-header">${date}</div>`;

        // Start a new row/group for this date's tasks
        dateSectionHtml += `<div class="task-date-group">`;

        // Render all tasks for this date
        tasksByDate[date].forEach(({ task, index }) => {
            let statusLabel = '';
            if (task.status === 'done') statusLabel = `<span class="done">✔
Виконано</span>`;
            else if (task.status === 'failed') statusLabel = `<span class="failed">✖ Не
виконано</span>`;
            else statusLabel = `<span class="status-new">Невиконано</span>`;

            let actions = '';
            if (currentUser.role === 'lead' || task.assignedTo === currentUser.name) {
                if (task.status === 'new') {
                    actions = `
                        <button class="status-btn" onclick="setStatus(${index},
'done')">Позначити виконаною</button>
                        <button class="fail-btn" onclick="setStatus(${index}, 'failed')">Не
виконано</button>
                    `;
                }
            }

            let deleteBtn = '';
```

```
            if (currentUser.role === 'lead') {
                deleteBtn = `<button class="delete-btn" onclick="deleteTask(${index})"
title="Видалити задачу">✕</button>`;
            }

            let card = `
                <div class="task">
                    ${deleteBtn}
                    <div class="task-body">
                        <div class="task-title">${task.title}</div>
                        <div class="task-meta">Пріоритет: ${task.priority}</div>
                        <div class="task-meta">Виконавець: <b>${task.assignedTo}</b> ·
Створив: ${task.createdBy}</div>
                        <div class="task-status">${statusLabel}</div>
                    </div>
                    <div class="task-actions">
                        ${actions}
                    </div>
                </div>
            `;

            dateSectionHtml += card;
        });

        // Close the date group and section
        dateSectionHtml += `</div></div>`;
        list.innerHTML += dateSectionHtml;
    });

    // Render completed and failed tasks
    completedTasks.forEach(({ task, index }) => {
        let statusLabel = '';
        if (task.status === 'done') statusLabel = `<span class="done">✔ Виконано</span>`;
        else if (task.status === 'failed') statusLabel = `<span class="failed">✖ Не
виконано</span>`;

        let actions = '';
        if (currentUser.role === 'lead' || task.assignedTo === currentUser.name) {
            if (task.status === 'done') {
                actions = `
                    <button class="undo-btn" onclick="setStatus(${index},
'new')">Скасувати</button>
                    <button class="fail-btn" onclick="setStatus(${index}, 'failed')">Не
виконано</button>
                `;
            } else if (task.status === 'failed') {
                actions = `
                    <button class="status-btn" onclick="setStatus(${index},
'done')">Позначити виконаною</button>
                    <button class="undo-btn" onclick="setStatus(${index},
'new')">Відновити</button>
                `;
            }
        }

        let deleteBtn = '';
        if (currentUser.role === 'lead') {
            deleteBtn = `<button class="delete-btn" onclick="deleteTask(${index})"
title="Видалити задачу">✕</button>`;
        }

        let card = `
            <div class="task">
                ${deleteBtn}
                <div class="task-body">
                    <div class="task-title">${task.title}</div>
                    <div class="task-meta">Дедлайн: ${task.deadline} · Пріоритет:
${task.priority}</div>
```

```
                    <div class="task-meta">Виконавець: <b>${task.assignedTo}</b> · Створив:
${task.createdBy}</div>
                    <div class="task-status">${statusLabel}</div>
                </div>
                <div class="task-actions">
                    ${actions}
                </div>
            </div>
        `;

        completedList.innerHTML += card;
    });

    // If lead, refresh stats
    if (currentUser && currentUser.role === 'lead') renderStats();
}

// ---------------------- MARK DONE ----------------------
function toggleStatus(i) {
    tasks[i].status = tasks[i].status === 'done' ? 'new' : 'done';
    saveTasks();
    renderTasks();
}

function setStatus(i, status) {
    if (!['new', 'done', 'failed'].includes(status)) return;
    tasks[i].status = status;
    saveTasks();
    renderTasks();
}

// ---------------------- DELETE TASK ----------------------
function deleteTask(index) {
    if (currentUser.role !== 'lead') {
        alert("Видаляти задачі можуть тільки тім ліди!");
        return;
    }

    const task = tasks[index];
    const confirmed = confirm(
        `Ви впевнені, що хочете видалити задачу "${task.title}"?\n\nЦю дію неможливо
скасувати.`
    );

    if (confirmed) {
        tasks.splice(index, 1);
        saveTasks();
        renderTasks();
    }
}

// ---------------------- CALENDAR ----------------------
let currentCalendarMonth = new Date().getMonth();
let currentCalendarYear = new Date().getFullYear();

function renderCalendar() {
    const year = currentCalendarYear;
    const month = currentCalendarMonth;

    // Get first day of month and number of days
    let firstDay = new Date(year, month, 1).getDay();
    // Convert Sunday (0) to 7, then shift so Monday is 0
    firstDay = firstDay === 0 ? 6 : firstDay - 1;
    const daysInMonth = new Date(year, month + 1, 0).getDate();

    // Month/year header
    const monthNames = [
        'Січень', 'Лютий', 'Березень', 'Квітень', 'Травень', 'Червень',
        'Липень', 'Серпень', 'Вересень', 'Жовтень', 'Листопад', 'Грудень'
    ];
```

```
    const headerContainer = document.getElementById('calendar-header');
    headerContainer.innerHTML = `
        <div class="calendar-nav">
            <button class="calendar-nav-btn" onclick="changeMonth(-1)">←</button>
            <div class="calendar-title">${monthNames[month]} ${year}</div>
            <button class="calendar-nav-btn" onclick="changeMonth(1)">→</button>
        </div>
    `;

    // Day names (starting from Monday)
    const dayNames = ['Пн', 'Вт', 'Ср', 'Чт', 'Пт', 'Сб', 'Нд'];
    let calendarHtml = '<div class="calendar-weekdays">';
    dayNames.forEach(day => {
        calendarHtml += `<div class="calendar-weekday">${day}</div>`;
    });
    calendarHtml += '</div>';

    // Days grid
    calendarHtml += '<div class="calendar-days">';

    // Empty cells before first day
    for (let i = 0; i < firstDay; i++) {
        calendarHtml += '<div class="calendar-day empty"></div>';
    }

    // Days of month
    for (let day = 1; day <= daysInMonth; day++) {
        const dateStr = `${year}-${String(month + 1).padStart(2, '0')}-
${String(day).padStart(2, '0')}`;
        // Filter tasks based on role: members see only their assigned tasks
        let dayTasks = tasks.filter(t => t.deadline === dateStr && t.status === 'new');
        if (currentUser.role === 'member') {
            dayTasks = dayTasks.filter(t => t.assignedTo === currentUser.name);
        }
        const dayTasksCount = dayTasks.length;

        // Check if weekend (Saturday=6 or Sunday=0 in JS)
        const dayOfWeek = new Date(year, month, day).getDay();
        const isWeekend = dayOfWeek === 0 || dayOfWeek === 6;

        let dayClass = 'calendar-day';
        if (isWeekend) {
            dayClass += ' weekend';
        }
        if (new Date(year, month, day).toDateString() === new Date().toDateString()) {
            dayClass += ' today';
        }

        let tasksList = '';
        dayTasks.slice(0, 2).forEach(t => {
            tasksList += `<div class="cal-task-item">${t.title}</div>`;
        });
        if (dayTasksCount > 2) {
            tasksList += `<div class="cal-task-more">+${dayTasksCount - 2}</div>`;
        }

        calendarHtml += `
            <div class="${dayClass}" onclick="showDayTasks('${dateStr}')">
                <div class="cal-day-number">${day}</div>
                <div class="cal-day-tasks">${tasksList || '<div class="cal-no-
tasks"></div>'}</div>
            </div>
        `;
    }

    calendarHtml += '</div>';
    document.getElementById('calendar-grid').innerHTML = calendarHtml;
}

// ---------------------- STATS ----------------------
```

```
function renderStats() {
    let container = document.getElementById('stats-list');
    if (!container) return;
    container.innerHTML = '';

    // show stats for each non-lead user
    let people = users.filter(u => u.role !== 'lead');
    if (people.length === 0) {
        container.innerHTML = '<div>Немає користувачів</div>';
        return;
    }

    people.forEach(u => {
        let assigned = tasks.filter(t => t.assignedTo === u.name);
        let total = assigned.length;
        let done = assigned.filter(t => t.status === 'done').length;
        let failed = assigned.filter(t => t.status === 'failed').length;
        let rate = total === 0 ? 0 : Math.round((done / total) * 100);

        // make each stat row clickable to view details for that user
        container.innerHTML += `
            <div class="stat-row stat-click"
onclick="showUserDetails('${u.name.replace(/'/g, "\\'")}')">
                <div class="stat-name">${u.name} · ${roleLabel(u.role)}</div>
                <div class="stat-values">Задач: <b>${total}</b> · Виконано: <b>${done}</b>
· Не виконано: <b>${failed}</b> · Успішність: <b>${rate}%</b></div>
            </div>
        `;
    });
}

// ---------------------- LEAD OVERALL EVALUATION ----------------------
// Compute intelligent per-user stats including priority-weighted performance score
function computeLeadStats() {
    const weights = { low: 1, medium: 2, high: 3 };
    const people = users.filter(u => u.role !== 'lead');
    const result = [];

    people.forEach(u => {
        const assigned = tasks.filter(t => t.assignedTo === u.name);
        const totals = { total: 0, low: 0, medium: 0, high: 0 };
        const done = { total: 0, low: 0, medium: 0, high: 0 };
        const failed = { total: 0, low: 0, medium: 0, high: 0 };

        assigned.forEach(t => {
            totals.total += 1;
            totals[t.priority] = (totals[t.priority] || 0) + 1;
            if (t.status === 'done') {
                done.total += 1;
                done[t.priority] = (done[t.priority] || 0) + 1;
            } else if (t.status === 'failed') {
                failed.total += 1;
                failed[t.priority] = (failed[t.priority] || 0) + 1;
            }
        });

        // weighted calculation
        let totalWeight = (totals.low * weights.low) + (totals.medium * weights.medium) +
(totals.high * weights.high);
        let doneWeight = (done.low * weights.low) + (done.medium * weights.medium) +
(done.high * weights.high);
        let failedWeight = (failed.low * weights.low) + (failed.medium * weights.medium) +
(failed.high * weights.high);

        let score = 0;
        if (totalWeight > 0) {
            // penalize failed tasks partially (50% weight penalty)
            score = ((doneWeight - 0.5 * failedWeight) / totalWeight) * 100;
            score = Math.round(Math.max(0, Math.min(100, score)));
        }
```

```
        const completionRate = totals.total === 0 ? 0 : Math.round((done.total /
totals.total) * 100);

        result.push({
            name: u.name,
            totals,
            done,
            failed,
            totalWeight,
            doneWeight,
            failedWeight,
            score,
            completionRate
        });
    });

    return result;
}

function openLeadStats() {
    if (!currentUser || currentUser.role !== 'lead') {
        alert('Доступно тільки для Тім ліда');
        return;
    }

    const stats = computeLeadStats();

    // build modal
    const existing = document.getElementById('lead-stats-backdrop');
    if (existing) existing.remove();

    const backdrop = document.createElement('div');
    backdrop.id = 'lead-stats-backdrop';
    backdrop.className = 'modal-backdrop';

    const modal = document.createElement('div');
    modal.className = 'modal';

    // header + close
    const header = document.createElement('div');
    header.style.position = 'relative';
    header.innerHTML = `<h3>Оцінка ефективності (інтелектуальна)</h3>`;
    const closeBtn = document.createElement('button');
    closeBtn.className = 'close-btn';
    closeBtn.innerText = '✕';
    closeBtn.onclick = () => backdrop.remove();
    header.appendChild(closeBtn);
    modal.appendChild(header);

    // explanation
    const expl = document.createElement('div');
    expl.style.marginTop = '6px';
    expl.style.fontSize = '13px';
    expl.style.color = '#2b3250';
    expl.innerText = 'Оцінка враховує пріоритети задач (Високий=3, Середній=2, Низький=1).
Неуспішні задачі частково знижують результат.';
    modal.appendChild(expl);

    // table
    const table = document.createElement('table');
    table.className = 'lead-stats-table';
    table.innerHTML = `
        <thead>

<tr><th>Користувач</th><th>Задач</th><th>Високі</th><th>Середні</th><th>Низькі</th><th>Вико
нано</th><th>Не виконано</th><th>Успішність</th><th>Оцінка</th></tr>
        </thead>
        <tbody></tbody>
```

```
    `;

    const tbody = table.querySelector('tbody');
    stats.forEach(s => {
        const tr = document.createElement('tr');
        tr.innerHTML = `
            <td><b>${s.name}</b></td>
            <td>${s.totals.total}</td>
            <td>${s.totals.high}</td>
            <td>${s.totals.medium}</td>
            <td>${s.totals.low}</td>
            <td>${s.done.total}</td>
            <td>${s.failed.total}</td>
            <td>${s.completionRate}%</td>
            <td>${s.score}%</td>
        `;
        tbody.appendChild(tr);
    });

    modal.appendChild(table);

    // actions
    const actions = document.createElement('div');
    actions.className = 'modal-actions';
    const sendBtn = document.createElement('button');
    sendBtn.className = 'send-email-btn';
    sendBtn.innerText = 'Надіслати на пошту';
    sendBtn.onclick = () => sendLeadStatsEmail(stats);
    const closeAction = document.createElement('button');
    closeAction.className = 'close-modal-btn';
    closeAction.innerText = 'Закрити';
    closeAction.onclick = () => backdrop.remove();
    actions.appendChild(sendBtn);
    actions.appendChild(closeAction);
    modal.appendChild(actions);

    backdrop.appendChild(modal);
    document.body.appendChild(backdrop);
}

function sendLeadStatsEmail(stats) {
    // ensure we have an email for the lead
    let leadEmail = (currentUser && currentUser.email) ? currentUser.email : null;
    if (!leadEmail) {
        leadEmail = prompt('Введіть e-mail Тім ліда для відправки статистики:', leadEmail
|| '');
        if (!leadEmail) return alert('Адреса електронної пошти не вказана. Відправка
скасована.');
        // persist onto currentUser and users[] if possible
        currentUser.email = leadEmail;
        const u = users.find(x => x.name === currentUser.name);
        if (u) { u.email = leadEmail; saveUsers(); }
        // update session
        setSessionUser();
    }

    // build body text
    let body = `Оцінка ефективності — ${new Date().toLocaleString()}\n\n`;
    body += 'Пояснення: пріоритети (Високий=3, Середній=2, Низький=1). Неуспішні задачі
частково знижують оцінку.\n\n';
    stats.forEach(s => {
        body += `Користувач: ${s.name}\n`;
        body += `Задач всього: ${s.totals.total} (Високі:${s.totals.high},
Середні:${s.totals.medium}, Низькі:${s.totals.low})\n`;
        body += `Виконано: ${s.done.total} · Не виконано: ${s.failed.total} · Успішність:
${s.completionRate}% · Оцінка: ${s.score}%\n\n`;
    });

    const subject = `Статистика ефективності ${new Date().toLocaleDateString()}`;
```

```
    const mailto =
`mailto:${encodeURIComponent(leadEmail)}?subject=${encodeURIComponent(subject)}&body=${enco
deURIComponent(body)}`;

    // open mail client
    window.location.href = mailto;
}

// state for filter
let statsFilterUser = null;

function showUserDetails(name) {
    const detail = document.getElementById('user-detail');
    if (!detail) return;
    const assigned = tasks.filter(t => t.assignedTo === name);
    const total = assigned.length;
    const done = assigned.filter(t => t.status === 'done').length;
    const failed = assigned.filter(t => t.status === 'failed').length;
    const rate = total === 0 ? 0 : Math.round((done / total) * 100);

    let listHtml = '';
    assigned.forEach(t => {
        listHtml += `
            <div class="task small">
                <div class="task-body">
                    <div class="task-title">${t.title}</div>
                    <div class="task-meta">Дедлайн: ${t.deadline} · Пріоритет:
${t.priority}</div>
                    <div class="task-status">${t.status === 'done' ? '<span class="done">✔
Виконано</span>' : t.status === 'failed' ? '<span class="failed">✖ Не виконано</span>' :
'<span class="status-new">Невиконано</span>'}</div>
                </div>
            </div>
        `;
    });

    detail.innerHTML = `
        <h4>Деталі: ${name}</h4>
        <div>Задач всього: <b>${total}</b></div>
        <div>Виконано: <b>${done}</b></div>
        <div>Не виконано: <b>${failed}</b></div>
        <div>Успішність: <b>${rate}%</b></div>
        <div style="margin-top:10px; display:flex; gap:8px">
            <button onclick="filterToUser('${name.replace(/'/g, "\\'")}')">Показати тільки
цю людину</button>
            <button onclick="clearFilter()">Скинути фільтр</button>
        </div>
        <div style="margin-top:10px">${listHtml || '<div>Немає задач</div>'}</div>
    `;
    detail.classList.remove('hidden');
}

function filterToUser(name) {
    statsFilterUser = name;
    renderTasks();
}

function clearFilter() {
    statsFilterUser = null;
    const detail = document.getElementById('user-detail');
    if (detail) detail.classList.add('hidden');
    renderTasks();
}

// On load: if there is a remembered logged-in user in sessionStorage, restore
if (sessionStorage.getItem('currentUser')) {
    currentUser = JSON.parse(sessionStorage.getItem('currentUser'));
    document.getElementById('login-container').classList.add('hidden');
    document.getElementById('app').classList.remove('hidden');
```

```
    document.getElementById('welcome').innerText = `Вітаю, ${currentUser.name}! Ваша роль:
${roleLabel(currentUser.role)}`;
    prepareRoleView();
    renderTasks();
}

// Persist currentUser in session so a refresh doesn't immediately log out during testing
function setSessionUser() {
    if (currentUser) sessionStorage.setItem('currentUser', JSON.stringify(currentUser));
}

// Wrap existing actions to set session user
const originalLogin = login;
login = function() {
    originalLogin();
    setSessionUser();
};

const originalLogout = logout;
logout = function() {
    sessionStorage.removeItem('currentUser');
    originalLogout();
};

// ---------------------- CALENDAR DAY TASKS ----------------------
function showDayTasks(dateStr) {
    const dayTasksPanel = document.getElementById('calendar-day-tasks');
    const dayTitle = document.getElementById('calendar-day-title');
    const dayList = document.getElementById('calendar-day-list');

    // Get all tasks for this date, filtered by role
    let dayTasks = tasks.filter(t => t.deadline === dateStr);
    if (currentUser.role === 'member') {
        dayTasks = dayTasks.filter(t => t.assignedTo === currentUser.name);
    }

    if (dayTasks.length === 0) {
        dayTasksPanel.classList.add('hidden');
        return;
    }

    // Format date for title
    const date = new Date(dateStr + 'T00:00:00');
    const day = date.getDate();
    const month = date.toLocaleDateString('uk-UA', { month: 'long' });
    const year = date.getFullYear();
    dayTitle.textContent = `Задачі на ${day} ${month} ${year}`;

    // Render tasks as cards
    let html = '';
    dayTasks.forEach(t => {
        const priorityLabels = { low: 'Низький', medium: 'Середній', high: 'Високий' };
        const statusLabels = { new: 'Потрібно виконати', done: 'Виконано', failed: 'Не
виконано' };

        let priorityClass = '';
        if (t.priority === 'high') priorityClass = 'task-high';
        else if (t.priority === 'medium') priorityClass = 'task-medium';
        else priorityClass = 'task-low';

        let statusClass = '';
        if (t.status === 'done') statusClass = 'task-done';
        else if (t.status === 'failed') statusClass = 'task-failed';

        // Show delete button only for leads
        const showDeleteBtn = currentUser.role === 'lead' && t.status === 'new';

        html += `
            <div class="task ${priorityClass} ${statusClass}">
                <div class="task-header">
```

```
                    <div class="task-title">${t.title}</div>
                    ${showDeleteBtn ? `<button class="delete-btn"
onclick="deleteTask(${t.id})">×</button>` : ''}
                </div>
                <div class="task-meta">
                    <span>Дедлайн: ${t.deadline}</span>
                    <span>Пріоритет: ${priorityLabels[t.priority]}</span>
                    <span>Статус: ${statusLabels[t.status]}</span>
                </div>
                ${t.assignedTo ? `<div class="task-assigned">Виконавець:
${t.assignedTo}</div>` : ''}
                ${t.createdBy ? `<div class="task-creator">Створив: ${t.createdBy}</div>` :
''}
            </div>
        `;
    });

    dayList.innerHTML = html;
    dayTasksPanel.classList.remove('hidden');
}

// ----------------------- CALENDAR NAVIGATION -----------------------
function changeMonth(direction) {
    currentCalendarMonth += direction;

    if (currentCalendarMonth > 11) {
        currentCalendarMonth = 0;
        currentCalendarYear++;
    } else if (currentCalendarMonth < 0) {
        currentCalendarMonth = 11;
        currentCalendarYear--;
    }

    renderCalendar();
}
```