МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ВЕБДОДАТКУ ДЛЯ ВЕДЕННЯ ОСОБИСТОГО ЩОДЕННИКА ТА ТРЕКІНГУ ЗВИЧОК ІЗ ВІЗУАЛІЗАЦІЄЮ ПРОГРЕСУ»**

Виконав: студент 4 курсу, групи 6.1222
спеціальності 122 комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми комп'ютерні науки
(назва освітньої програми)
І. О. Єфіменко
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, д.е.н., доцент Полуектова Н. Р..
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, к.ф.-м.н., доцент Лісняк А. О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 комп'ютерні науки
(шифр і назва)

Освітня програма комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних наук, к.пед.н., доцент

_____ Пшенична О.С.
(підпис)

" 12 " грудня 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Єфіменку Івану Олександровичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка вебдодатку для ведення особистого щоденника та трекінгу звичок із візуалізацією прогресу

керівник роботи Полуектова Наталія Робертівна, д.е.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 « грудня 2025 року № 1741-с

2. Строк подання студентом роботи 04.06.2026

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Теоретичні основи.
2. Проєктування вебдодатку.
3. Реалізація вебдодатку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 05.12.2026

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	03.03.2026	
2.	Збір вихідних даних.	20.03.2026	
3.	Обробка методичних та теоретичних	23.03.2025	
	джерел.		
4.	Розробка першого та другого розділу.	14.04.2026	
5.	Розробка третього розділу.	01.05.2026	
6.	Оформлення та нормоконтроль	04.06.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2026	

Студент _____ І. О. Єфіменко
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н. Р. Полуектова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка вебдодатку для ведення особистого щоденника та трекінгу звичок із візуалізацією прогресу»: 88 с., 12 рис., 8 табл., 32 джерела, 2 додатки.

АВТОМАТИЗАЦІЯ ОСОБИСТОГО ЩОДЕННИКА, ВЕБДОДАТОК, POSTGRESQL.REACT, TYPESCRIPT, TAILWIND CSS.

Об’єкт дослідження – процес організації особистої діяльності користувачів із використанням цифрових інструментів самоменеджменту та підвищення продуктивності.

Предмет дослідження – методи та програмні засоби розробки вебдодатків для ведення особистого щоденника, трекінгу звичок і візуалізації індивідуального прогресу користувача.

Метою роботи є проєктування та розробка вебдодатка для ведення особистого щоденника і трекінгу звичок із можливістю візуалізації прогресу користувача, що дозволяє підвищити рівень самоорганізації та ефективності щоденної діяльності.

Методи дослідження включають аналіз наукової літератури та існуючих вебдодатків, системний аналіз і порівняння аналогів, методи проєктування програмних систем і баз даних.

У роботі проаналізовано концепції особистого щоденника та трекінгу звичок, а також існуючі програмні рішення для самоменеджменту з визначенням їхніх переваг і недоліків. Спроєктовано архітектуру вебдодатка, модель даних та основні взаємодії між компонентами системи. Реалізовано основні модулі застосунку: інтерфейс для ведення щоденника, систему керування звичками з фіксацією виконання та алгоритми розрахунку прогресу. Окрему увагу приділено візуалізації даних користувача для відстеження динаміки звичок та емоційного стану.

SUMMARY

Bachelor's Thesis «Development of a Web Application for Maintaining a Personal Diary and Habit Tracking with Progress Visualization»: 88 pages, 12 figures, 8 tables, 32 sources, 2 appendices.

AUTOMATION OF PERSONAL DIARY, WEB APPLICATION, POSTGRESQL.REACT, TYPESCRIPT, TAILWIND CSS

Object of research – the process of organizing users' personal activities using digital self-management tools and productivity enhancement methods.

Subject of research – methods and software tools for developing web applications for personal diary keeping, habit tracking, and visualization of individual user progress.

The aim of the work is to design and develop a web application for maintaining a personal diary and tracking habits with progress visualization, enabling improved self-organization and efficiency of daily activities.

Research methods include analysis of scientific literature and existing web applications, system analysis and comparison of analogues, and methods of software system and database design.

The work analyzes the concepts of personal diary keeping and habit tracking, as well as existing self-management software solutions with identification of their advantages and disadvantages. The system architecture, data model, and main component interactions are designed. The main application modules are implemented, including a diary interface, a habit tracking system with completion logging, and progress calculation algorithms. Special attention is given to data visualization, which enables tracking of habit dynamics and emotional state.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2

Реферат 4

Summary 5

Вступ..... 7

1 Теоретичні основи..... 9

1.1 Поняття особистого щоденника та його функції у психології та менеджменті 9

1.2 Методи трекінгу звичок: огляд сучасних практик та інструментів... 12

1.3 Аналіз існуючих вебдодатків для самоменеджменту (Diary, Habitica, Notion, Google Keep) 16

1.4 Вимоги до системи: функціональні та нефункціональні 19

2 Проєктування вебдодатку 22

2.1 Архітектура системи: клієнт–серверна модель, бази даних, API 22

2.2. Вибір технологій 27

2.3 Моделювання даних: структура записів щоденника, модуль звичок, користувацькі профілі 30

2.4 UML-діаграми..... 34

3 Реалізація вебдодатку 41

3.1 Реалізація інтерфейсу користувача: створення та редагування записів, додавання звичок..... 41

3.2 Механізм трекінгу звичок: щоденні/тижневі нагадування, статус виконання..... 47

3.3 Візуалізація прогресу: графіки, діаграми, календарі 51

3.4 Інтеграція з зовнішніми сервісами 53
3.5 Тестування функціоналу та юзабіліті 56
3.6 Інструкція користувача..... 59
Висновки 61
Перелік посилань..... 63
Додаток А Лістинг реалізації інтерфейсу користувача..... 67
Додаток Б Інтерфейс вебдодатку..... 84

ВСТУП

Сучасний розвиток інформаційних технологій сприяє активному використанню цифрових інструментів для самоорганізації, планування діяльності та контролю особистих досягнень. Особливої популярності набувають вебдодатки для ведення особистих щоденників і трекінгу звичок, які допомагають користувачам фіксувати важливі події, відстежувати прогрес у досягненні цілей та формувати корисні звички.

Попри значну кількість існуючих програмних рішень, багато з них орієнтовані лише на окремі аспекти самоменеджменту та не забезпечують комплексного підходу до ведення особистих записів, контролю звичок і аналізу результатів. Тому актуальним є створення вебдодатка, який поєднує функції особистого щоденника, трекінгу звичок та засобів візуалізації прогресу в єдиному середовищі.

Метою бакалаврської роботи є проєктування та розробка вебдодатка для ведення особистого щоденника і трекінгу звичок із можливістю візуалізації прогресу користувача.

Для досягнення поставленої мети необхідно вирішити такі завдання: проаналізувати теоретичні основи ведення особистого щоденника та трекінгу звичок; дослідити існуючі вебдодатки для самоменеджменту; сформувати функціональні та нефункціональні вимоги до системи; спроєктувати архітектуру вебдодатка та структуру бази даних; реалізувати функціонал ведення записів і відстеження звичок; розробити механізми візуалізації прогресу та провести тестування програмного продукту.

Об’єктом дослідження є процес організації особистої діяльності користувачів за допомогою цифрових засобів самоменеджменту.

Предметом дослідження є методи та засоби розробки вебдодатків для ведення особистого щоденника, трекінгу звичок і візуалізації статистичних даних.

Практичне значення роботи полягає у створенні вебдодатка, який забезпечує зручне ведення особистих записів, контроль виконання звичок та аналіз результатів діяльності користувача за допомогою графіків і статистичних показників.

Новизна роботи полягає в інтеграції функцій особистого щоденника, трекінгу звичок та візуалізації прогресу в єдиній веборієнтованій системі, що сприяє підвищенню ефективності самоорганізації та досягненню особистих цілей.

1 ТЕОРЕТИЧНІ ОСНОВИ

1.1 Поняття особистого щоденника та його функції у психології та менеджменті

Особистий щоденник є одним із найдавніших інструментів фіксації індивідуального досвіду людини. Упродовж багатьох століть люди використовували щоденники для запису важливих подій, власних думок, спостережень, переживань і життєвих висновків. Спочатку щоденники існували переважно у паперовому вигляді, однак із розвитком цифрових технологій вони поступово трансформувалися в електронні сервіси та вебдодатки, що надають значно ширші можливості для збереження, аналізу та візуалізації інформації.

У сучасному розумінні особистий щоденник являє собою систему регулярного документування особистих подій, емоцій, думок і цілей, яка сприяє кращому розумінню власного внутрішнього світу та навколишньої дійсності. Ведення щоденника дозволяє людині не лише зберігати важливі спогади, але й аналізувати власну поведінку, оцінювати результати прийнятих рішень і відстежувати зміни, що відбуваються в житті. Саме тому особистий щоденник розглядається як ефективний інструмент саморозвитку та самопізнання [1].

Особливо важливого значення щоденник набуває у сфері психології. Сучасні психологічні дослідження підтверджують, що регулярне ведення особистих записів позитивно впливає на психоемоційний стан людини, сприяє зниженню рівня тривожності та допомагає ефективніше справлятися зі стресовими ситуаціями. Процес письмового викладення думок дозволяє структурувати внутрішні переживання, усвідомити причини виникнення певних емоцій та знайти способи їх подолання [2].

Однією з найважливіших психологічних функцій особистого щоденника є саморефлексія. Вона полягає у здатності людини аналізувати власні дії, вчинки та переживання. Записуючи свої думки, людина ніби дивиться на ситуацію з

боку, що дозволяє більш об’єктивно оцінити власну поведінку та зробити відповідні висновки. Завдяки цьому формується навичка критичного мислення та підвищується рівень самосвідомості.

Іншою важливою функцією є емоційна регуляція. Щоденник створює безпечний простір для вираження емоцій, які часто складно висловити іншим людям. Записування власних переживань допомагає зменшити психологічне напруження, позбутися негативних емоцій та знайти внутрішню рівновагу. Наукові дослідження свідчать, що практика джорналінгу сприяє покращенню емоційного стану та зниженню рівня стресу [3].

Не менш важливою є функція формування особистісної ідентичності. Аналізуючи власні записи за тривалий період часу, людина може простежити зміни у своїх поглядах, цінностях та життєвих пріоритетах. Це сприяє кращому розумінню власних потреб, мотивів та життєвих цілей. Щоденник дозволяє усвідомити особистий прогрес і стає своєрідною історією розвитку людини.

Також особистий щоденник виконує меморіальну функцію, оскільки забезпечує збереження важливої інформації про події, досягнення та життєвий досвід. Людина може повертатися до попередніх записів, аналізувати минулі ситуації та використовувати отримані знання для прийняття майбутніх рішень [4]. Таким чином, щоденник виступає своєрідною зовнішньою пам’яттю, яка допомагає зберігати значущу інформацію протягом тривалого часу.

Крім психологічного значення, особистий щоденник широко використовується у сфері менеджменту та самоорганізації. У сучасних умовах ефективне управління часом і ресурсами є необхідною складовою успішної професійної діяльності. Саме тому щоденники активно використовуються керівниками, підприємцями, студентами та працівниками різних галузей для планування роботи та контролю виконання поставлених завдань.

Однією з основних функцій щоденника у менеджменті є планування діяльності. За допомогою щоденника користувач може формувати перелік завдань на день, тиждень або місяць, визначати строки їх виконання та

контролювати результати роботи. Це дозволяє раціонально розподіляти навантаження та уникати перевантаження великою кількістю справ.

Тісно пов’язаною з плануванням є функція тайм-менеджменту. Ведення щоденника сприяє ефективнішому використанню часу шляхом розподілу завдань за пріоритетністю. Багато методик управління часом передбачають фіксацію завдань у щоденнику та їх подальше ранжування за ступенем важливості. Завдяки цьому користувач отримує можливість концентруватися насамперед на найбільш значущих завданнях та уникати втрати часу на другорядні справи [5].

Ще однією важливою функцією є контроль виконання завдань та оцінювання результатів діяльності. Фіксація виконаних справ дозволяє аналізувати власну продуктивність, визначати причини невиконання окремих завдань та знаходити шляхи підвищення ефективності роботи. У професійному середовищі подібні записи часто використовуються як основа для складання звітів та оцінювання результатів діяльності працівників.

В умовах сучасної цифровізації особливого поширення набули електронні щоденники та вебдодатки для самоменеджменту. На відміну від традиційних паперових аналогів, вони забезпечують автоматичне збереження даних, швидкий пошук інформації, можливість додавання нагадувань, інтеграцію з календарями та візуалізацію статистики. Крім того, такі системи часто поєднують функції особистого щоденника і трекера звичок, що дозволяє не лише записувати події, а й відстежувати досягнення довгострокових цілей.

Таким чином, особистий щоденник є універсальним інструментом, який поєднує функції психологічної підтримки, саморефлексії, планування та контролю діяльності. У психології він сприяє розвитку самосвідомості, емоційної стабільності та особистісного зростання, а в менеджменті виступає ефективним засобом організації часу, планування роботи та підвищення продуктивності. Саме тому сучасні цифрові системи самоменеджменту активно використовують концепцію особистого щоденника як основу для створення комплексних платформ підтримки особистої ефективності.

1.2 Методи трекінгу звичок: огляд сучасних практик та інструментів

У сучасному суспільстві питання формування корисних звичок та підтримання особистої продуктивності набувають особливого значення. Постійне зростання обсягу інформації, високий темп життя та необхідність досягнення особистих і професійних цілей вимагають від людини високого рівня самоорганізації. Одним із найбільш ефективних інструментів самоконтролю є трекінг звичок, який дозволяє систематично відстежувати виконання певних дій та оцінювати прогрес у досягненні поставлених цілей.

Трекер звичок являє собою інструмент для моніторингу регулярності виконання певної діяльності протягом визначеного періоду часу. Його основне призначення полягає у формуванні нових корисних звичок, підтриманні вже існуючих та поступовому усуненні небажаних моделей поведінки. Зазвичай трекер відображає інформацію у вигляді таблиці, календаря або графічної візуалізації, що дозволяє користувачеві наочно оцінювати власні досягнення та контролювати послідовність виконання завдань [6].

З наукової точки зору звичка розглядається як автоматизована форма поведінки, яка формується в результаті багаторазового повторення певних дій у схожих умовах. Згодом така поведінка потребує все менше свідомих зусиль і стає природною частиною повсякденного життя людини. Формування звичок базується на механізмах навчання та закріплення поведінкових реакцій, що пояснюється численними дослідженнями у сфері психології та нейронаук [7].

Важливим науковим підґрунтям трекінгу звичок є концепція самомоніторингу (self-monitoring). Дослідження демонструють, що сам факт відстеження власної поведінки значно підвищує ймовірність досягнення поставленої мети. Метааналіз за участю понад 19 тисяч осіб показав, що систематичний контроль прогресу позитивно впливає на успішність формування нових звичок та дотримання поведінкових змін [8]. Завдяки цьому трекінг став одним із найбільш поширених методів саморегуляції, який використовується як

у психологічній практиці, так і в сучасних цифрових системах самоменеджменту.

Ефективність трекінгу звичок пояснюється дією кількох психологічних механізмів. Першим із них є самомоніторинг, який забезпечує постійний зворотний зв'язок між поведінкою людини та її результатами. Кожна відмітка про виконання завдання створює відчуття прогресу та стимулює подальше дотримання обраної моделі поведінки.

Другим механізмом є дія системи винагороди мозку. Коли користувач бачить серію успішно виконаних днів або досягнуті цілі, активізуються процеси позитивного підкріплення. Навіть невеликі досягнення викликають відчуття задоволення та мотивують до подальшого виконання звички. Саме на цьому принципі базуються популярні механізми «серій» (streaks), які використовуються в багатьох цифрових застосунках для вивчення мов, спорту чи особистої продуктивності.

Третім важливим механізмом є зниження когнітивного навантаження. Людині не потрібно постійно пам'ятати про всі свої завдання та звички, оскільки відповідна інформація зберігається в трекері. Це дозволяє зосередитися безпосередньо на виконанні дій, а не на утриманні великої кількості нагадувань у пам'яті.

Особливу роль у формуванні звичок відіграє концепція ідентичності. Дослідники зазначають, що нові звички закріплюються значно ефективніше тоді, коли людина починає асоціювати їх із власною особистістю. Наприклад, замість формулювання «я намагаюся читати книги» формується установка «я є людиною, яка регулярно читає». Трекер дозволяє візуально підтверджувати цей процес, демонструючи накопичений досвід виконання певної поведінки [9].

На практиці існує декілька основних методів трекінгу звичок. Одним із найпоширеніших є метод щоденної сітки звичок (Daily Habit Grid). Його суть полягає у створенні таблиці, де по вертикалі розміщуються звички, а по горизонталі – дні місяця. Кожного разу після виконання відповідної дії

користувач ставить позначку у відповідній клітинці. Такий підхід дозволяє швидко оцінити регулярність виконання звичок і виявити пропущені дні.

Іншим популярним методом є накопичення звичок (Habit Stacking). Його принцип полягає у прив'язуванні нової звички до вже існуючої дії. Наприклад, після ранкової чашки кави користувач може виконувати коротку фізичну вправу або записувати плани на день. Такий підхід спрощує формування нових моделей поведінки, оскільки використовує вже сформовані автоматичні дії як тригери для нових звичок.

Широко використовується також аналіз петлі звички (Habit Loop Analysis), який ґрунтується на концепції поведінкового циклу. У межах цього підходу аналізуються чотири ключові компоненти: тригер, бажання, реакція та винагорода. Виявлення взаємозв'язків між цими елементами дозволяє краще зрозуміти причини виникнення певної поведінки та розробити стратегії її зміни.

Для довгострокового контролю результатів застосовуються прогрес-трекери (Progress Trackers), які дозволяють аналізувати виконання звичок протягом тижня, місяця або більш тривалого періоду. Такі інструменти допомагають визначати рівень успішності, виявляти типові перешкоди та коригувати подальші дії для досягнення кращих результатів [10].

Сучасні інструменти трекінгу умовно поділяються на паперові та цифрові. До паперових належать класичні щоденники, календарі, спеціалізовані планери та система Bullet Journal. Остання поєднує елементи нотатника, календаря, списку справ і трекера звичок, забезпечуючи високу гнучкість налаштування відповідно до потреб користувача [11].

Перевагами паперових інструментів є простота використання, відсутність технічних вимог та можливість творчого оформлення. Дослідження показують, що для багатьох людей фізичне записування інформації сприяє кращому запам'ятовуванню та усвідомленню власного прогресу [12].

Проте останніми роками все більшої популярності набувають цифрові інструменти трекінгу звичок. Їх розвиток став можливим завдяки поширенню мобільних застосунків, вебсервісів та хмарних технологій. Цифрові системи

дозволяють автоматизувати процес збору даних, надсилати нагадування, генерувати статистичні звіти та візуалізувати результати за допомогою графіків і діаграм.

Згідно з результатами досліджень цифрових інтервенцій зміни поведінки, найефективнішими елементами таких систем є постановка цілей, самомоніторинг, нагадування, зворотний зв'язок та соціальна підтримка [13]. Саме ці компоненти сьогодні використовуються більшістю сучасних платформ для самоменеджменту.

Серед популярних цифрових рішень можна виділити Habitica, яка використовує механізми гейміфікації та перетворює виконання звичок на рольову гру; Streaks, що базується на підтримці безперервних серій виконання завдань; а також Notion, яка дозволяє створювати власні шаблони трекерів і системи персонального планування.

Особливу увагу сучасні цифрові сервіси приділяють використанню підказок або тригерів (cues). Найчастіше застосовуються часові нагадування у вигляді push-повідомлень, які активуються у визначений момент доби. Також використовуються соціальні тригери, контекстні нагадування залежно від місця перебування користувача та інші механізми підтримки необхідної поведінки [13].

Разом із численними перевагами трекінг звичок має певні обмеження. Надмірний контроль та постійне відстеження показників можуть викликати підвищену тривожність, перфекціонізм або залежність від статистичних результатів. Тому фахівці рекомендують зосереджуватися не на абсолютній бездоганності виконання, а на поступовому формуванні стабільної поведінки та регулярному аналізі власного прогресу [9].

Отже, трекінг звичок є одним із найбільш ефективних сучасних інструментів самоменеджменту, що базується на принципах самомоніторингу, позитивного підкріплення та систематичного аналізу поведінки. Сучасні методи трекінгу охоплюють широкий спектр практик – від простих паперових таблиць до складних цифрових платформ із засобами аналітики та візуалізації.

Використання таких інструментів сприяє розвитку самодисципліни, підвищенню особистої продуктивності та успішному формуванню корисних звичок, що робить їх важливою складовою сучасних систем особистого щоденника та самоменеджменту.

1.3 Аналіз існуючих вебдодатків для самоменеджменту (Diary, Habitica, Notion, Google Keep)

Сучасний ринок програмного забезпечення пропонує широкий спектр вебдодатків для організації особистого часу, відстеження звичок та ведення нотаток. Для визначення напрямів розробки власного рішення необхідно проаналізувати найпоширеніші існуючі інструменти самоменеджменту: щоденники (Diary/Penzu), ігровий трекер звичок Habitica, багатофункціональну платформу Notion та легкий інструмент нотаток Google Keep.

Цифрові щоденники (Diary-додатки) є одним із найстаріших форматів інструментів самоменеджменту. Серед найвідоміших веборієнтованих рішень виділяються Penzu та Day One. Penzu – вебцентрична платформа, що існує з 2008 року і позиціонує себе як найбезпечніший цифровий щоденник у світі [14]. Day One орієнтований насамперед на екосистему Apple і пропонує мультимедійне збереження записів.

Ключові можливості Penzu включають: шифрування за стандартом 256-bit AES, подвійний захист паролем, довільне форматування тексту, нагадування електронною поштою та функцію «капсула часу» – відправку нотаток самому собі у майбутньому [14]. Day One підтримує додавання фотографій, відео, аудіозаписів, автоматичне збереження метаданих (місцезнаходження, погода, музика) та перегляд записів у форматі часової шкали.

Серед переваг цього класу додатків виділяють надійний захист приватності, простоту використання та фокус на тривалому зберіганні особистих роздумів [15]. Проте Diary-додатки мають суттєві обмеження з погляду

самоменеджменту: вони призначені переважно для рефлексії, а не для планування; не містять функцій управління завданнями, відстеження цілей або аналітики продуктивності; мультимедійна підтримка Penzu поступається конкурентам. Інтерфейс веброзробки залишається мінімалістичним без підтримки складних структур даних [15].

Habitica – це вебдодаток і мобільний застосунок для відстеження звичок та управління завданнями, що використовує механіку рольових ігор (RPG) для мотивації користувачів. Заснований у 2013 році, він перетворює щоденні задачі на ігрові квести [16]. Додаток безкоштовний, проте пропонує платну підписку вартістю від 4,99 до 47,99 доларів на рік.

Функціональність Habitica поділяється на три категорії задач: Habits (гнучкі звички, що можна відмічати декілька разів на день), Dailies (щоденні завдання, що повторюються автоматично) та To-Dos (одноразові задачі) [16]. За виконання завдань користувач отримує очки досвіду (XP), золото та підвищує рівень персонажа. Пропуск завдань призводить до втрати здоров'я аватара. Серед соціальних функцій – формування груп (Parties), гільдій та участь у спільних квестах.

Перевагою Habitica є висока мотивація через гейміфікацію та соціальну підзвітність [17]. Разом із тим виявлено низку недоліків: інтерфейс переобтяжений ігровими елементами, що ускладнює роботу для користувачів, які не захоплюються іграми; спостерігаються нестабільність роботи, помилки та некоректне скидання серій [17]; відсутні розширені інструменти аналізу продуктивності та управління проектами. Додаток погано масштабується для складних робочих процесів і командної роботи.

Notion – це універсальна платформа, розроблена компанією Notion Labs, Inc. Перший реліз відбувся 2016 року, а сьогодні вона поєднує нотатки, управління проектами, вікі-бази знань та бази даних в єдиному настроюваному середовищі [18]. Один із ключових елементів Notion – блокова структура контенту, що дозволяє комбінувати текст, таблиці, зображення, код і вбудовані об'єкти.

До основних можливостей платформи належать: реляційні бази даних із підтримкою kanban-дошок, календарів і таблиць; гнучке налаштування шаблонів; спільна робота в реальному часі; гранульоване управління дозволами; інтеграція з зовнішніми сервісами. У лютому 2023 року було представлено Notion AI – вбудований ШІ-асистент для резюмування сторінок, генерації ідей та автоматизації контенту [18]. У вересні 2025 року вийшла версія 3.0 з функцією ШІ-агента.

Notion визнається лідером серед інструментів продуктивності завдяки своїй гнучкості: від простих списків справ до складних корпоративних вікі [18]. Однак платформа має значну криву навчання – нові користувачі стикаються з труднощами через надмірну кількість параметрів налаштування [15]. Крім того, гнучкість є одночасно і слабкістю: відсутність чіткої структури змушує користувачів витрачати час на побудову системи замість виконання завдань. Для управління виключно особистими звичками та часом Notion може виявитися надлишково складним рішенням.

Google Keep – це безкоштовний хмарний інструмент нотаток, розроблений компанією Google. Він інтегрований з екосистемою Google Workspace і надає доступ до нотаток через браузер, мобільні застосунки та розширення для Chrome [19]. Основні можливості включають: текстові нотатки, списки, аудіозаписи, зображення, розпізнавання тексту на фото (OCR), нагадування за часом і місцем, мітки та спільний доступ.

Головними перевагами Google Keep є простота освоєння, висока швидкість введення нотаток і безшовна інтеграція з Google Calendar, Gmail та Google Drive [19]. Рейтинг зручності використання за оцінками користувачів сягає 4,9 з 5. Проте платформа має суттєві обмеження: відсутня підтримка папок для категоризації нотаток, не підтримуються відео та PDF, немає таблиць і розширеного форматування тексту [20]. Додавання прапорців автоматично перетворює всю нотатку на список завдань, унеможливлюючи змішаний формат. Google Keep не призначений для управління складними проектами та не забезпечує розширеної аналітики чи відстеження звичок.

Кожен із розглянутих додатків реалізує окремий аспект самоменеджменту, проте жоден не покриває всього спектра потреб. Diary-додатки забезпечують якісне зберігання особистих роздумів, але позбавлені інструментів планування. Habitica мотивує через гейміфікацію, проте не підходить для структурованого управління часом. Notion пропонує найбагатший функціонал, але потребує значних зусиль на налаштування і не розрахований спеціально на відстеження звичок. Google Keep є найпростішим у використанні, однак надто обмежений для повноцінного самоменеджменту.

Таким чином, аналіз існуючих рішень підтверджує потребу у розробці вебдодатку, який об’єднає ведення щоденника, відстеження звичок та управління завданнями в єдиному інтуїтивному інтерфейсі без надлишкової складності або вузькоспеціалізованого підходу.

1.4 Вимоги до системи: функціональні та нефункціональні

Перед початком проєктування вебдодатку «Особистий щоденник і трекер звичок» було сформовано повний перелік вимог до системи. Вони визначають, які функції повинна виконувати система, а також які якісні характеристики вона має забезпечувати. Для структуризації всі вимоги поділено на функціональні та нефункціональні.

Функціональні вимоги описують безпосередню поведінку системи та дії, які доступні користувачеві, тоді як нефункціональні визначають якість роботи програмного продукту, включаючи безпеку, продуктивність та зручність використання.

Функціональні вимоги визначають основні можливості вебдодатку та сценарії взаємодії користувача із системою. Вони охоплюють чотири ключові підсистеми: авторизацію, ведення щоденника, трекінг звичок та аналітику. Кожна вимога має унікальний ідентифікатор та пріоритет реалізації за методологією MoSCoW.

Нижче наведено узагальнений перелік функціональних вимог до системи (табл. 1.1).

Таблиця 1.1 – Функціональні вимоги до системи

ID	Опис вимоги	Пріоритет
FR-01	Реєстрація користувача за email і паролем	Must
FR-02	Авторизація та вихід із системи	Must
FR-03	Підтримка сесій користувача та автоматичний вхід	Must
FR-04	Редагування профілю користувача	Should
FR-05	Відновлення пароля	Should
FR-06	Створення записів щоденника (текст, дата, настрій, теги)	Must
FR-07	Перегляд записів у хронологічному порядку	Must
FR-08	Редагування та видалення записів	Must
FR-09	Фільтрація та пошук записів	Should
FR-10	Створення звичок (назва, колір, іконка, частота)	Must
FR-11	Редагування та видалення звичок	Must
FR-12	Відмітка виконання звички за день	Must
FR-13	Автоматичний розрахунок streak (серії виконання)	Must
FR-14	Розрахунок максимальної серії виконань	Should
FR-15	Формування графіків та статистики виконання	Must
FR-16	Відображення heatmap активності	Should
FR-17	Відображення KPI показників (статистика користувача)	Must
FR-18	Надсилання нагадувань про звички	Could
FR-19	Інтеграція із зовнішніми сервісами (календар, email)	Could

Як видно з таблиці, базовий функціонал системи включає авторизацію, ведення щоденника, управління звичками та базову аналітику. Додаткові можливості, такі як інтеграції та нагадування, віднесені до другорядних і можуть бути реалізовані в майбутніх версіях системи.

Нефункціональні вимоги (табл. 1.2) визначають якісні характеристики системи, тобто те, як саме вона повинна працювати. Вони охоплюють безпеку, продуктивність, зручність використання та стабільність роботи вебдодатку.

Таблиця 1.2 – Нефункціональні вимоги до системи

ID	Опис вимоги	Критерій перевірки
NFR-01	Захист паролів користувачів	використання bcrypt або argon2
NFR-02	Обмеження доступу до власних даних користувача	перевірка userId у запитах
NFR-03	Захищена передача даних	HTTPS (TLS 1.2+)
NFR-04	Захист від XSS та SQL-ін’єкцій	валідація та санітизація вводу
NFR-05	Адаптивний інтерфейс	коректне відображення від 320px

NFR-06	Відповідність стандартам доступності	WCAG 2.1 AA
NFR-07	Підтримка світлої та темної теми	перемикання теми
NFR-08	Швидкість завантаження сторінок	LCP ≤ 2.5 сек
NFR-09	Швидкість реакції інтерфейсу	INP ≤ 200 мс
NFR-10	Стабільність інтерфейсу при завантаженні	CLS ≤ 0.1
NFR-11	Збереження даних при перезавантаженні	localStorage / IndexedDB
NFR-12	Масштабованість системи	індексація БД та оптимізація запитів

Як видно з таблиці, нефункціональні вимоги спрямовані на забезпечення безпеки користувацьких даних, стабільності роботи системи та комфортної взаємодії з інтерфейсом. Вони є критично важливими для забезпечення якості програмного продукту.

Отже, сформовані вимоги визначають повний функціональний обсяг та якісні характеристики майбутнього вебдодатку. Функціональні вимоги описують логіку роботи системи та її можливості, тоді як нефункціональні забезпечують надійність, безпеку та продуктивність. Це створює основу для подальшого проєктування архітектури та реалізації програмного продукту.

2 ПРОЄКТУВАННЯ ВЕБДОДАТКУ

2.1 Архітектура системи: клієнт–серверна модель, бази даних, API

Архітектура вебдодатку «Особистий щоденник і трекер звичок» побудована на основі класичної трирівневої клієнт–серверної моделі, яка є одним із найбільш поширених підходів у проєктуванні сучасних вебсистем. Така модель дозволяє логічно розділити систему на окремі рівні відповідальності, що, у свою чергу, забезпечує масштабованість, гнучкість розвитку та спрощує супровід програмного продукту. Основною ідеєю такої архітектури є ізоляція інтерфейсу користувача, бізнес-логіки та рівня збереження даних, що дозволяє змінювати кожен із компонентів незалежно від інших.

Першим рівнем архітектури є рівень представлення (Presentation Layer), який реалізується у вигляді вебзастосунку, що працює безпосередньо у браузері користувача. Саме цей рівень відповідає за взаємодію з користувачем, відображення інтерфейсу, обробку подій та первинну валідацію введених даних. У межах системи цей рівень включає сторінки щоденника, модуль трекінгу звичок, аналітичну панель та профіль користувача. Для забезпечення високої швидкості взаємодії та зменшення кількості звернень до сервера використовується локальне кешування даних і реактивне оновлення інтерфейсу. Це дозволяє користувачу миттєво бачити зміни, не очікуючи відповіді від серверної частини.

Другим рівнем є рівень бізнес-логіки (Application Layer), який реалізується у вигляді серверного API. Саме цей рівень виконує ключову роль у системі, оскільки забезпечує обробку всіх запитів, що надходять від клієнта. Тут реалізуються механізми авторизації та автентифікації користувачів, перевірка коректності даних, виконання бізнес-правил, обчислення аналітичних показників, а також інтеграція із зовнішніми сервісами. До таких сервісів належать, зокрема, Google Calendar для синхронізації подій та email-сервіси для

надсилання сповіщень. Важливо, що саме на цьому рівні здійснюється контроль доступу до даних, що гарантує безпеку та ізоляцію інформації кожного користувача.

Третім рівнем архітектури є рівень даних (Data Layer), який відповідає за зберігання, організацію та доступ до інформації. У реалізованому прототипі застосовується локальна база даних IndexedDB, що дозволяє забезпечити офлайн-доступ до даних і швидку роботу застосунку без постійного підключення до сервера. Такий підхід є особливо ефективним на етапі розробки та тестування, оскільки забезпечує автономність системи. Водночас у промисловій (production) версії передбачається використання реляційної бази даних PostgreSQL, яка забезпечує надійне зберігання структурованих даних, підтримку складних запитів, транзакційність та масштабованість системи при зростанні кількості користувачів.

Загальну структуру взаємодії компонентів системи можна представити у вигляді наступної діаграми (рис. 2.1).

Подана діаграма ілюструє повний цикл взаємодії в системі. Користувач взаємодіє з інтерфейсом, який через шар стану або безпосередньо через API надсилає запити на сервер. Серверний рівень спочатку виконує перевірку авторизації, після чого передає запит до бізнес-логіки. Далі, залежно від типу операції, система звертається до бази даних або зовнішніх сервісів. Отриманий результат повертається назад до клієнта та відображається у користувацькому інтерфейсі.

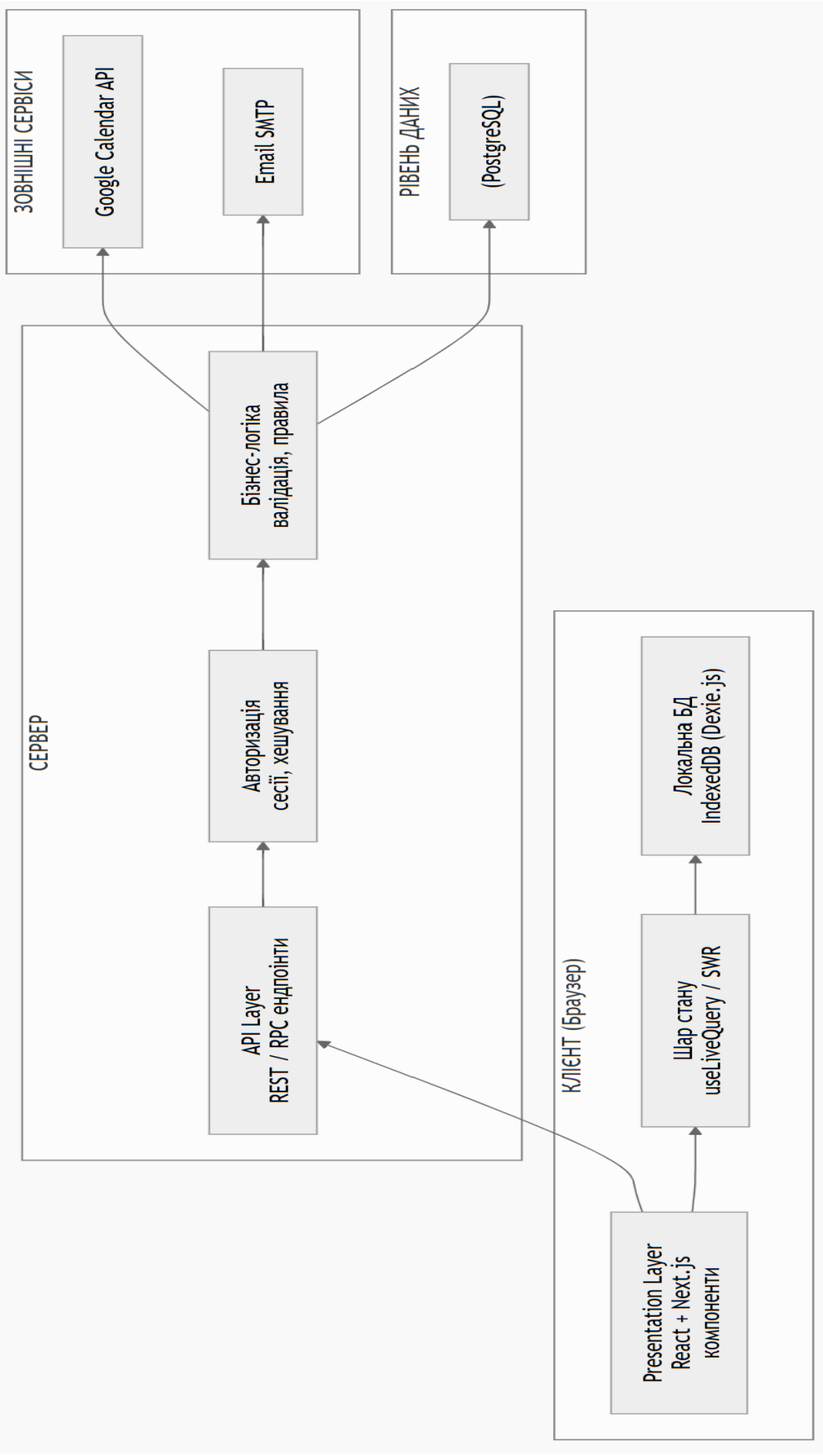


Рисунок 2.1 – Архітектура вебдодатку «Особистий щоденник і трекер звичок» (клієнт–серверна модель)

Функціональна роль API у системі є багатокомпонентною та охоплює всі ключові сценарії роботи користувача. Передусім API відповідає за авторизацію та реєстрацію користувачів, забезпечуючи захищений доступ до персональних даних. Далі воно реалізує управління записами щоденника, дозволяючи створювати, редагувати, отримувати та видаляти записи. Окремим важливим блоком є робота зі звичками: API дозволяє створювати звички, фіксувати їх виконання та зберігати історію логів. Крім того, через API формується аналітичний шар системи, який агрегує дані про активність користувача та перетворює їх у зручні показники та візуалізації. Завершальним компонентом є інтеграція із зовнішніми сервісами, такими як календарі або сервіси сповіщень, що розширює функціональність системи за межі базового застосунку.

Нижче наведено структурований опис REST API системи у вигляді таблиці, яка узагальнює основні ендпоінти, їх призначення та вимоги до авторизації (табл. 2.1). Таблиця дозволяє швидко зрозуміти логіку побудови API та визначити, які операції є публічними, а які доступні лише авторизованим користувачам.

Таблиця 2.1 – REST API ендпоінти системи

Метод і шлях	**Призначення**	**Авторизація**
POST /api/auth/register	Реєстрація нового користувача	Ні
POST /api/auth/login	Авторизація користувача та отримання сесії	Ні
GET /api/journal	Отримання списку записів щоденника	Так
POST /api/journal	Створення нового запису щоденника	Так
PUT /api/journal/:id	Оновлення існуючого запису	Так
DELETE /api/journal/:id	Видалення запису	Так
GET /api/habits	Отримання списку звичок користувача	Так
POST /api/habits	Створення нової звички	Так
POST /api/habits/:id/logs	Відмітка виконання звички за день	Так
GET /api/analytics/summary	Отримання зведеної аналітики	Так

У цій таблиці кожен рядок відображає окремий REST-ендпоінт системи. Перший стовпець визначає HTTP-метод і маршрут запиту, що вказує, яку операцію виконує клієнт. Другий стовпець містить функціональне призначення

ендпоінта, тобто опис того, яку дію він реалізує в межах системи. Третій стовпець відображає вимогу до авторизації, що дозволяє відрізнити публічні запити (наприклад, реєстрація та логін) від захищених операцій, доступних лише після автентифікації користувача.

Процес взаємодії між компонентами системи реалізується як послідовний, логічно структурований ланцюг дій, який охоплює всі рівні трирівневої архітектури – від інтерфейсу користувача до рівня збереження даних і повернення результату назад у клієнтську частину. Така організація взаємодії забезпечує передбачуваність поведінки системи, спрощує її масштабування та дозволяє чітко розмежувати відповідальність між окремими модулями.

На першому етапі користувач взаємодіє з вебзастосунком через браузер, де працює Presentation Layer. Саме цей рівень відповідає за візуалізацію даних, відображення інтерфейсних компонентів, обробку подій користувача (натискання кнопок, введення тексту, вибір параметрів) та формування первинної логіки взаємодії. У сучасній реалізації цей шар також може частково виконувати клієнтську валідацію даних, що дозволяє зменшити кількість некоректних запитів до сервера та підвищити загальну продуктивність системи.

Після дії користувача клієнтський застосунок формує HTTP-запит до API. У цьому запиті передаються необхідні параметри, ідентифікатори ресурсів та службова інформація, як правило, у форматі JSON. Використання JSON забезпечує легкість передачі даних, їх універсальність та сумісність із більшістю сучасних вебтехнологій. На цьому етапі також визначається тип операції, яка виконується, через HTTP-метод (GET, POST, PUT або DELETE), що відповідає принципам REST-архітектури.

Далі запит потрапляє на серверний рівень, де першочергово виконується перевірка авторизації та автентифікації користувача. Це дозволяє гарантувати, що доступ до даних отримують лише уповноважені користувачі. Після успішної перевірки запит передається до шару бізнес-логіки, який є центральним елементом серверної частини системи. Саме тут відбувається основна обробка даних: валідація вхідної інформації, застосування бізнес-правил, обчислення

показників (наприклад, streak у трекері звичок), формування аналітичних результатів та підготовка структурованої відповіді.

У випадках, коли операція потребує збереження або отримання даних, сервер взаємодіє з рівнем бази даних. Тут здійснюються операції читання, запису, оновлення або видалення інформації в Postgrebазі даних або локальному сховищі (залежно від конфігурації системи). Використання реляційної бази даних дозволяє забезпечити цілісність даних, підтримку зв’язків між сутностями (користувачі, записи щоденника, звички, логи виконання) та ефективне виконання аналітичних запитів.

Після завершення всіх необхідних операцій результат формується у стандартизованому форматі JSON і повертається на клієнтський рівень. На цьому етапі Presentation Layer отримує відповідь, оновлює стан інтерфейсу та відображає актуальні дані користувачу без необхідності перезавантаження сторінки. Це забезпечується завдяки використанню реактивних механізмів оновлення стану, які дозволяють динамічно змінювати інтерфейс відповідно до отриманих даних.

Отже, розглянуто архітектуру вебдодатку, побудовану за трирівневою клієнт–серверною моделлю. Визначено взаємодію між клієнтом, сервером і базою даних, а також роль REST API у забезпеченні обміну даними у форматі JSON. Запропонована архітектура забезпечує масштабованість, гнучкість та можливість подальшого розвитку системи.

2.2. Вибір технологій

Вибір технологічного стеку для вебдодатку «Особистий щоденник і трекер звичок» здійснювався на основі комплексу критеріїв, які враховують як технічні, так і практичні аспекти розробки сучасних вебзастосунків. Серед основних факторів були визначені: продуктивність і швидкодія системи, наявність статичної типізації, зрілість і популярність технологій, активність спільноти

розробників, можливість уніфікації мови програмування між клієнтською та серверною частинами, а також придатність стеку до швидкої реалізації прототипу з подальшим масштабуванням.

У результаті аналізу було сформовано цілісний стек, який охоплює всі рівні системи – від інтерфейсу користувача до бази даних і тестування. Такий підхід дозволяє забезпечити узгодженість архітектури та зменшити складність інтеграції між компонентами.

Фронтенд-частина системи реалізується на основі React 19 [21] у поєднанні з Next.js 16 [22], що є одним із найсучасніших підходів до побудови вебінтерфейсів. React забезпечує компонентну модель розробки, яка дозволяє розбивати інтерфейс на незалежні, перевикористовувані частини. Це особливо важливо для даного застосунку, який містить різні функціональні модулі: щоденник, трекер звичок та аналітичні панелі.

Next.js додатково розширює можливості React завдяки серверному рендерингу (SSR), статичній генерації сторінок (SSG) та підтримці React Server Components. Це дозволяє значно покращити продуктивність завантаження сторінок, SEO-оптимізацію та загальний користувацький досвід. Альтернативними рішеннями розглядалися Angular [23] і Vue [24], проте React було обрано завдяки його більшій гнучкості, масштабованій екосистемі та широкій підтримці спільноти, що є критично важливим для довгострокових проєктів.

Для мови програмування обрано TypeScript [25], який є строго типізованим надмножиною JavaScript. Використання TypeScript дозволяє значно зменшити кількість помилок під час розробки завдяки статичній перевірці типів ще на етапі компіляції. Крім того, він покращує читабельність коду, полегшує рефакторинг та забезпечує кращу масштабованість проєкту. Важливою перевагою є також можливість використання єдиного стеку мови як на фронтенді, так і на бекенді, що суттєво спрощує взаємодію між компонентами системи.

Для стилізації інтерфейсу обрано комбінацію Tailwind CSS v4 та бібліотеки shadcn/ui. Tailwind CSS [26] реалізує підхід utility-first, який дозволяє створювати

дизайн без написання великої кількості кастомних CSS-стилів. Це значно прискорює процес розробки та забезпечує консистентність інтерфейсу. Бібліотека shadcn/ui [27], у свою чергу, надає набір готових, доступних та адаптивних компонентів, побудованих поверх сучасних стандартів UI-дизайну.

Для візуалізації даних у системі використовується бібліотека Recharts [28], яка дозволяє створювати інтерактивні графіки на основі React-компонентів. Вона забезпечує декларативний підхід до побудови візуалізацій, що спрощує інтеграцію графіків у вже існуючу архітектуру інтерфейсу.

Окрему роль у системі відіграє локальне збереження даних, яке реалізовано через IndexedDB [29] із обгорткою Dexie.js. IndexedDB є повноцінною клієнтською базою даних, що працює безпосередньо у браузері та дозволяє зберігати значні обсяги структурованих даних. Використання Dexie.js спрощує роботу з цією базою, надаючи зручний API для виконання запитів та індексації. На відміну від localStorage, який має обмежений обсяг і синхронну природу, IndexedDB підтримує асинхронні операції, що позитивно впливає на продуктивність і дозволяє реалізувати офлайн-режим роботи застосунку.

На серверному рівні як основна технологія обрано Node.js [30], що дозволяє використовувати JavaScript/TypeScript для розробки бекенд-логіки. Це забезпечує єдину мову програмування для всього стеку, що значно спрощує розробку, тестування та підтримку системи. Node.js також базується на асинхронній неблокуючій моделі виконання, що робить його ефективним для обробки великої кількості одночасних запитів.

Для збереження даних у серверній частині використовується PostgreSQL [31], яка є однією з найнадійніших і найпотужніших реляційних баз даних. Вона підтримує складні SQL-запити, транзакційність, індексацію, а також роботу з JSON-структурами, що є важливим для аналітичних модулів та гнучкого зберігання даних користувачів.

Окремо варто відзначити систему тестування, де використовується зв’язка Jest + ts-jest [32], яка дозволяє виконувати модульне тестування безпосередньо у середовищі TypeScript. Це забезпечує контроль якості коду, дозволяє перевіряти

бізнес-логіку ізольовано та зменшує ризики появи помилок під час масштабування системи.

У підсумку, обраний технологічний стек формує цілісну TypeScript-орієнтовану екосистему, у якій фронтенд, бекенд і частково логіка даних працюють у єдиному мовному середовищі. Це забезпечує високу узгодженість компонентів, спрощує розробку, підвищує продуктивність команди та створює основу для подальшого масштабування вебдодатку без необхідності кардинальної зміни архітектури.

2.3 Моделювання даних: структура записів щоденника, модуль звичок, користувацькі профілі

Модель даних системи вебдодатку «Особистий щоденник і трекер звичок» побудована на основі чотирьох основних сутностей, між якими реалізовано класичні зв’язки типу «один-до-багатьох». Центральною сутністю виступає користувач (User), який є власником усіх персональних даних у системі. Кожен користувач може створювати множину записів щоденника (JournalEntry) та набір звичок (Habit), тоді як кожна звичка додатково має власний журнал виконань (HabitLog), що дозволяє відстежувати динаміку поведінки в часі. Така структура забезпечує логічну ізоляцію даних між користувачами та підтримує масштабованість системи при зростанні кількості записів.

Записи щоденника представлені сутністю JournalEntry (табл. 2.2), яка призначена для збереження текстової рефлексії користувача, його емоційного стану та контекстної інформації. Кожен запис має унікальний ідентифікатор та обов’язково прив’язується до конкретного користувача через зовнішній ключ userId. Дата запису зберігається у форматі YYYY-MM-DD, що дозволяє ефективно виконувати сортування та фільтрацію за часовими проміжками.

Основними змістовими полями є title та content, які відповідно відповідають за заголовок і основний текст запису. Додатково система підтримує

поле mood, яке реалізоване у вигляді переліку (enum) значень – від great до awful, що дозволяє формалізувати емоційний стан користувача та використовувати його в аналітиці. Для розширеної класифікації записів використовується масив tags, який дозволяє групувати записи за темами або подіями.

Також у структурі передбачено технічні поля createdAt та updatedAt, які зберігають часові мітки створення та останнього оновлення запису. Це дозволяє реалізувати історичність змін та відстеження активності користувача в системі.

Таблиця 2.2 – Сутність JournalEntry

Поле	Тип	Обмеження	Опис
id	number	PK, auto	Унікальний ідентифікатор запису
userId	number	FK → User	Ідентифікатор власника запису
date	string (YYYY-MM-DD)	indexed	Дата створення запису
title	string	not null	Заголовок запису
content	string	–	Основний текст щоденника
mood	enum	great / good / neutral / bad / awful	Емоційна оцінка стану користувача
tags	string[]	–	Набір тематичних тегів для фільтрації
createdAt	number	timestamp	Час створення запису
updatedAt	number	timestamp	Час останнього оновлення

Модуль звичок реалізовано через дві взаємопов’язані сутності: Habit та HabitLog, що дозволяє розділити опис самої звички та факти її виконання (табл. 2.3). Така структура є важливою для коректного трекінгу поведінки користувача у часі.

Сутність Habit описує саму звичку як об’єкт планування. Вона містить базові атрибути, такі як назва звички (name), візуальні параметри (icon, color), а також логічні характеристики, зокрема frequency, яка визначає частоту виконання (щоденна або щотижнева), та targetPerWeek, що задає цільову кількість виконань за тиждень. Поле archived використовується для приховування неактивних звичок без їх видалення, що дозволяє зберігати історичні дані.

Фактичне виконання звичок фіксується у сутності HabitLog, яка виступає журналом подій. Кожен запис містить посилання на відповідну звичку через habitId та дату виконання.

Для запобігання дублюванню даних у системі введено складений унікальний індекс [habitId + date], що гарантує неможливість повторного відмічання однієї і тієї ж звички в межах одного дня. Це є важливим елементом цілісності даних і коректності аналітики.

Таблиця 2.3 – Модуль звичок (Habit + HabitLog)

Сутність	Поле	Тип	Опис
Habit	id	number (PK)	Унікальний ідентифікатор звички
Habit	userId	number (FK)	Власник звички
Habit	name	string	Назва звички
Habit	icon	string	Іконка для візуального представлення
Habit	color	string	Колір інтерфейсного маркування
Habit	frequency	enum (daily/weekly)	Частота виконання
Habit	targetPerWeek	number	Цільова кількість виконань на тиждень
Habit	archived	boolean	Статус архівації звички
HabitLog	id	number (PK)	Ідентифікатор запису виконання
HabitLog	habitId	number (FK)	Посилання на звичку
HabitLog	date	string (YYYY-MM-DD)	Дата виконання
HabitLog	unique	habitId + date	Унікальність виконання за день

Сутність User є базовою в системі та містить інформацію про зареєстрованого користувача (табл. 2.4). Вона включає унікальний ідентифікатор id, електронну пошту email (яка є унікальною і використовується як логін), а також passwordHash, що забезпечує безпечне зберігання паролю у зашифрованому вигляді.

Додатково профіль користувача містить поля для персоналізації системи, зокрема name, avatarUrl, timezone та theme. Поле timezone є особливо важливим для коректної роботи нагадувань і відображення аналітики у локальному часі користувача, тоді як theme дозволяє зберігати обраний режим інтерфейсу (світлий або темний).

Таблиця 2.4 – Користувацькі профілі (User)

Поле	Тип	Опис
id	number (PK)	Унікальний ідентифікатор користувача
email	string (unique)	Електронна адреса (логін)
passwordHash	string	Хешований пароль користувача
name	string	Ім’я користувача
avatarUrl	string	Посилання на аватар профілю
timezone	string	Часовий пояс для коректних нагадувань
theme	enum (light / dark)	Обрана тема інтерфейсу

Загалом модель даних побудована таким чином, щоб забезпечити баланс між простотою структури та достатньою гнучкістю для реалізації складних сценаріїв використання. В основі архітектури лежить принцип чіткого розділення сутностей, де кожна з них виконує свою окрему роль і відповідає за конкретний аспект предметної області: користувачі зберігають персональні дані, записи щоденника фіксують рефлексивний контент, а модуль звичок забезпечує довготрекінг поведінкових патернів.

Використання зв’язків типу «один-до-багатьох» дозволяє природно відобразити реальні сценарії використання системи: один користувач може мати багато записів щоденника та звичок, а кожна звичка, у свою чергу, може містити численні записи виконання. Така структура забезпечує логічну узгодженість даних і спрощує їх обробку як на рівні запитів, так і під час побудови аналітики.

Додатково модель орієнтована на масштабованість: вона дозволяє без суттєвих змін у структурі розширювати функціональність системи, наприклад, додаючи нові типи аналітики, категоризацію записів або розширені параметри звичок. Завдяки індексації ключових полів (таких як дата, userId та habitId) забезпечується ефективний доступ до даних навіть при значному їх обсязі, що позитивно впливає на продуктивність вебдодатку.

Таким чином, запропонована модель даних є достатньо простою для реалізації, але водночас потужною для підтримки складної логіки самоменеджменту, аналітики та персоналізованої взаємодії з користувачем.

2.4 UML-діаграми

У межах проєктування вебдодатку «Особистий щоденник і трекер звичок» було виконано моделювання поведінки системи за допомогою UML-діаграм, що дозволяє формалізувати ключові сценарії використання, структуру даних та логіку взаємодії між компонентами. Використання UML забезпечує більш наочне представлення архітектури системи та сприяє кращому розумінню взаємозв'язків між її елементами ще на етапі проєктування, що є особливо важливим для складних інформаційних систем з декількома функціональними модулями.

Першою розглядається діаграма прецедентів (Use Case Diagram). Вона відображає загальну структуру взаємодії користувача із системою та демонструє перелік основних функцій, доступних у межах вебзастосунку (рис. 2.2). Центральним актором виступає користувач, який взаємодіє із системою через набір визначених сценаріїв. На рівні базової функціональності передбачено процеси реєстрації та авторизації, що забезпечують доступ до персоналізованого середовища. Після входу в систему користувач отримує можливість працювати з модулями щоденника та звичок, зокрема створювати, редагувати та видаляти записи щоденника, а також формувати та налаштовувати власні звички.

Окремо в діаграмі відображено функціонал відмітки виконання звичок, який є ключовим елементом трекінгу поведінки користувача. Саме цей сценарій забезпечує накопичення даних для подальшої аналітики та формування статистичних показників. Крім того, система передбачає перегляд аналітики прогресу, що дозволяє користувачеві оцінювати власну активність у динаміці та відстежувати рівень дотримання звичок.

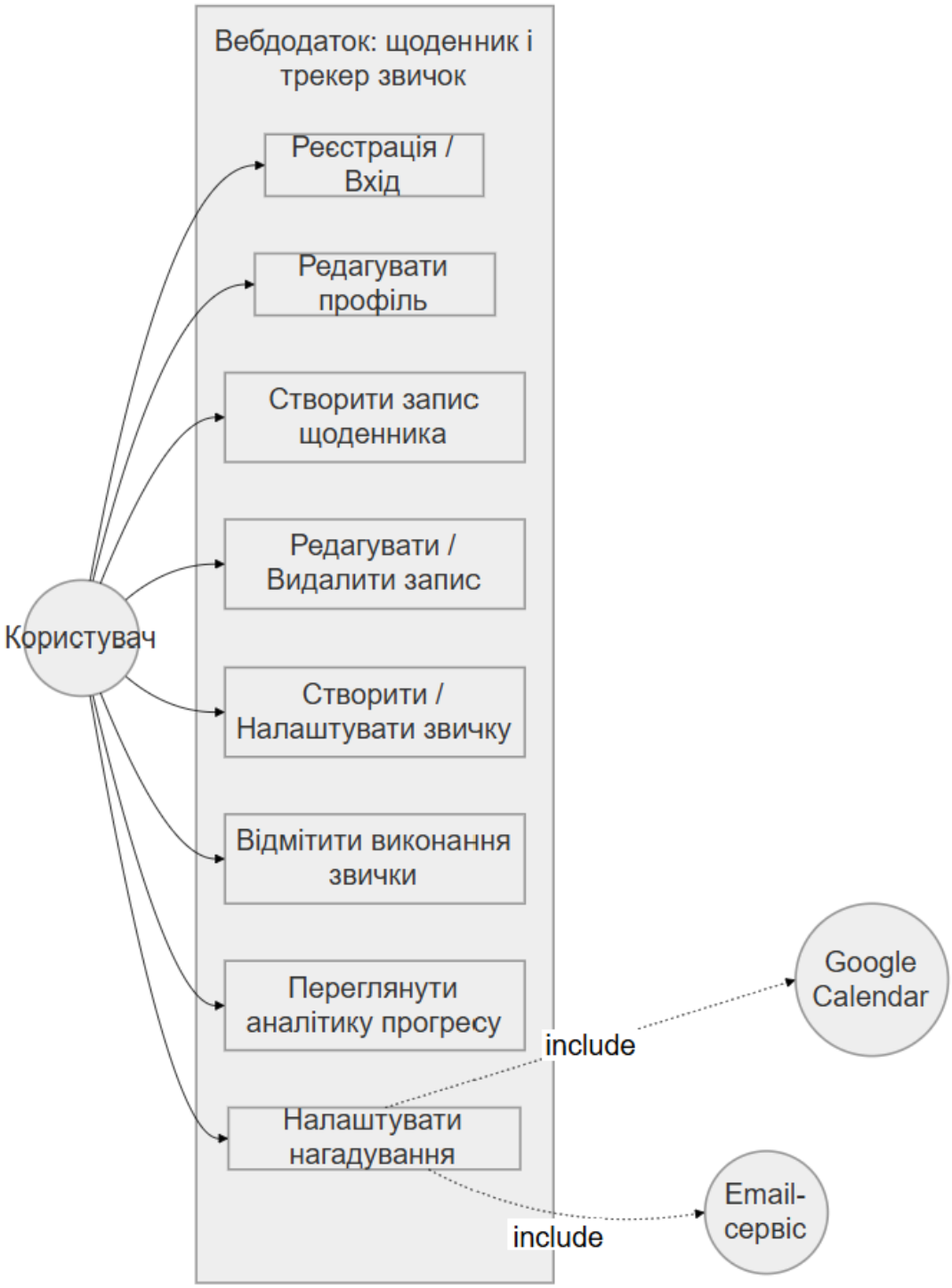


Рисунок 2.2 – Діаграма сценаріїв використання вебдодатку “Щоденник і трекер звичок”

Важливим доповненням є інтеграційні сценарії, пов’язані з налаштуванням нагадувань. У цьому контексті система взаємодіє із зовнішніми сервісами, зокрема Google Calendar та Email-сервісом, що дозволяє автоматизувати процес інформування користувача про необхідність виконання запланованих дій. Таким

чином, діаграма прецедентів демонструє не лише внутрішню функціональність системи, а й її здатність до інтеграції із зовнішнім середовищем.

Діаграма класів (Class Diagram), представлена на рисунку 2.3, формалізує логічну структуру даних системи, визначаючи основні сутності, їхні атрибути, методи та зв'язки між ними. Вона відображає об'єктно-орієнтовану модель предметної області та є ключовою для розуміння того, як саме інформація зберігається, пов'язується та обробляється в межах застосунку.

У центрі моделі знаходиться сутність User, яка виступає базовим об'єктом системи та описує користувача вебдодатку. Вона містить такі атрибути, як ідентифікатор користувача, електронна пошта, хеш паролю, ім'я, часовий пояс та налаштування теми інтерфейсу. Саме ця сутність є відправною точкою для всіх інших зв'язків у системі, оскільки кожен користувач є власником власних даних у межах застосунку.

З користувачем у відношенні типу «один-до-багатьох» пов'язана сутність JournalEntry, яка моделює записи щоденника. Це означає, що один користувач може створювати необмежену кількість записів, тоді як кожен окремий запис належить лише одному користувачу. Клас JournalEntry містить структуровану інформацію про запис, зокрема дату створення, заголовок, текстовий вміст, емоційний стан (mood), а також набір тегів для категоризації. Таке структурування дозволяє не лише зберігати дані, але й забезпечує можливість їх подальшого аналізу та фільтрації.

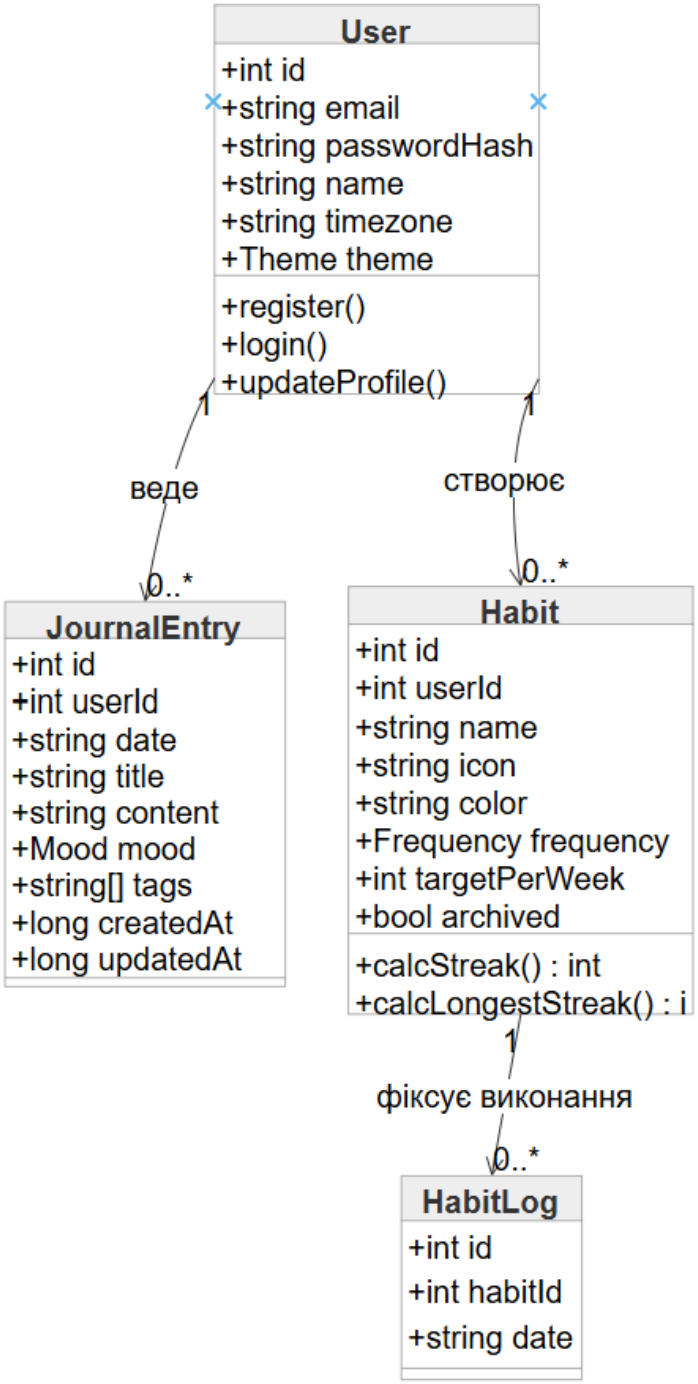


Рисунок 2.3 – Діаграма класів структури даних вебдодатку “Щоденник і трекер звичок”

Наступною важливою сутністю є Habit, яка відповідає за модуль управління звичками користувача. Як і у випадку зі щоденником, між User та Habit встановлено зв’язок «один-до-багатьох», що відображає можливість створення користувачем декількох звичок. Клас Habit містить атрибути, які описують параметри звички: її назву, іконку, колір для візуального

представлення в інтерфейсі, частоту виконання (наприклад, щоденну або щотижневу), цільову кількість виконань на тиждень, а також статус архівації. Крім того, у класі визначено методи для обчислення аналітичних показників, зокрема поточної серії виконань (calcStreak) та найдовшої серії (calcLongestStreak), що дозволяє реалізувати базову поведінкову аналітику безпосередньо на рівні моделі.

Для фіксації факту виконання звичок використовується сутність HabitLog, яка знаходиться у відношенні «один-до-багатьох» із класом Habit. Це означає, що одна звичка може мати багато записів виконання, кожен з яких відповідає конкретній даті. Основними атрибутами HabitLog є ідентифікатор запису, посилання на відповідну звичку та дата виконання. Важливою особливістю цієї сутності є наявність складеного унікального обмеження (habitId + date), яке запобігає дублюванню записів і гарантує, що одна й сама звичка не може бути відмічена більше одного разу за один день.

Діаграма послідовності описує динамічний аспект роботи системи та демонструє один із ключових сценаріїв використання – процес відмітки виконання звички користувачем і подальший перерахунок прогресу (рис. 2.4). Цей сценарій є центральним для трекера звичок, оскільки саме він формує аналітичні показники та візуальний прогрес у системі.

Процес починається з того, що користувач взаємодіє з інтерфейсом модуля звичок (HabitsView) і натискає кнопку «Виконано сьогодні». У відповідь на цю дію клієнтський застосунок формує запис у локальній базі даних IndexedDB (через Dexie.js), додаючи новий елемент у сутність HabitLog із зазначенням ідентифікатора звички та дати виконання.

Після успішного збереження запису база даних повертає підтвердження, і інтерфейс автоматично реагує завдяки реактивному механізму useLiveQuery, який відстежує зміни у даних і миттєво отримує оновлений масив записів виконання. Це забезпечує синхронне оновлення стану без необхідності ручного перезавантаження сторінки або повторних запитів.

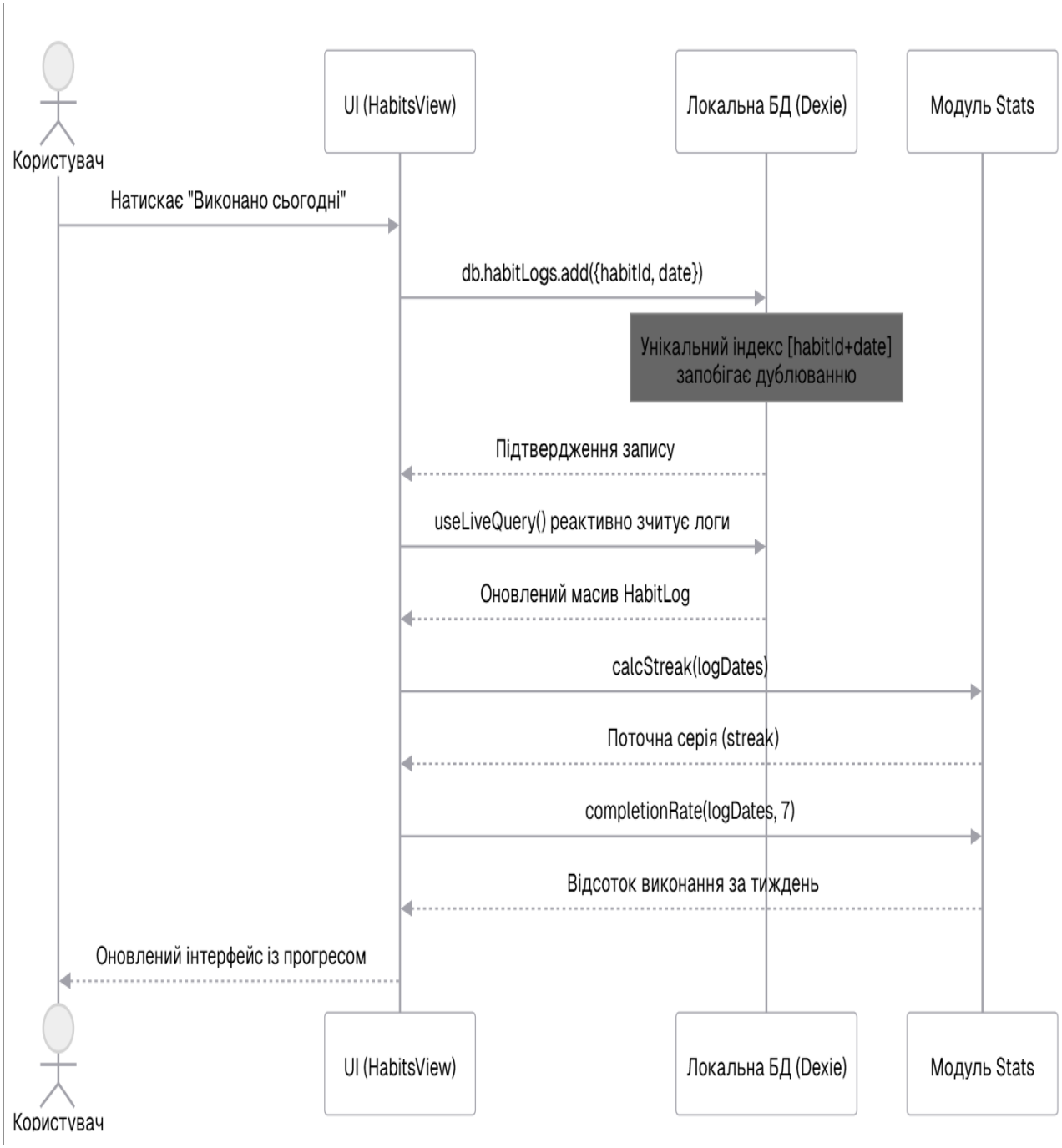


Рисунок 2.4 – Діаграма послідовності

Далі дані передаються до модуля статистики, де виконуються обчислення ключових метрик. Зокрема, функція calcStreak() визначає поточну серію безперервного виконання звички, що є важливим мотиваційним показником. Паралельно функція completionRate() обчислює відсоток виконання звички за останні сім днів, що дозволяє оцінити регулярність поведінки користувача.

На завершальному етапі отримані аналітичні показники повертаються до інтерфейсу, де відбувається оновлення візуальних компонентів прогресу. Таким чином користувач одразу бачить результат своєї дії у вигляді оновленої статистики, що підсилює інтерактивність системи.

У підсумку дана діаграма ілюструє реактивний характер роботи додатку, де кожна дія користувача миттєво відображається у стані системи, а обчислення прогресу виконується автоматично, що забезпечує високу швидкість відгуку та зручність використання.

Отже, UML-діаграми дозволили формалізувати структуру, сценарії використання та взаємодію компонентів системи «Особистий щоденник і трекер звичок». Вони забезпечили наочне представлення архітектури, що спростило проєктування та узгодження логіки роботи додатку перед реалізацією.

3 РЕАЛІЗАЦІЯ ВЕБДОДАТКУ

3.1 Реалізація інтерфейсу користувача: створення та редагування записів, додавання звичок

Реалізація інтерфейсу користувача вебдодатку «Особистий щоденник і трекер звичок» базується на компонентній архітектурі React із використанням Next.js та UI-бібліотеки shadcn/ui. Основний принцип побудови інтерфейсу полягає у поділі функціональності на логічні модулі, що дозволяє забезпечити просту навігацію, чітку структуру взаємодії та мінімальне перевантаження користувача інформацією. Уся система інтерфейсу організована навколо головного компонента застосунку та спеціалізованих представлень для роботи з даними.

Центральним елементом системи є компонент DiaryApp, який виконує роль каркасу всього застосунку (Додаток А.1). Саме в ньому ініціалізується базова логіка роботи застосунку, включаючи підготовку локальної бази даних та налаштування користувацького середовища.

Під час першого завантаження застосунку виконується функція seedIfEmpty(), яка перевіряє наявність даних у локальному сховищі та, у разі його порожності, ініціалізує базову структуру даних. Після завершення цієї операції система переходить у стан готовності, що дозволяє відобразити основний інтерфейс користувачу.

Окрему увагу приділено реалізації підтримки темної та світлої теми. У компоненті реалізовано зчитування попередньо збережених налаштувань із localStorage, а також визначення системних уподобань користувача через matchMedia. Зміна теми виконується функцією toggleTheme, яка не лише перемикає стан інтерфейсу, але й синхронізує його з локальним сховищем, забезпечуючи збереження вибору між сеансами роботи (Додаток А.1).

Інтерфейс побудовано у вигляді трьох основних вкладок, реалізованих через компонент Tabs:

- Прогрес – відображення аналітичних даних;
- Звички – керування звичками та їх виконанням;
- Щоденник – створення та редагування записів.

Така структура забезпечує логічне розділення функціональності та дозволяє користувачу швидко перемикатися між різними аспектами роботи з даними без перевантаження інтерфейсу.

Функціональність роботи зі щоденником реалізовано у компоненті JournalView, який відповідає за повний життєвий цикл записів: створення, відображення, редагування та видалення (Додаток А.2).

Отримання даних здійснюється за допомогою реактивного хука useLiveQuery із бібліотеки Dexie, що забезпечує автоматичну синхронізацію інтерфейсу з локальною базою даних IndexedDB. Завдяки цьому будь-які зміни у структурі даних миттєво відображаються у користувацькому інтерфейсі без необхідності ручного оновлення сторінки.

Записи щоденника відображаються у вигляді карток, кожна з яких містить ключову інформацію: заголовок запису, дату, емоційний стан (настрій), текстовий вміст та набір тегів. Візуалізація настрою реалізована через комбінацію емодзі та кольорового індикатора, що дозволяє швидко сприймати емоційний контекст кожного запису.

Для забезпечення зручності взаємодії з даними у кожній картці передбачено набір швидких дій. Користувач може редагувати або видаляти запис за допомогою відповідних кнопок-іконок, що інтегровані безпосередньо в елемент інтерфейсу.

Операції створення та редагування записів реалізовані через окремий компонент EntryForm, який відкривається у модальному вікні (Dialog). Такий підхід дозволяє ізолювати процес введення даних від основного інтерфейсу, зберігаючи його чистоту та зручність використання.

Форма містить набір структурованих полів, що забезпечують повне наповнення запису: заголовок, дата, вибір настрою, основний текст та теги. Вибір настрою реалізовано у вигляді інтерактивних кнопок, що відображають емоційні стани користувача за допомогою емодзі та кольорових позначок.

Перед збереженням дані проходять попередню обробку. Теги, введені у вигляді текстового рядка, перетворюються у масив значень, очищаються від зайвих символів і порожніх елементів. Додатково до кожного запису додаються часові мітки створення та оновлення, що дозволяє відстежувати історію змін.

Залежно від контексту використання реалізовано два сценарії роботи: створення нового запису або редагування існуючого. У першому випадку дані додаються до бази через db.journal.add, у другому – оновлюються за допомогою db.journal.update. Після успішного виконання операції модальне вікно закривається, а інтерфейс автоматично оновлюється завдяки реактивній природі useLiveQuery.

У межах реалізації користувацького інтерфейсу особливу увагу приділено модулю керування звичками, який реалізовано у компоненті HabitsView (Додаток А.3). Його основне призначення полягає у забезпеченні повного циклу роботи зі звичками користувача – від створення та налаштування до відстеження виконання та аналізу прогресу. Архітектурно компонент побудований на основі реактивного підходу з використанням useLiveQuery, що забезпечує автоматичну синхронізацію інтерфейсу з локальною базою даних без необхідності ручного оновлення сторінки.

Під час ініціалізації компонента відбувається одночасне отримання двох основних наборів даних: списку звичок із таблиці habits та логів виконання із таблиці habitLogs. На основі цих даних формується актуальний стан інтерфейсу, при цьому активними вважаються лише ті звички, які не були архівовані. Додатково визначається поточна дата у форматі toDateString, що дозволяє коректно перевіряти виконання звички саме за сьогоднішній день і формувати відповідний стан інтерфейсу.

Відображення звичок реалізовано у вигляді карток, кожна з яких містить основну інформацію про звичку: назву, опис, піктограму та кольорове позначення. Для цього використовуються конфігураційні структури HABIT_ICONS та HABIT_COLORS, які забезпечують уніфіковану візуальну систему і дозволяють користувачу швидко ідентифікувати різні звички за кольором та символом. Окрім візуального представлення, кожна звичка супроводжується аналітичними показниками, які розраховуються в режимі реального часу за допомогою допоміжних функцій модуля статистики, зокрема calcStreak для визначення поточної серії виконання та completionRate для обчислення відсотка виконання за останні 30 днів.

Ключовим функціональним елементом є механізм відмітки виконання звички за поточний день, який реалізовано через функцію toggleToday. Її логіка базується на перевірці наявності відповідного запису у таблиці habitLogs. Якщо запис уже існує, він видаляється, що означає скасування виконання, якщо ж запис відсутній – створюється новий лог із прив’язкою до конкретної звички та дати. Таким чином забезпечується ідемпотентна поведінка операції, при якій повторне натискання не створює дублюючих записів, а лише змінює стан виконання.

Окремим елементом інтерфейсу є мініатюрна теплова карта активності за останні сім днів, яка формується на основі функції lastNDays. Для кожного дня перевіряється наявність виконання у відповідному наборі логів, і результат візуалізується за допомогою кольорового кодування: виконані дні відображаються у фірмовому кольорі звички, тоді як пропущені – у нейтральному відтінку. Такий підхід дозволяє користувачу швидко оцінити свою регулярність без необхідності аналізу числових показників.

Додатково в кожній картці реалізовано інтерактивну кнопку відмітки виконання, яка змінює свій вигляд залежно від стану звички за поточний день. У разі виконання вона відображається як підтверджена дія, а у випадку невиконання – пропонує користувачу виконати відповідну дію. Це створює

ефект мікроінтеракцій, які підвищують інтуїтивність інтерфейсу та зменшують когнітивне навантаження.

Створення нових звичок реалізовано через модальний компонент HabitForm, який відкривається у діалоговому вікні (Dialog). Такий підхід відповідає загальній концепції застосунку, де всі операції введення та редагування даних винесені в окремі модальні інтерфейси, що дозволяє зберігати основний екран максимально чистим. У формі користувач послідовно задає назву та опис звички, після чого обирає її візуальне представлення у вигляді іконки з набору HABIT_ICONS та колір із палітри HABIT_COLORS.

Наступним кроком є налаштування частоти виконання звички. Користувач може обрати щоденний або щотижневий режим, причому у випадку щотижневого режиму додатково задається цільова кількість виконань на тиждень. Після заповнення всіх параметрів дані зберігаються у локальну базу через метод db.habits.add, після чого інтерфейс автоматично оновлюється завдяки реактивності useLiveQuery, що виключає потребу ручного перерендерингу.

Окремо реалізовано функціонал видалення звичок через компонент DeleteHabitButton. При видаленні система не лише видаляє саму звичку, але й очищає всі пов’язані записи у таблиці habitLogs, що забезпечує цілісність даних і запобігає накопиченню застарілої інформації в системі. Такий підхід відповідає принципам каскадного видалення у локальних сховищах даних.

Таким чином, модуль керування звичками забезпечує повноцінну інтерактивну систему трекінгу поведінки користувача, поєднуючи зручний інтерфейс, реактивне оновлення даних та базову аналітику у вигляді візуальних індикаторів і статистичних показників.

Окремим логічним продовженням інтерфейсу є модуль аналітики ProgressView (Додаток А.4), який відповідає за узагальнення всіх даних системи та їх візуалізацію. Його головна функція полягає у перетворенні сирих даних про звички та записи щоденника у зрозумілі користувачу показники та графіки.

На етапі ініціалізації компонент отримує дані про звички, логи виконання та записи щоденника через useLiveQuery, після чого формується набір ключових метрик. До них належать кількість виконань за поточний день, найдовша серія виконання звичок, загальна кількість відміток та кількість записів у щоденнику. Ці значення відображаються у вигляді KPI-карток, що дозволяє користувачу швидко оцінити загальний рівень своєї активності.

Для візуалізації динаміки використання системи застосовується бібліотека Recharts, яка забезпечує побудову інтерактивних графіків. Зокрема, стовпчикова діаграма відображає рівень виконання звичок за тиждень, дозволяючи оцінити стабільність поведінки користувача у короткостроковому періоді. Додатково реалізовано лінійний графік тренду настрою за останні 14 днів, що демонструє емоційну динаміку та дозволяє простежити зміни психологічного стану користувача у часі.

Окремим елементом є теплова карта активності звичок за 35 днів, яка відображає регулярність виконання кожної звички у вигляді матриці. Така візуалізація дозволяє швидко виявляти періоди стабільності або пропусків, що є важливим для аналізу довгострокової поведінки.

Завершальним компонентом є блок розподілу настрою, який відображає статистичну структуру емоційних станів користувача у вигляді прогрес-барів. Це дозволяє отримати узагальнене уявлення про емоційний фон користувача протягом усього періоду використання системи.

У підсумку, реалізація модуля ProgressView у поєднанні з HabitsView формує повноцінну систему аналітики, яка забезпечує користувача не лише інструментами фіксації даних, але й засобами їх глибокого аналізу та візуального інтерпретування.

У межах підрозділу було реалізовано інтерфейс користувача вебдодатку на основі компонентної архітектури React, який забезпечує зручну взаємодію з даними щоденника, звичок і аналітики. Використання вкладкової навігації та модальних вікон дозволило структуровано розподілити функціональність і мінімізувати навантаження на основний екран. Реалізовані компоненти

забезпечують створення, редагування та видалення записів і звичок, а також їхню візуалізацію та базову аналітику в режимі реального часу. Загалом інтерфейс є інтуїтивним, реактивним і зручним для користувача, що підвищує ефективність роботи з системою.

3.2 Механізм трекінгу звичок: щоденні/тижневі нагадування, статус виконання

Механізм трекінгу звичок у вебдодатку реалізовано як реактивну систему фіксації виконання дій користувача з подальшим розрахунком статистичних показників та формуванням логіки нагадувань. Основою збереження стану виконання є таблиця habitLogs, у якій фіксується кожне виконання звички з прив’язкою до конкретної дати та ідентифікатора звички. Така структура дозволяє однозначно відстежувати історію активності користувача та уникати дублювання записів.

Для забезпечення коректної роботи системи реалізовано унікальну логіку відмітки виконання у вигляді toggle-механізму. Це означає, що повторне натискання на кнопку зміни стану не створює новий запис, а навпаки – видаляє вже існуючий, забезпечуючи тим самим перемикання стану між «виконано» та «не виконано». Такий підхід реалізовано у функції toggleToday, яка перевіряє наявність запису у habitLogs за комбінацією habitId та поточної дати, після чого або створює новий запис, або видаляє існуючий (Додаток А.3).

```
async function toggleToday(habit: Habit) {
  if (!habit.id) return
  const existing = await db.habitLogs.where({ habitId: habit.id, date: today }).first()
  if (existing?.id) {
    await db.habitLogs.delete(existing.id)
```

```
  } else {
    await db.habitLogs.add({
      habitId: habit.id,
      date: today,
      createdAt: Date.now()
    })
  }
}
```

Завдяки такій реалізації система гарантує ідемпотентність операції та запобігає дублюванню даних, що є критично важливим для точного розрахунку статистики.

Після фіксації виконання звички інтерфейс автоматично оновлює стан картки, відображаючи актуальний статус користувача. Кожна звичка супроводжується набором візуальних та аналітичних показників, які дозволяють оцінити ефективність її виконання. Зокрема, у картці звички відображається поточна серія виконань (streak), яка розраховується функцією calcStreak і показує кількість днів поспіль без пропусків. Також використовується показник completionRate, що визначає відсоток виконання звички за останні 30 днів, що дозволяє оцінити стабільність поведінки користувача у середньостроковій перспективі.

Окремим елементом інтерфейсу є мініатюрна теплова карта останніх семи днів, яка генерується на основі функції lastNDays. Вона візуально відображає регулярність виконання звички, де кожен день позначається як виконаний або пропущений. Такий підхід дозволяє користувачу швидко оцінити власну дисципліну без аналізу числових даних.

Окремим підсистемним елементом є логіка нагадувань (рис. 3.1), яка базується на параметрах frequency та targetPerWeek, що задаються при створенні звички. У системі передбачено два основні типи поведінки: щоденний режим (daily) та тижневий режим (weekly). У випадку щоденних звичок система очікує

від користувача відмітку виконання кожного дня, тоді як для тижневих звичок формується цільова кількість виконань у межах тижня.

На логічному рівні це дозволяє формувати різні сценарії нагадувань: щоденні звички ініціюють регулярні сповіщення щодня, тоді як тижневі активують нагадування до моменту досягнення встановленого targetPerWeek.

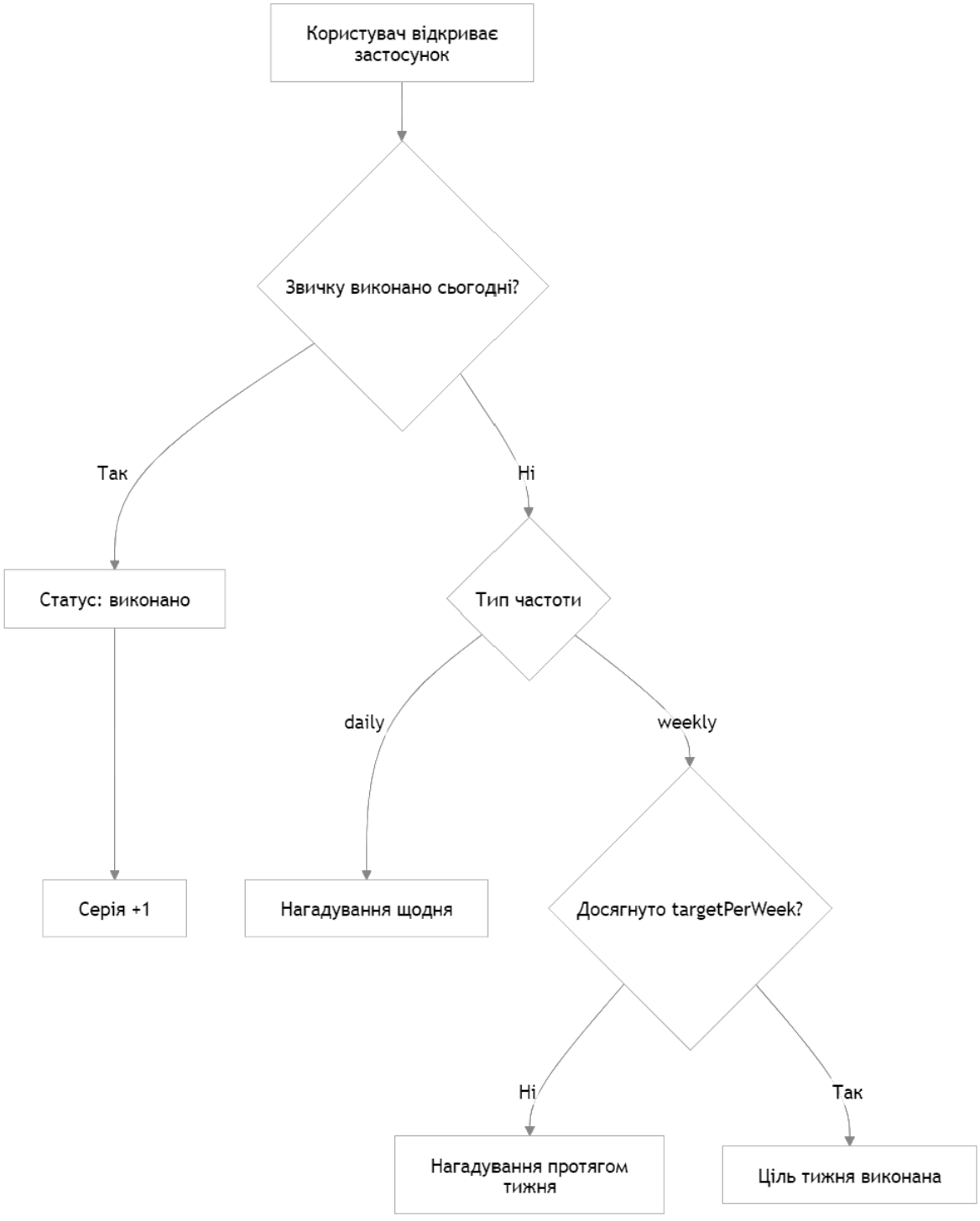


Рисунок 3.1 – Щоденні / тижневі нагадування

Діаграма відображає загальну логіку роботи механізму трекінгу звичок у системі, тобто процес перевірки виконання звички та формування нагадувань залежно від її типу.

Після відкриття застосунку система перевіряє, чи була звичка виконана сьогодні. Якщо виконання вже зафіксоване, звичка отримує статус «виконано», а також оновлюється показник серії, тобто кількість днів поспіль, протягом яких користувач регулярно виконує цю звичку. Це дозволяє відстежувати стабільність формування поведінкового патерну.

Якщо звичка ще не виконана, система переходить до аналізу її типу частоти. У випадку щоденної звички активується режим щоденних нагадувань, які стимулюють користувача виконати дію протягом поточного дня.

Для звичок із тижневою частотою додатково перевіряється, чи досягнуто встановленої цільової кількості виконань за тиждень. Якщо ціль ще не виконана, система продовжує надсилати нагадування протягом тижня, підтримуючи мотивацію користувача.

Якщо ж необхідну кількість виконань уже досягнуто, тижнева ціль вважається виконаною, і подальші нагадування для цього циклу припиняються або переходять у наступний період.

Отже, реалізовано механізм трекінгу звичок, який забезпечує фіксацію виконання дій користувача, обчислення поточного стану звички та формування адаптивних нагадувань.

Завдяки використанню toggle-логіки та унікальних записів за датою система гарантує коректність відміток виконання, а також дозволяє відстежувати серії та рівень регулярності. Реалізований підхід забезпечує гнучке управління як щоденними, так і тижневими звичками, що підвищує ефективність самоконтролю користувача та підтримує стабільність формування корисних поведінкових патернів.

3.3 Візуалізація прогресу: графіки, діаграми, календарі

Важливою складовою вебдодатку «Особистий щоденник і трекер звичок» є модуль візуалізації прогресу, реалізований у компоненті ProgressView (Додаток А.4). Його основним призначенням є перетворення накопичених даних про звички та записи щоденника у зрозумілі графічні та числові показники, що дозволяють користувачу аналізувати власну активність, оцінювати результати та відстежувати динаміку змін у часі.

Для побудови графіків використовується бібліотека Recharts, яка забезпечує створення інтерактивних діаграм у середовищі React. Джерелом інформації виступають дані, що зберігаються у локальній базі IndexedDB та отримуються через реактивний механізм useLiveQuery. Підготовка наборів даних виконується за допомогою спеціалізованих функцій модуля статистики (stats), які здійснюють необхідні обчислення та формують структури даних для подальшої візуалізації.

На початковому етапі роботи компонент завантажує три основні категорії інформації: список звичок, журнал їх виконання та записи користувача у щоденнику. Після отримання даних система автоматично формує набір ключових показників ефективності (KPI), які відображаються у верхній частині сторінки у вигляді інформаційних карток.

Для розрахунку KPI використовуються чотири основні метрики, представлені в таблиці 3.1.

Таблиця 3.1 – Основні елементи візуалізації прогресу

Елемент візуалізації	Тип	Джерело даних
KPI-картки	Числові показники	doneToday, bestStreak, totalCompletions, journalCount
Виконання за тиждень	Стовпчиковий графік (BarChart)	weeklyCompletionData(active, logs)
Тренд настрою	Лінійний графік (LineChart)	moodTrendData(entries, 14)
Карта активності	Теплова карта (35 днів)	lastNDays(35) + logSetForHabit
Розподіл настрою	Горизонтальні смуги	Підрахунок за MOOD_META

Перший блок візуалізації представлений KPI-картками, які забезпечують швидкий доступ до найбільш важливої статистики. Серед показників відображаються кількість звичок, виконаних протягом поточного дня, найдовша безперервна серія виконання серед усіх звичок, загальна кількість відміток виконання та кількість записів, створених у щоденнику. Для кожної картки використовується окрема іконка та кольорове оформлення, що покращує сприйняття інформації та дозволяє швидко ідентифікувати потрібний показник. Такий підхід забезпечує отримання узагальненої інформації без необхідності аналізувати детальні звіти.

Наступним елементом інтерфейсу є стовпчиковий графік виконання звичок за тиждень, який будується на основі даних, сформованих функцією weeklyCompletionData. Графік відображає відсоток виконання звичок для кожного дня тижня та дозволяє оцінити рівень дисципліни користувача у короткостроковій перспективі. Для побудови діаграми використовується компонент BarChart, який автоматично масштабує значення та забезпечує інтерактивні підказки при наведенні курсора на окремі стовпчики. Завдяки цьому користувач може швидко визначити дні з найвищою та найнижчою активністю.

Для аналізу емоційного стану реалізовано лінійний графік тренду настрою за останні чотирнадцять днів. Дані для нього формуються функцією moodTrendData, яка перетворює записи щоденника на числові значення відповідно до обраного настрою. Отримані результати відображаються за допомогою компонента LineChart, що дозволяє простежити зміни емоційного стану користувача в часі. Особливістю реалізації є перевірка достатності даних для побудови графіка. Якщо система виявляє менше двох записів із зазначеним настроєм, замість порожньої діаграми відображається інформаційне повідомлення з рекомендацією додати більше записів до щоденника. Такий підхід підвищує якість користувацького інтерфейсу та запобігає появі некоректних або малоінформативних візуалізацій.

Важливим інструментом аналізу регулярності виконання звичок є теплова карта активності, яка охоплює останні тридцять п’ять днів. Для її побудови

використовується функція lastNDays(35), що формує набір послідовних дат, а також функція logSetForHabit, яка визначає, у які саме дні користувач виконував конкретну звичку. Для кожної звички створюється окремий рядок, який містить тридцять п’ять клітинок. Виконані дні зафарбовуються кольором, обраним для відповідної звички, тоді як невиконані відображаються нейтральним кольором інтерфейсу. Така форма подання інформації нагадує календар активності та дозволяє миттєво оцінити регулярність виконання завдань, виявити перерви або періоди підвищеної продуктивності.

Завершальним елементом модуля є блок розподілу настрою, який формує статистику використання різних емоційних станів у записах щоденника. Для кожної категорії настрою виконується підрахунок кількості відповідних записів, після чого результати відображаються у вигляді горизонтальних смуг прогресу. Довжина смуги пропорційна частці відповідного настрою від загальної кількості записів, а кольорове оформлення відповідає налаштуванням, визначеним у структурі MOOD_META. Такий підхід дозволяє швидко оцінити загальний емоційний фон користувача за весь період використання системи.

Таким чином, компонент ProgressView забезпечує комплексну систему візуального аналізу даних, поєднуючи числові показники, графіки, діаграми та елементи календарного представлення інформації. Використання бібліотеки Recharts у поєднанні з власними алгоритмами обробки статистичних даних дозволило створити зручний інструмент моніторингу особистої продуктивності, який допомагає користувачеві оцінювати результати роботи зі звичками та відстежувати зміни власного емоційного стану в динаміці.

3.4 Інтеграція з зовнішніми сервісами

Для підвищення функціональних можливостей вебдодатку «Особистий щоденник і трекер звичок» передбачено інтеграцію із зовнішніми сервісами, які забезпечують автоматизацію нагадувань та розширення можливостей взаємодії

користувача із системою. На відміну від основного функціоналу, що працює локально в браузері, інтеграційні механізми реалізуються на серверному рівні. Такий підхід дозволяє безпечно зберігати облікові дані доступу до сторонніх сервісів, не розкриваючи їх у клієнтській частині застосунку, а також запобігає блокуванню інтерфейсу під час виконання зовнішніх запитів.

Архітектура інтеграції побудована за принципом проміжного серверного шару, де вебдодаток взаємодіє лише з власним бекендом, а всі операції зі сторонніми платформами виконуються через спеціалізовані серверні модулі. Це забезпечує централізоване керування зовнішніми підключеннями, спрощує масштабування системи та підвищує її безпеку.

Основні інтеграції, передбачені в архітектурі системи, наведено у таблиці 3.2.

Таблиця 3.2 – Зовнішні сервіси, інтегровані із системою

Сервіс	**Призначення**	**Технологія / Протокол**
Google Calendar API	Створення подій-нагадувань про звички у календарі користувача	OAuth 2.0 + REST
Email-нотифікації	Щоденні та щотижневі звіти, нагадування про невиконані звички	SMTP / транзакційний email-провайдер

Однією з ключових інтеграцій є взаємодія із сервісом Google Calendar. Її основним призначенням є автоматичне створення подій-нагадувань для звичок, які користувач бажає виконувати регулярно. Для отримання доступу до календаря використовується протокол OAuth 2.0, який забезпечує безпечну авторизацію без передачі пароля сторонньому застосунку. Після успішного підключення користувач може активувати нагадування для окремої звички, а серверна частина системи автоматично створює відповідну подію у його календарі через REST API. У результаті користувач отримує стандартні повідомлення Google Calendar на комп’ютері або мобільному пристрої, що сприяє своєчасному виконанню запланованих дій.

Ще одним важливим напрямом інтеграції є система електронних повідомлень. Для цього використовується SMTP-протокол або спеціалізовані

сервіси транзакційних розсилок. Завдяки цьому користувач може отримувати автоматично сформовані листи з інформацією про поточний прогрес, статистику виконання звичок та нагадування про завдання, які залишилися невиконаними. Залежно від налаштувань такі повідомлення можуть надсилатися щодня або один раз на тиждень у вигляді зведеного звіту.

Логіка взаємодії між компонентами системи під час роботи механізму нагадувань наведена на діаграмі (рис. 3.2). Після того як користувач активує нагадування для певної звички, вебдодаток надсилає відповідний HTTP-запит до серверного API. Сервер обробляє отримані параметри та виконує створення події у Google Calendar із використанням авторизації OAuth 2.0. У відповідь сервіс повертає ідентифікатор створеної події, який може використовуватися для подальшого редагування або видалення нагадування.

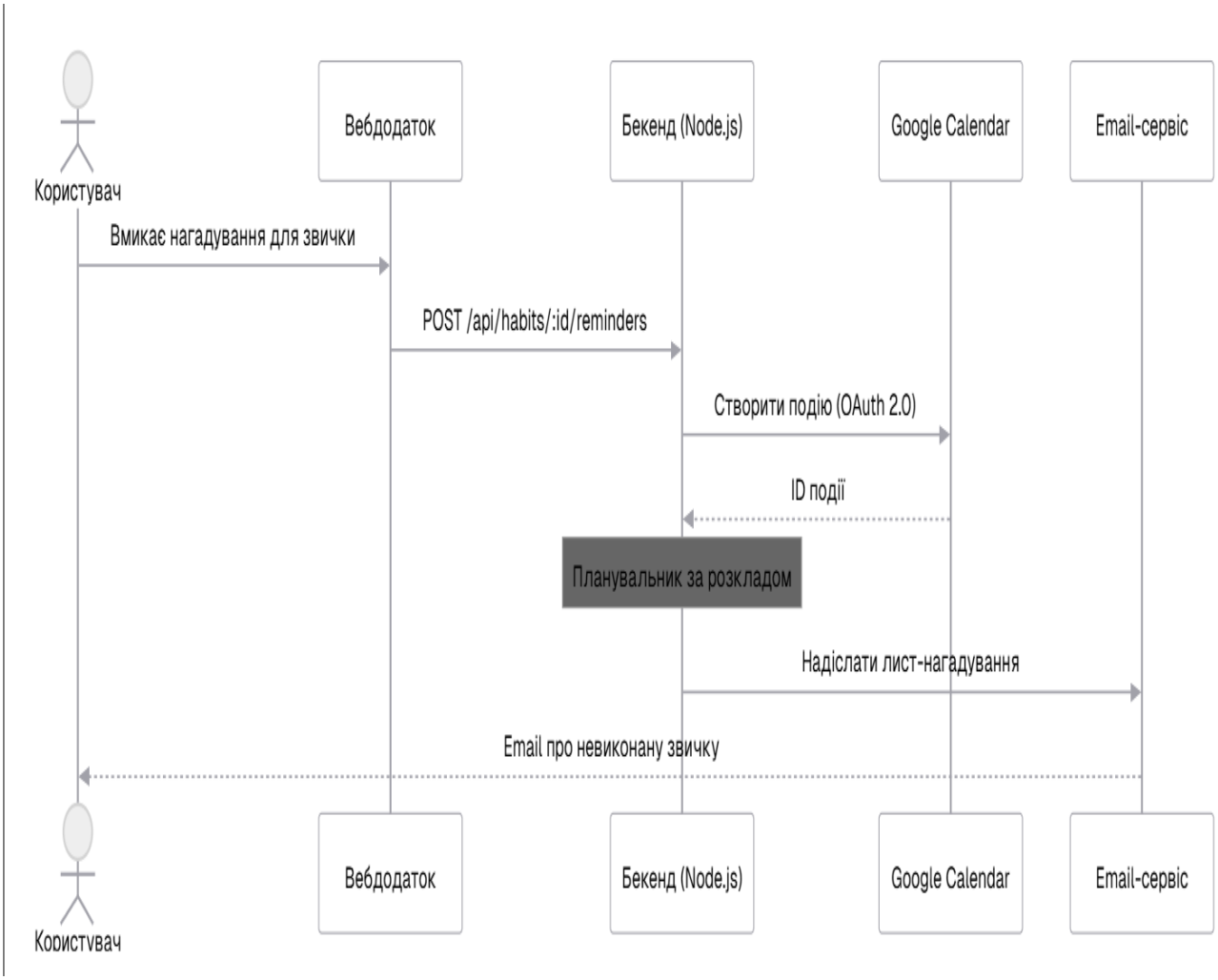


Рисунок 3.2 – Інтеграція з зовнішніми сервісами

Паралельно із цим працює механізм планувальника завдань, який у визначені проміжки часу аналізує стан виконання звичок користувачів. Якщо система виявляє невиконані завдання або необхідність формування звіту, вона автоматично звертається до сервісу електронної пошти та надсилає відповідне повідомлення. Таким чином користувач отримує нагадування через звичні для нього канали комунікації навіть тоді, коли вебдодаток не відкритий у браузері.

Представлена архітектура має низку переваг. По-перше, вона дозволяє розвантажити клієнтську частину застосунку та зберегти високу швидкодію інтерфейсу. По-друге, усі конфіденційні дані, включаючи токени доступу до зовнішніх сервісів, зберігаються виключно на сервері. По-третє, використання популярних платформ, таких як Google Calendar та електронна пошта, забезпечує інтеграцію з уже звичними для користувача інструментами організації часу.

Таким чином, інтеграція із зовнішніми сервісами розширює функціональність системи за межі локального вебдодатку та забезпечує автоматизовану підтримку користувача під час формування й підтримання корисних звичок.

3.5 Тестування функціоналу та юзабіліті

Після завершення реалізації основних функціональних модулів вебдодатку було проведено тестування програмного забезпечення з метою перевірки коректності роботи алгоритмів трекінгу звичок, розрахунку статистичних показників та забезпечення стабільності користувацького інтерфейсу. Тестування виконувалося із застосуванням фреймворку Jest, який дозволяє автоматизувати перевірку окремих функціональних компонентів системи та контролювати правильність їх роботи під час подальшого розвитку проєкту.

Основна увага була зосереджена на модулі обробки звичок (habit-utils), оскільки саме він забезпечує формування ключових показників прогресу

користувача, які використовуються в аналітичних розділах системи. Для перевірки функціональності було створено набір модульних тестів, що охоплюють операції роботи з датами, формування часових інтервалів, розрахунок поточних та максимальних серій виконання звичок, а також визначення відсотка виконання за певний період.

Файл тестування habit-utils.test.ts містить набір сценаріїв перевірки основних функцій системи (Додаток В). Для спрощення підготовки тестових даних реалізовано допоміжну функцію daysAgo(), яка автоматично генерує дату зі зміщенням відносно поточного дня та повертає її у форматі YYYY-MM-DD. Це дозволяє створювати незалежні від календарної дати тестові сценарії та забезпечує коректність результатів незалежно від моменту запуску тестів.

Одним із перших етапів стала перевірка функції toDateString(), яка використовується для стандартизованого представлення дат у всій системі. Тестування підтвердило правильність формування рядків у форматі «рік-місяць-день» із додаванням ведучих нулів для однозначних значень місяців і днів. Коректність роботи цієї функції є важливою передумовою для подальшого порівняння дат та формування статистики.

Наступним етапом було тестування функції lastNDays(), яка формує список останніх N днів для побудови теплових карт активності та графіків прогресу. У ході перевірки підтверджено, що функція повертає рівно задану кількість дат, причому останнім елементом завжди є поточний день, а першим – дата, зміщена на відповідну кількість днів назад.

Особливу увагу приділено перевірці алгоритму обчислення поточної серії виконання звички (calcStreak). Було протестовано декілька сценаріїв роботи функції. Перший сценарій перевіряв ситуацію, коли відсутні відмітки як за сьогоднішній, так і за вчорашній день. У такому випадку функція коректно повертає значення нуль. Другий сценарій моделював безперервне виконання звички протягом кількох днів поспіль, у результаті чого функція правильно обчислювала довжину серії. Окремо перевірялася ситуація, коли поточний день ще не завершений і відмітка за нього відсутня, проте звичка була виконана

напередодні. Результати тестування підтвердили правильність реалізованої логіки підтримання серії.

Для аналізу довгострокової активності користувача реалізовано функцію calcLongestStreak(), яка визначає найдовшу безперервну послідовність виконань. У процесі тестування було сформовано набір дат із декількома незалежними серіями виконання. Функція успішно знаходила найбільший безперервний інтервал та повертала його довжину. Додатково перевірявся випадок порожнього набору даних, для якого функція коректно повертала нульове значення.

Окремим напрямом тестування стала функція completionRate(), що використовується для визначення відсотка виконання звички за заданий період. Було перевірено сценарій повного виконання, коли всі дні в інтервалі мають відмітки, а також сценарій часткового виконання. Результати тестування підтвердили правильність математичних розрахунків та округлення отриманих значень до цілих відсотків.

Результати запуску тестового набору наведено на рисунку 3.3.

```
PASS __tests__/habit-utils.test.ts
  toDateString
    √ форматує дату як YYYY-MM-DD з ведучими нулями (55 ms)
  lastNDays
    √ повертає рівно N днів, останній — сьогодні (12 ms)
  calcStreak
    √ дорівнює 0, якщо немає відміток ні сьогодні, ні вчора (10 ms)
    √ рахує серію поспіль починаючи з сьогодні (7 ms)
    √ серія тримається, якщо сьогодні ще не відмічено, але вчора було (10 ms)
  calcLongestStreak
    √ знаходить найдовший безперервний відрізок (12 ms)
    √ повертає 0 для порожньої множини (11 ms)
  completionRate
    √ 100% коли всі дні відмічені (11 ms)
    √ округлює відсоток (1 з 3 днів ≈ 33%) (9 ms)

Test Suites: 1 passed, 1 total
Tests:       9 passed, 9 total
Snapshots:   0 total
Time:        12.206 s
Ran all test suites
```

Рисунок 3.3 – Результати автоматизованого тестування

Крім функціонального тестування, було проведено тестування юзабіліті застосунку. Перевірялася зручність навігації між вкладками «Прогрес», «Звички» та «Щоденник», швидкість створення нових записів і звичок, зрозумілість візуальних індикаторів та графіків, а також коректність роботи інтерфейсу в світлому і темному режимах. Результати перевірки показали, що всі основні сценарії взаємодії виконуються без додаткових інструкцій для користувача, а інформація про прогрес відображається зрозуміло та наочно.

Таким чином, проведене тестування підтвердило коректність роботи основних алгоритмів трекінгу звичок, правильність формування статистичних показників та стабільність інтерфейсу користувача. Отримані результати свідчать про готовність розробленого вебдодатку до подальшого використання та розширення функціональних можливостей.

3.6 Інструкція користувача

Вебдодаток для трекінгу звичок та ведення особистого щоденника розроблено як інтуїтивно зрозумілу систему, що не потребує спеціальної підготовки користувача. Усі основні функції згруповано у три логічні розділи: «Прогрес», «Звички» та «Щоденник», між якими здійснюється перемикання через верхню панель вкладок.

Після відкриття застосунку користувач потрапляє на головний екран, де автоматично завантажуються локальні дані з бази IndexedDB. У випадку першого запуску система може створити початкові тестові записи, що дозволяє одразу ознайомитися з функціоналом без необхідності ручного введення даних.

Розділ «Звички» призначений для створення, редагування та відстеження регулярних дій користувача. Для додавання нової звички необхідно натиснути кнопку «Додати», після чого відкривається модальне вікно з формою введення. У формі користувач зазначає назву звички, її опис, обирає іконку та колір для візуального представлення, а також визначає частоту виконання (щоденну або

щотижневу). Після збереження нова звичка автоматично з’являється у списку активних.

Для фіксації виконання звички достатньо натиснути кнопку «Відмітити сьогодні». Повторне натискання цієї ж кнопки скасовує відмітку, що дозволяє уникати помилкових записів. У картці кожної звички також відображається поточна серія виконання, відсоток виконання за останні 30 днів та мініатюрна теплова карта активності за тиждень.

Розділ «Щоденник» використовується для ведення особистих записів. Для створення нового запису необхідно натиснути кнопку «Новий запис», після чого відкривається форма, у якій користувач може ввести заголовок, основний текст, обрати настрій та додати теги. Запис можна редагувати або видаляти у будь-який момент. Усі записи автоматично сортуються за датою, що дозволяє переглядати хронологію подій.

Розділ «Прогрес» призначений для візуалізації досягнень користувача. Тут відображаються ключові показники активності, зокрема кількість виконаних звичок за день, найдовша серія виконання, загальна кількість відміток та кількість записів у щоденнику. Додатково доступні графіки виконання звичок, тренд настрою, теплова карта активності та розподіл емоційних станів. Усі графіки оновлюються автоматично відповідно до змін у даних.

Інтерфейс підтримує світлу та темну тему, перемикання між якими здійснюється за допомогою відповідної кнопки у верхній частині екрана. Обраний режим зберігається автоматично у локальному сховищі браузера та застосовується при наступному вході.

Усі дані користувача зберігаються локально у браузері за допомогою IndexedDB, що забезпечує автономну роботу системи без необхідності підключення до серверної частини. Це дозволяє використовувати застосунок як офлайн-інструмент для особистого трекінгу звичок і саморефлексії.

ВИСНОВКИ

У межах виконання бакалаврської роботи було розроблено вебзастосунок для ведення особистого щоденника та трекінгу звичок із візуалізацією прогресу. Основною метою проєкту було створення зручного цифрового інструменту для самоменеджменту, який поєднує функції фіксації щоденних записів, відстеження поведінкових звичок та аналізу особистої ефективності користувача.

У теоретичній частині роботи було досліджено поняття особистого щоденника та його роль у психології й управлінні поведінкою людини. Проаналізовано сучасні підходи до трекінгу звичок, а також розглянуто популярні існуючі рішення у цій сфері. На основі проведеного аналізу сформовано функціональні та нефункціональні вимоги до майбутньої системи, що дозволило визначити ключові напрямки її реалізації.

У другому розділі здійснено проєктування вебдодатку, зокрема визначено архітектуру системи, обґрунтовано вибір технологічного стеку та спроєктовано структуру даних. Було розроблено модель взаємодії основних компонентів системи, включаючи модуль щоденника, модуль звичок та механізми збереження й обробки даних. Це забезпечило логічну цілісність і масштабованість майбутнього застосунку.

У процесі реалізації програмного продукту було створено користувацький інтерфейс із поділом на функціональні розділи, реалізовано механізм додавання та редагування записів щоденника, а також систему керування звичками з можливістю щоденного та тижневого відстеження виконання. Окрему увагу приділено розробці алгоритмів підрахунку статистики, зокрема серій виконання та відсотка завершення звичок.

Важливою складовою проєкту стала візуалізація прогресу користувача, яка реалізована за допомогою графіків, діаграм і теплових карт. Це дозволило перетворити набір числових даних на наочну аналітичну інформацію, що сприяє кращому розумінню власної продуктивності та емоційного стану. Додатково

було реалізовано інтеграцію з зовнішніми сервісами, зокрема системами email-сповіщень та календарними сервісами, що розширює функціональні можливості застосунку.

Проведене тестування підтвердило коректність роботи основних функцій системи, стабільність алгоритмів обробки даних та зручність користувацького інтерфейсу. Юзабіліті-тестування показало, що застосунок є інтуїтивно зрозумілим і не потребує додаткового навчання для користувача.

Отже, розроблений вебзастосунок повністю відповідає поставленим вимогам і забезпечує ефективний інструмент для ведення особистого щоденника, трекінгу звичок та візуалізації прогресу. Отримані результати можуть бути використані як основа для подальшого розширення функціональності, зокрема впровадження мобільної версії, системи рекомендацій та елементів штучного інтелекту для персоналізації користувацького досвіду.

ПЕРЕЛІК ПОСИЛАНЬ

1. Особистий щоденник: що це таке і в чому його користь? URL : https://cau.in.ua/501-osobistiy-shchodennik.html (дата звернення : 23.03.2026).

2. Зуєва А. Мистецтво саморефлексії: як особистий щоденник покращує психічне здоров'я. Klepkainfo.com. 2024. URL : https://klepkainfo.com/psyhichne-zdorovya/mystecztvo-samorefleksiyi-yak-osobystyj-shhodennyk-pokrashhuye-psyhichne-zdorovya.html (дата звернення : 23.03.2026).

3. Mazzarisi A. The Power of Journaling: What Science Says About the Benefits for Mental Health and Well-Being. Child Mind Institute. 2025. URL : https://childmind.org/blog/the-power-of-journaling/ (дата звернення : 23.03.2026).

4. Majolo M., Gomes W. B., DeCastro T. G. Self-Consciousness and Self-Awareness: Associations between Stable and Transitory Levels of Evidence. Behavioral Sciences. 2023. Vol. 13, No. 2. P. 117. DOI : https://doi.org/10.3390/bs13020117.

5. Особистий щоденник: чому корисно вести щоденник під час тривоги та депресії. Сторінка психолога. URL : https://storinka-psihologa3.webnode.com.ua/l/osobistij-shchodennik-chomu-korisno-vesti-shchodennik-pid-chas-trivogi-ta-depresiji/ (дата звернення : 23.03.2026).

6. Усе про трекери звичок: навіщо потрібні, які бувають і як допомагають змінити життя. ENTERSTORE. 2025. URL : https://shop.enterprint.com.ua/use-pro-trekery-zvychok-navishcho-potribni-yaki-buvaiut-i-yak-dopomahaiut-zminyty-zhyttia/ (дата звернення : 23.03.2026).

7. Gardner B. Доведення до звичного: використання наукового підходу до формування звички у реальному житті. Practical Health Psychology. URL : https://practicalhealthpsychology.com/uk/2016/01/getting-into-the-habit-applying-the-science-of-habit-formation-to-the-real-world/ (дата звернення : 23.03.2026).

8. Habit Tracking: The 16-Year Reversal. The National Law Review. URL :

https://natlawreview.com/press-releases/habit-tracking-16-year-reversal (дата звернення : 23.03.2026).

9. Subramaniam A. How Feedback, Identity, and Reward Loops Turn Everyday Choices into Change. Psychology Today. URL : https://www.psychologytoday.com/us/blog/parenting-from-a-neuroscience-perspective/202512/the-science-behind-habit-tracking (дата звернення : 23.03.2026).

10. The Art of Habit Tracking Through Journaling: Build Better Habits with Journal Buddy. Journal Buddy. URL : https://www.journalbuddy.ai/blog/habit-tracking-with-journaling (дата звернення : 23.03.2026).

11. Postma H. Using Habit Trackers in Bullet Journaling to Build Lasting Routines. Scrapbook & Journal. 2025. URL : https://scrapbookandjournal.com/blogs/blog/using-habit-trackers-in-bullet-journaling-to-build-lasting-routines (дата звернення : 23.03.2026).

12. Habit Tracking. Guy Pursey. URL : https://guypursey.com/blog/202101241950-habit-tracking (дата звернення : 23.03.2026).

13. Zhu Y., Long Y., Wang H., Lee K. P., Zhang L., Wang S. J. Digital Behavior Change Intervention Designs for Habit Formation: Systematic Review. Journal of Medical Internet Research. 2024. Vol. 26. DOI : https://doi.org/10.2196/54375.

14. Penzu — The Most Secure Online Diary and Personal Journal. URL : https://penzu.com (дата звернення : 23.03.2026).

15. Top 10 Personal Journal / Diary Apps: Features, Pros, Cons & Comparison. scmGalaxy. 2026. URL : https://www.scmgalaxy.com/tutorials/top-10-personal-journal-diary-apps-features-pros-cons-comparison/ (дата звернення : 23.03.2026).

16. Habitica App Review 2024: Pros & Cons, Cost, & Who It's Right For. Choosing Therapy. 2024. URL : https://www.choosingtherapy.com/habitica-app-review/ (дата звернення : 23.03.2026).

17. Habitica: Gamify Your Tasks — Review & Alternatives. ProdApps. 2026.

URL : https://productivity-apps.com/apps/habitica (дата звернення : 23.03.2026).

18. Notion (productivity software). Wikipedia. 2026. URL : https://en.wikipedia.org/wiki/Notion_(productivity_software) (дата звернення : 23.03.2026).

19. Google Keep 2026: Benefits, Features & Pricing. Software Advice. 2026. URL : https://www.softwareadvice.com/note-taking/google-keep-profile/ (дата звернення : 23.03.2026).

20. Google Keep is Missing 6 Basic Features and It Really Annoys Me. Android Police. 2025. URL : https://www.androidpolice.com/google-keep-missing-features-annoy-me/ (дата звернення : 23.03.2026).

21. React v19. React. 2024. URL : https://react.dev/blog/2024/12/05/react-19 (дата звернення : 23.03.2026).

22. Lai J., Story J., Markbåge S., Neutkens T. Next.js 16. Next.js. 2025. URL : https://nextjs.org/blog/next-16 (дата звернення : 23.03.2026).

23. Angular. URL : https://angular.dev/ (дата звернення : 23.03.2026).

24. Vue.js. The Progressive JavaScript Framework. URL : https://vuejs.org/ (дата звернення : 23.03.2026).

25. TypeScript: JavaScript with Syntax for Types. URL : https://www.typescriptlang.org/ (дата звернення : 23.03.2026).

26. Tailwind CSS. URL : https://tailwindcss.com/ (дата звернення : 23.03.2026).

27. Shadcn/UI. The Foundation for Your Design System. URL : https://ui.shadcn.com/ (дата звернення : 23.03.2026).

28. Recharts. URL : https://recharts.github.io/ (дата звернення : 23.03.2026).

29. IndexedDB API. MDN Web Docs. 2025. URL : https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API (дата звернення : 23.03.2026).

30. Node.js — Run JavaScript Everywhere. URL : https://nodejs.org/en (дата звернення : 23.03.2026).

31. PostgreSQL. URL : https://www.postgresql.org/ (дата звернення :

23.03.2026).

32. Jest. URL : https://jestjs.io/ (дата звернення : 23.03.2026).

ДОДАТОК А

ЛІСТИНГ РЕАЛІЗАЦІЇ ІНТЕРФЕЙСУ КОРИСТУВАЧА

Лістинг А.1 – components/diary-app.tsx

```
"use client"

import { useEffect, useState } from "react"
import { seedIfEmpty } from "@/lib/db"
import { HabitsView } from "@/components/habits-view"
import { JournalView } from "@/components/journal-view"
import { ProgressView } from "@/components/progress-view"
import { Tabs, TabsList, TabsTrigger, TabsContent } from "@/components/ui/tabs"
import { Button } from "@/components/ui/button"
import { BarChart3, BookOpen, Moon, Sun, Target } from "lucide-react"

export function DiaryApp() {
  const [ready, setReady] = useState(false)
  const [dark, setDark] = useState(false)

  useEffect(() => {
    seedIfEmpty().finally(() => setReady(true))
  }, [])

  useEffect(() => {
    const stored = localStorage.getItem("diary-theme")
    const prefersDark = window.matchMedia("(prefers-color-scheme: dark)").matches
    const isDark = stored ? stored === "dark" : prefersDark
    setDark(isDark)
    document.documentElement.classList.toggle("dark", isDark)
  }, [])

  function toggleTheme() {
    const next = !dark
    setDark(next)
    document.documentElement.classList.toggle("dark", next)
    localStorage.setItem("diary-theme", next ? "dark" : "light")
  }

  return (
    <div className="mx-auto min-h-screen w-full max-w-5xl px-4 py-8 sm:px-6">
      <header className="mb-8 flex items-center justify-between gap-4">
```

```
        <div className="flex items-center gap-3">
          <span className="flex size-10 items-center justify-center rounded-xl bg-
primary text-primary-foreground">
            <Target className="size-5" />
          </span>
          <div>
            <h1 className="text-lg font-semibold leading-tight text-foreground">Мій
Щоденник</h1>
            <p className="text-xs text-muted-foreground">Звички · Записи ·
Прогрес</p>
          </div>
        </div>
        <Button variant="outline" size="icon" onClick={toggleTheme} aria-
label="Перемкнути тему">
          {dark ? <Sun className="size-4" /> : <Moon className="size-4" />}
        </Button>
      </header>

      {!ready ? (
        <div className="py-20 text-center text-sm text-muted-
foreground">Завантаження бази даних…</div>
      ) : (
        <Tabs defaultValue="progress">
          <TabsList className="mb-6 grid w-full grid-cols-3 sm:w-auto">
            <TabsTrigger value="progress">
              <BarChart3 className="size-4" />
              <span className="hidden sm:inline">Прогрес</span>
            </TabsTrigger>
            <TabsTrigger value="habits">
              <Target className="size-4" />
              <span className="hidden sm:inline">Звички</span>
            </TabsTrigger>
            <TabsTrigger value="journal">
              <BookOpen className="size-4" />
              <span className="hidden sm:inline">Щоденник</span>
            </TabsTrigger>
          </TabsList>

          <TabsContent value="progress">
            <ProgressView />
          </TabsContent>
          <TabsContent value="habits">
            <HabitsView />
          </TabsContent>
          <TabsContent value="journal">
            <JournalView />
          </TabsContent>
        </Tabs>
      )}
```

```
      <footer className="mt-12 border-t border-border pt-6 text-center text-xs
text-muted-foreground">
        Усі дані зберігаються локально у вашому браузері (IndexedDB)
      </footer>
    </div>
  )
}
```

Лістинг А.2 – components/journal-view.tsx

```
"use client"

import { useState } from "react"
import { useLiveQuery } from "dexie-react-hooks"
import { db, type JournalEntry, type Mood, MOOD_META, toDateString } from
"@/lib/db"
import { Card } from "@/components/ui/card"
import { Button } from "@/components/ui/button"
import { Input } from "@/components/ui/input"
import { Textarea } from "@/components/ui/textarea"
import { Label } from "@/components/ui/label"
import { Badge } from "@/components/ui/badge"
import {
  Dialog,
  DialogContent,
  DialogHeader,
  DialogTitle,
  DialogTrigger,
  DialogFooter,
} from "@/components/ui/dialog"
import { Plus, Trash2, Pencil } from "lucide-react"
import { cn } from "@/lib/utils"

const MOODS = Object.keys(MOOD_META) as Mood[]

export function JournalView() {
  const entries = useLiveQuery(() =>
db.journal.orderBy("date").reverse().toArray(), [])
  const [open, setOpen] = useState(false)
  const [editing, setEditing] = useState<JournalEntry | null>(null)

  function openNew() {
    setEditing(null)
    setOpen(true)
  }
  function openEdit(entry: JournalEntry) {
    setEditing(entry)
    setOpen(true)
```

```
  }

  return (
    <div className="space-y-6">
      <div className="flex items-center justify-between gap-4">
        <div>
          <h2 className="text-xl font-semibold text-foreground">Щоденник</h2>
          <p className="text-sm text-muted-foreground">Записуйте думки та
відстежуйте настрій</p>
        </div>
        <Dialog open={open} onOpenChange={setOpen}>
          <DialogTrigger render={<Button />} onClick={openNew}>
            <Plus className="size-4" />
            Новий запис
          </DialogTrigger>
          <EntryForm entry={editing} onDone={() => setOpen(false)} />
        </Dialog>
      </div>

      {entries && entries.length === 0 && (
        <Card className="p-10 text-center text-sm text-muted-foreground">
          Ще немає записів. Створіть перший!
        </Card>
      )}

      <div className="space-y-4">
        {entries?.map((entry) => {
          const mood = MOOD_META[entry.mood]
          const d = new Date(entry.date + "T00:00:00")
          return (
            <Card key={entry.id} className="p-5">
              <div className="flex items-start gap-4">
                <div
                  className="flex size-12 shrink-0 items-center justify-center
rounded-xl text-2xl"
                  style={{ backgroundColor: `color-mix(in oklch, ${mood.color} 16%,
transparent)` }}
                  title={mood.label}
                >
                  {mood.emoji}
                </div>
                <div className="min-w-0 flex-1">
                  <div className="flex items-start justify-between gap-2">
                    <div>
                      <h3 className="font-medium text-foreground">{entry.title ||
"Без назви"}</h3>
                      <p className="text-xs text-muted-foreground">
                        {d.toLocaleDateString("uk-UA", { day: "numeric", month:
"long", year: "numeric" })} · {mood.label}
                      </p>
                    </div>
```

```
                    <div className="flex gap-1">
                      <Button variant="ghost" size="icon" className="size-8"
onClick={() => openEdit(entry)} aria-label="Редагувати">
                        <Pencil className="size-4" />
                      </Button>
                      <Button
                        variant="ghost"
                        size="icon"
                        className="size-8 text-muted-foreground hover:text-
destructive"
                        onClick={() => entry.id && db.journal.delete(entry.id)}
                        aria-label="Видалити"
                      >
                        <Trash2 className="size-4" />
                      </Button>
                    </div>
                  </div>
                  <p className="mt-2 whitespace-pre-wrap text-sm leading-relaxed
text-foreground/90">{entry.content}</p>
                  {entry.tags.length > 0 && (
                    <div className="mt-3 flex flex-wrap gap-1.5">
                      {entry.tags.map((t) => (
                        <Badge key={t} variant="secondary" className="font-normal">
                          #{t}
                        </Badge>
                      ))}
                    </div>
                  )}
                </div>
              </div>
            </Card>
          )
        })}
      </div>
    </div>
  )
}

function EntryForm({ entry, onDone }: { entry: JournalEntry | null; onDone: () =>
void }) {
  const [title, setTitle] = useState(entry?.title ?? "")
  const [content, setContent] = useState(entry?.content ?? "")
  const [mood, setMood] = useState<Mood>(entry?.mood ?? "good")
  const [tagsInput, setTagsInput] = useState(entry?.tags.join(", ") ?? "")
  const [date, setDate] = useState(entry?.date ?? toDateString(new Date()))

  async function save() {
    if (!content.trim()) return
    const tags = tagsInput
      .split(",")
      .map((t) => t.trim().replace(/^#/, ""))
```

```
      .filter(Boolean)
    const now = Date.now()
    if (entry?.id) {
      await db.journal.update(entry.id, { title, content, mood, tags, date,
updatedAt: now })
    } else {
      await db.journal.add({ title, content, mood, tags, date, createdAt: now,
updatedAt: now })
    }
    onDone()
  }

  return (
    <DialogContent className="max-h-[90vh] overflow-y-auto">
      <DialogHeader>
        <DialogTitle>{entry ? "Редагувати запис" : "Новий запис"}</DialogTitle>
      </DialogHeader>
      <div className="space-y-4">
        <div className="grid grid-cols-2 gap-3">
          <div className="space-y-2">
            <Label htmlFor="j-title">Заголовок</Label>
            <Input id="j-title" value={title} onChange={(e) =>
setTitle(e.target.value)} placeholder="Про що цей день?" />
          </div>
          <div className="space-y-2">
            <Label htmlFor="j-date">Дата</Label>
            <Input id="j-date" type="date" value={date} onChange={(e) =>
setDate(e.target.value)} />
          </div>
        </div>

        <div className="space-y-2">
          <Label>Настрій</Label>
          <div className="flex flex-wrap gap-2">
            {MOODS.map((m) => {
              const meta = MOOD_META[m]
              return (
                <button
                  key={m}
                  type="button"
                  onClick={() => setMood(m)}
                  className={cn(
                    "flex flex-col items-center gap-1 rounded-lg border px-3 py-2
text-xs transition-colors",
                    mood === m ? "border-primary bg-primary/10" : "border-border
hover:bg-muted",
                  )}
                >
                  <span className="text-xl">{meta.emoji}</span>
                  <span className="text-muted-foreground">{meta.label}</span>
                </button>
```

```
              )
            })}
          </div>
        </div>

        <div className="space-y-2">
          <Label htmlFor="j-content">Запис</Label>
          <Textarea id="j-content" value={content} onChange={(e) =>
setContent(e.target.value)} rows={6} placeholder="Що сталося сьогодні? Як ви
почуваєтесь?" />
        </div>

        <div className="space-y-2">
          <Label htmlFor="j-tags">Теги (через кому)</Label>
          <Input id="j-tags" value={tagsInput} onChange={(e) =>
setTagsInput(e.target.value)} placeholder="робота, спорт, сім'я" />
        </div>
      </div>
      <DialogFooter>
        <Button onClick={save} disabled={!content.trim()}>
          Зберегти
        </Button>
      </DialogFooter>
    </DialogContent>
  )
}
```

Лістинг А.3 – components/habits-view.tsx

```
"use client"

import { useState } from "react"
import { useLiveQuery } from "dexie-react-hooks"
import { db, type Habit, type HabitFrequency, HABIT_COLORS, toDateString } from
"@/lib/db"
import { calcStreak, completionRate, logSetForHabit, lastNDays } from "@/lib/stats"
import { HabitIcon, HABIT_ICONS } from "@/components/habit-icon"
import { Card } from "@/components/ui/card"
import { Button } from "@/components/ui/button"
import { Input } from "@/components/ui/input"
import { Textarea } from "@/components/ui/textarea"
import { Label } from "@/components/ui/label"
import { Badge } from "@/components/ui/badge"
import {
  Dialog,
  DialogContent,
  DialogHeader,
  DialogTitle,
```

```
  DialogTrigger,
  DialogFooter,
} from "@/components/ui/dialog"
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from
"@/components/ui/select"
import { Check, Flame, Plus, Trash2 } from "lucide-react"
import { cn } from "@/lib/utils"

const ICON_KEYS = Object.keys(HABIT_ICONS)
const COLOR_KEYS = Object.keys(HABIT_COLORS)

export function HabitsView() {
  const allHabits = useLiveQuery(() => db.habits.toArray(), [])
  const logs = useLiveQuery(() => db.habitLogs.toArray(), [])
  const [open, setOpen] = useState(false)

  const active = (allHabits ?? []).filter((h) => !h.archived)
  const today = toDateString(new Date())

  async function toggleToday(habit: Habit) {
    if (!habit.id) return
    const existing = await db.habitLogs.where({ habitId: habit.id, date: today
}).first()
    if (existing?.id) {
      await db.habitLogs.delete(existing.id)
    } else {
      await db.habitLogs.add({ habitId: habit.id, date: today, createdAt:
Date.now() })
    }
  }

  return (
    <div className="space-y-6">
      <div className="flex items-center justify-between gap-4">
        <div>
          <h2 className="text-xl font-semibold text-foreground">Мої звички</h2>
          <p className="text-sm text-muted-foreground">Відмічайте виконання та
слідкуйте за серіями</p>
        </div>
        <Dialog open={open} onOpenChange={setOpen}>
          <DialogTrigger render={<Button />}>
            <Plus className="size-4" />
            Додати
          </DialogTrigger>
          <HabitForm onDone={() => setOpen(false)} />
        </Dialog>
      </div>

      {active.length === 0 && (
        <Card className="flex flex-col items-center justify-center gap-2 p-10 text-
center">
```

```
          <p className="text-sm text-muted-foreground">Поки що немає звичок.
Додайте першу!</p>
        </Card>
      )}

      <div className="grid gap-4 sm:grid-cols-2">
        {active.map((habit) => {
          const set = logSetForHabit(logs ?? [], habit.id!)
          const streak = calcStreak(set)
          const rate = completionRate(set, 30)
          const doneToday = set.has(today)
          const color = HABIT_COLORS[habit.color] ?? HABIT_COLORS.emerald
          const week = lastNDays(7)
          return (
            <Card key={habit.id} className="flex flex-col gap-4 p-5">
              <div className="flex items-start justify-between gap-3">
                <div className="flex items-center gap-3">
                  <span
                    className="flex size-11 items-center justify-center rounded-xl"
                    style={{ backgroundColor: `color-mix(in oklch, ${color} 18%,
transparent)`, color }}
                  >
                    <HabitIcon name={habit.icon} className="size-5" />
                  </span>
                  <div>
                    <h3 className="font-medium leading-tight text-
foreground">{habit.name}</h3>
                    <p className="text-xs text-muted-
foreground">{habit.description}</p>
                  </div>
                </div>
                <DeleteHabitButton habit={habit} />
              </div>

              <div className="flex items-center gap-4 text-sm">
                <span className="flex items-center gap-1 font-medium" style={{
color }}>
                  <Flame className="size-4" />
                  {streak} дн.
                </span>
                <span className="text-muted-foreground">{rate}% за 30 дн.</span>
                <Badge variant="secondary" className="ml-auto">
                  {habit.frequency === "daily" ? "Щодня" :
`${habit.targetPerWeek}×/тиждень`}
                </Badge>
              </div>

              {/* Міні теплова карта тижня */}
              <div className="flex items-center gap-1.5">
                {week.map((ds) => {
                  const done = set.has(ds)
```

```
                  const d = new Date(ds + "T00:00:00")
                  return (
                    <div
                      key={ds}
                      title={ds}
                      className="flex h-7 flex-1 items-center justify-center
rounded-md text-[10px] font-medium"
                      style={{
                        backgroundColor: done ? color : "var(--muted)",
                        color: done ? "white" : "var(--muted-foreground)",
                      }}
                    >
                      {["Н", "П", "В", "С", "Ч", "П", "С"][d.getDay()]}
                    </div>
                  )
                })}
              </div>

              <Button
                variant={doneToday ? "default" : "outline"}
                onClick={() => toggleToday(habit)}
                className={cn("w-full", doneToday && "border-transparent")}
                style={doneToday ? { backgroundColor: color, color: "white" } :
undefined}
              >
                <Check className="size-4" />
                {doneToday ? "Виконано сьогодні" : "Відмітити сьогодні"}
              </Button>
            </Card>
          )
        })}
      </div>
    </div>
  )
}

function DeleteHabitButton({ habit }: { habit: Habit }) {
  async function remove() {
    if (!habit.id) return
    await db.habitLogs.where("habitId").equals(habit.id).delete()
    await db.habits.delete(habit.id)
  }
  return (
    <Button
      variant="ghost"
      size="icon"
      className="size-8 text-muted-foreground hover:text-destructive"
      onClick={remove}
      aria-label="Видалити звичку"
    >
      <Trash2 className="size-4" />
```

```
    </Button>
  )
}

function HabitForm({ onDone }: { onDone: () => void }) {
  const [name, setName] = useState("")
  const [description, setDescription] = useState("")
  const [icon, setIcon] = useState("target")
  const [color, setColor] = useState("emerald")
  const [frequency, setFrequency] = useState<HabitFrequency>("daily")
  const [target, setTarget] = useState(3)

  async function save() {
    if (!name.trim()) return
    await db.habits.add({
      name: name.trim(),
      description: description.trim(),
      icon,
      color,
      frequency,
      targetPerWeek: frequency === "daily" ? 7 : target,
      createdAt: Date.now(),
      archived: false,
    })
    onDone()
  }

  return (
    <DialogContent>
      <DialogHeader>
        <DialogTitle>Нова звичка</DialogTitle>
      </DialogHeader>
      <div className="space-y-4">
        <div className="space-y-2">
          <Label htmlFor="h-name">Назва</Label>
          <Input id="h-name" value={name} onChange={(e) => setName(e.target.value)}
placeholder="Наприклад: Ранкова пробіжка" />
        </div>
        <div className="space-y-2">
          <Label htmlFor="h-desc">Опис</Label>
          <Textarea id="h-desc" value={description} onChange={(e) =>
setDescription(e.target.value)} placeholder="Деталі звички" rows={2} />
        </div>

        <div className="space-y-2">
          <Label>Іконка</Label>
          <div className="flex flex-wrap gap-2">
            {ICON_KEYS.map((key) => (
              <button
                key={key}
                type="button"
```

```
                onClick={() => setIcon(key)}
                className={cn(
                  "flex size-10 items-center justify-center rounded-lg border
transition-colors",
                  icon === key ? "border-primary bg-primary/10 text-primary" :
"border-border text-muted-foreground hover:bg-muted",
                )}
                aria-label={key}
              >
                <HabitIcon name={key} className="size-5" />
              </button>
            ))}
          </div>
        </div>

        <div className="space-y-2">
          <Label>Колір</Label>
          <div className="flex flex-wrap gap-2">
            {COLOR_KEYS.map((key) => (
              <button
                key={key}
                type="button"
                onClick={() => setColor(key)}
                className={cn("size-8 rounded-full border-2 transition-transform",
color === key ? "scale-110 border-foreground" : "border-transparent")}
                style={{ backgroundColor: HABIT_COLORS[key] }}
                aria-label={key}
              />
            ))}
          </div>
        </div>

        <div className="grid grid-cols-2 gap-3">
          <div className="space-y-2">
            <Label>Частота</Label>
            <Select value={frequency} onValueChange={(v) => setFrequency(v as
HabitFrequency)}>
              <SelectTrigger>
                <SelectValue />
              </SelectTrigger>
              <SelectContent>
                <SelectItem value="daily">Щодня</SelectItem>
                <SelectItem value="weekly">Щотижня</SelectItem>
              </SelectContent>
            </Select>
          </div>
          {frequency === "weekly" && (
            <div className="space-y-2">
              <Label htmlFor="h-target">Разів на тиждень</Label>
              <Input id="h-target" type="number" min={1} max={7} value={target}
onChange={(e) => setTarget(Number(e.target.value))} />
```

```
            </div>
          )}
        </div>
      </div>
      <DialogFooter>
        <Button onClick={save} disabled={!name.trim()}>
          Зберегти
        </Button>
      </DialogFooter>
    </DialogContent>
  )
}
```

Лістинг А.4 – components/progress-view.tsx

```
"use client"

import { useLiveQuery } from "dexie-react-hooks"
import { db, HABIT_COLORS, MOOD_META, toDateString } from "@/lib/db"
import {
  calcStreak,
  calcLongestStreak,
  completionRate,
  logSetForHabit,
  weeklyCompletionData,
  moodTrendData,
  lastNDays,
} from "@/lib/stats"
import { Card } from "@/components/ui/card"
import { HabitIcon } from "@/components/habit-icon"
import { Flame, Trophy, CalendarCheck, NotebookPen } from "lucide-react"
import {
  ResponsiveContainer,
  BarChart,
  Bar,
  XAxis,
  YAxis,
  Tooltip,
  LineChart,
  Line,
  CartesianGrid,
} from "recharts"

export function ProgressView() {
  const habits = useLiveQuery(() => db.habits.toArray(), [])
  const logs = useLiveQuery(() => db.habitLogs.toArray(), [])
  const entries = useLiveQuery(() => db.journal.toArray(), [])
```

```
  if (!habits || !logs || !entries) {
    return <div className="py-20 text-center text-sm text-muted-
foreground">Завантаження…</div>
  }

  const active = habits.filter((h) => !h.archived)
  const today = toDateString(new Date())

  // Загальна статистика
  const doneToday = logs.filter((l) => l.date === today).length
  const bestStreak = active.reduce((max, h) => Math.max(max,
calcLongestStreak(logSetForHabit(logs, h.id!))), 0)
  const totalCompletions = logs.length
  const journalCount = entries.length

  const weekData = weeklyCompletionData(active, logs)
  const moodData = moodTrendData(entries, 14).filter((d) => d.mood !== null)
  const heatDays = lastNDays(35)

  const stats = [
    { label: "Виконано сьогодні", value: `${doneToday}/${active.length}`, icon:
CalendarCheck, color: HABIT_COLORS.emerald },
    { label: "Найдовша серія", value: `${bestStreak} дн.`, icon: Trophy, color:
HABIT_COLORS.amber },
    { label: "Всього відміток", value: totalCompletions, icon: Flame, color:
HABIT_COLORS.rose },
    { label: "Записів у щоденнику", value: journalCount, icon: NotebookPen, color:
HABIT_COLORS.violet },
  ]
  return (
    <div className="space-y-6">
      <div>
        <h2 className="text-xl font-semibold text-foreground">Прогрес</h2>
        <p className="text-sm text-muted-foreground">Візуалізація ваших звичок та
настрою</p>
      </div>

      {/* KPI картки */}
      <div className="grid grid-cols-2 gap-4 lg:grid-cols-4">
        {stats.map((s) => (
          <Card key={s.label} className="flex items-center gap-3 p-4">
            <span
              className="flex size-10 shrink-0 items-center justify-center rounded-
lg"
              style={{ backgroundColor: `color-mix(in oklch, ${s.color} 16%,
transparent)`, color: s.color }}
            >
              <s.icon className="size-5" />
            </span>
            <div className="min-w-0">
              <p className="truncate text-xs text-muted-foreground">{s.label}</p>
```

```
              <p className="text-lg font-semibold text-foreground">{s.value}</p>
            </div>
          </Card>
        ))}
      </div>
      <div className="grid gap-4 lg:grid-cols-2">
        {/* Виконання за тиждень */}
        <Card className="p-5">
          <h3 className="mb-4 font-medium text-foreground">Виконання звичок за
тиждень</h3>
          <ResponsiveContainer width="100%" height={220}>
            <BarChart data={weekData}>
              <CartesianGrid strokeDasharray="3 3" stroke="var(--border)"
vertical={false} />
              <XAxis dataKey="day" stroke="var(--muted-foreground)" fontSize={12}
tickLine={false} axisLine={false} />
              <YAxis stroke="var(--muted-foreground)" fontSize={12}
tickLine={false} axisLine={false} unit="%" domain={[0, 100]} />
              <Tooltip
                cursor={{ fill: "var(--muted)" }}
                contentStyle={{
                  background: "var(--popover)",
                  border: "1px solid var(--border)",
                  borderRadius: "0.5rem",
                  color: "var(--popover-foreground)",
                  fontSize: 12,
                }}
                formatter={(v) => [`${v}%`, "Виконано"]}
              />
              <Bar dataKey="percent" fill="var(--chart-1)" radius={[6, 6, 0, 0]} />
            </BarChart>
          </ResponsiveContainer>
        </Card>

        {/* Тренд настрою */}
        <Card className="p-5">
          <h3 className="mb-4 font-medium text-foreground">Тренд настрою (14
днів)</h3>
          {moodData.length > 1 ? (
            <ResponsiveContainer width="100%" height={220}>
              <LineChart data={moodData}>
                <CartesianGrid strokeDasharray="3 3" stroke="var(--border)"
vertical={false} />
                <XAxis dataKey="label" stroke="var(--muted-foreground)"
fontSize={12} tickLine={false} axisLine={false} />
                <YAxis domain={[1, 5]} ticks={[1, 2, 3, 4, 5]} stroke="var(--muted-
foreground)" fontSize={12} tickLine={false} axisLine={false} />
                <Tooltip
                  contentStyle={{
                    background: "var(--popover)",
                    border: "1px solid var(--border)",
```

```
                    borderRadius: "0.5rem",
                    color: "var(--popover-foreground)",
                    fontSize: 12,
                  }}
                  formatter={(v) => [v, "Настрій"]}
                />
                <Line type="monotone" dataKey="mood" stroke="var(--chart-3)"
strokeWidth={2.5} dot={{ r: 3, fill: "var(--chart-3)" }} />
              </LineChart>
            </ResponsiveContainer>
          ) : (
            <div className="flex h-[220px] items-center justify-center text-center
text-sm text-muted-foreground">
              Додайте більше записів у щоденник, щоб побачити тренд настрою
            </div>
          )}
        </Card>
      </div>
      {/* Теплова карта по звичках */}
      <Card className="p-5">
        <h3 className="mb-4 font-medium text-foreground">Карта активності (35
днів)</h3>
        {active.length === 0 ? (
          <p className="text-sm text-muted-foreground">Немає активних звичок.</p>
        ) : (
          <div className="space-y-3 overflow-x-auto">
            {active.map((habit) => {
              const set = logSetForHabit(logs, habit.id!)
              const color = HABIT_COLORS[habit.color] ?? HABIT_COLORS.emerald
              return (
                <div key={habit.id} className="flex items-center gap-3">
                  <div className="flex w-32 shrink-0 items-center gap-2">
                    <HabitIcon name={habit.icon} className="size-4 shrink-0"
style={{ color }} />
                    <span className="truncate text-xs text-
foreground">{habit.name}</span>
                  </div>
                  <div className="flex gap-1">
                    {heatDays.map((ds) => {
                      const done = set.has(ds)
                      return (
                        <div
                          key={ds}
                          title={ds}
                          className="size-4 rounded-sm"
                          style={{ backgroundColor: done ? color : "var(--muted)"
}}
                        />
                      )
                    })}
                  </div>
```

```
                  <span className="ml-auto shrink-0 text-xs font-medium" style={{
color }}>
                    {completionRate(set, 35)}%
                  </span>
                </div>
              )
            })}
          </div>
        )}
      </Card>

      {/* Розподіл настрою */}
      <MoodDistribution entries={entries} />
    </div>
  )
}
function MoodDistribution({ entries }: { entries: { mood: keyof typeof MOOD_META
}[] }) {
  const counts = (Object.keys(MOOD_META) as (keyof typeof MOOD_META)[]).map((m) =>
({
    mood: m,
    label: MOOD_META[m].label,
    emoji: MOOD_META[m].emoji,
    color: MOOD_META[m].color,
    count: entries.filter((e) => e.mood === m).length,
  }))
  const total = entries.length || 1
  return (
    <Card className="p-5">
      <h3 className="mb-4 font-medium text-foreground">Розподіл настрою</h3>
      <div className="space-y-3">
        {counts.map((c) => (
          <div key={c.mood} className="flex items-center gap-3">
            <span className="w-28 shrink-0 text-sm text-foreground">
              {c.emoji} {c.label}
            </span>
            <div className="h-3 flex-1 overflow-hidden rounded-full bg-muted">
              <div
                className="h-full rounded-full transition-all"
                style={{ width: `${(c.count / total) * 100}%`, backgroundColor:
c.color }}
              />
            </div>
            <span className="w-8 shrink-0 text-right text-xs text-muted-
foreground">{c.count}</span>
          </div>
        ))}
      </div>
    </Card>
  )
}
```

ДОДАТОК Б

ІНТЕРФЕЙС ВЕБДОДАТКУ

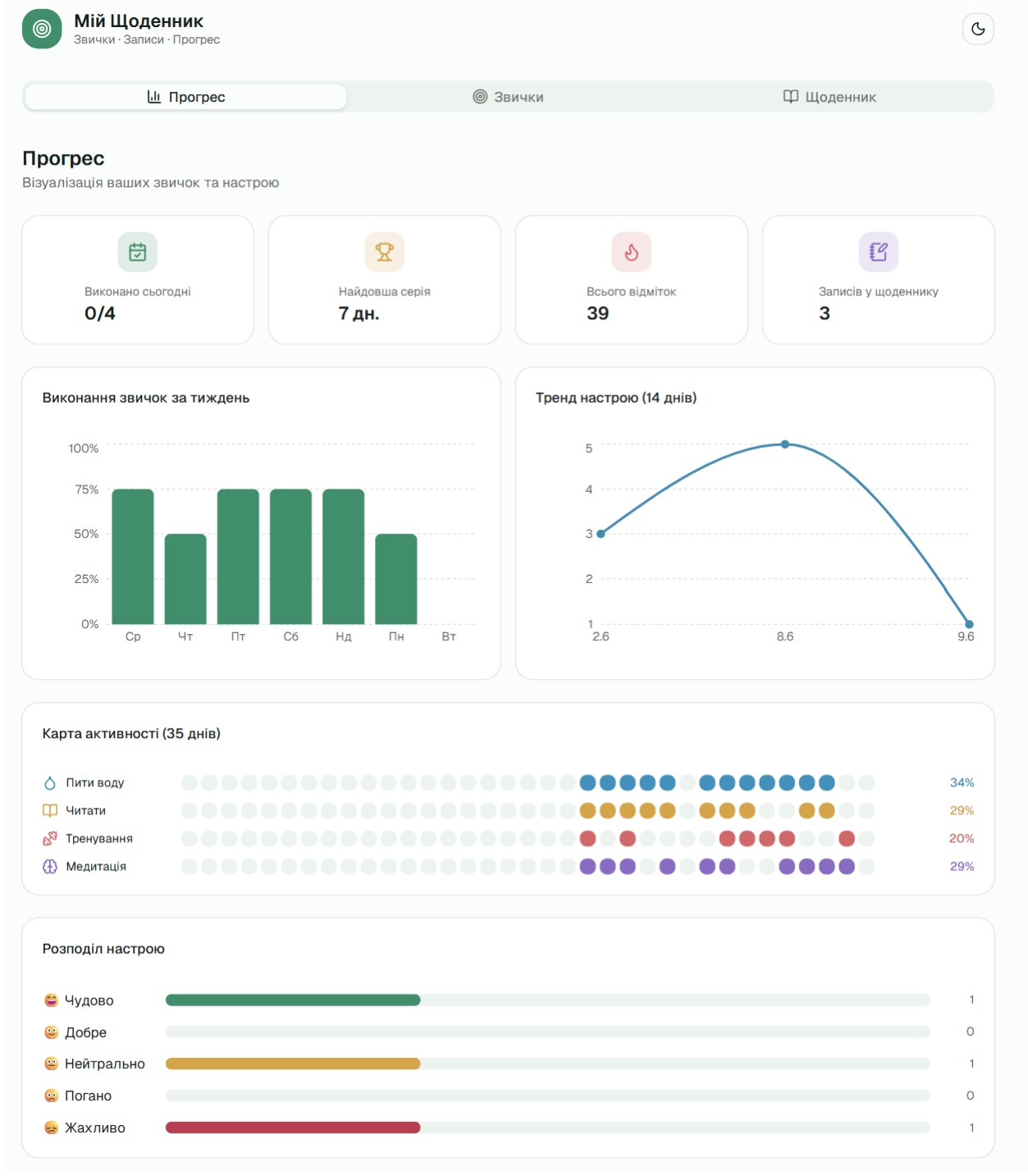


Рисунок В.1 – Сторінка прогресу

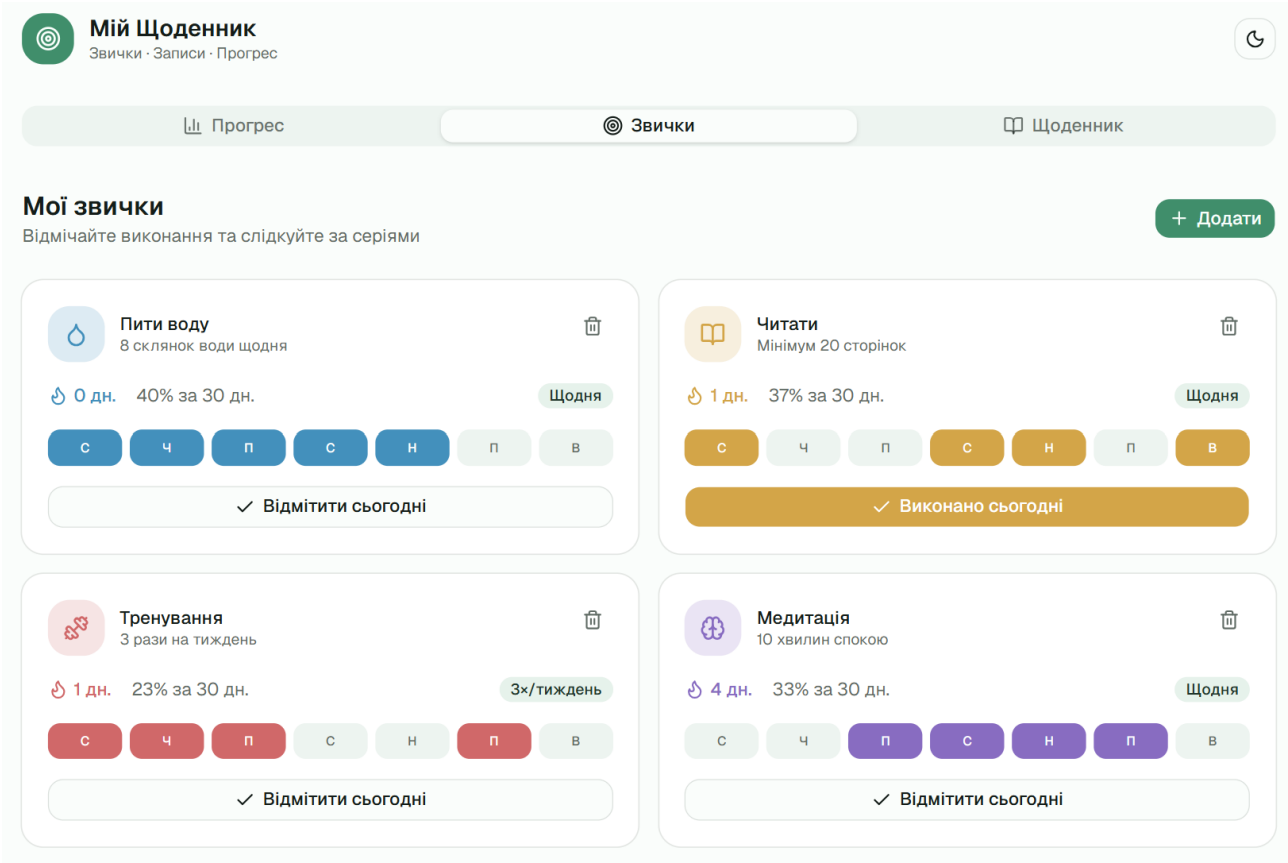


Рисунок В.2 – Сторінка звичок

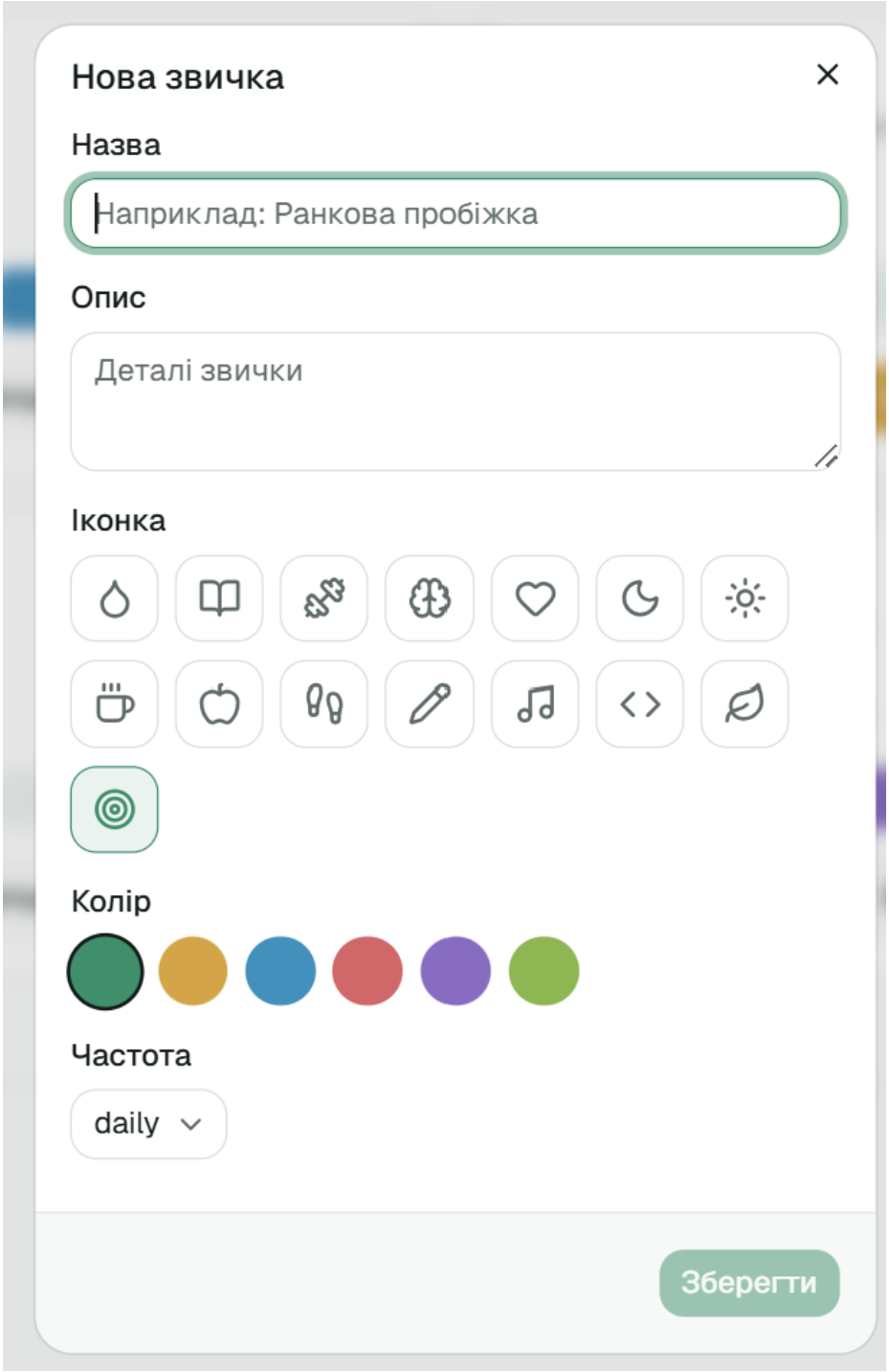


Рисунок В.3 – Вікно додавання звички

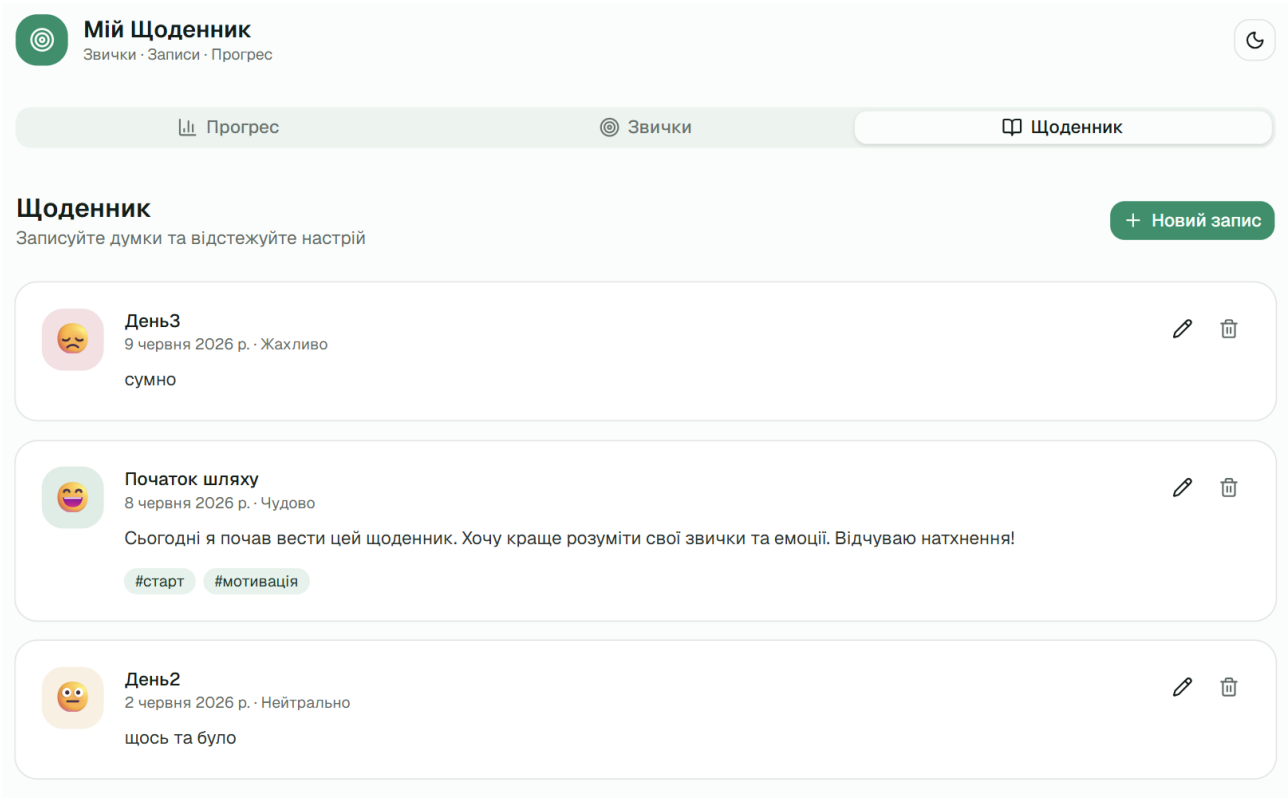


Рисунок В.4 – Сторінка щоденника

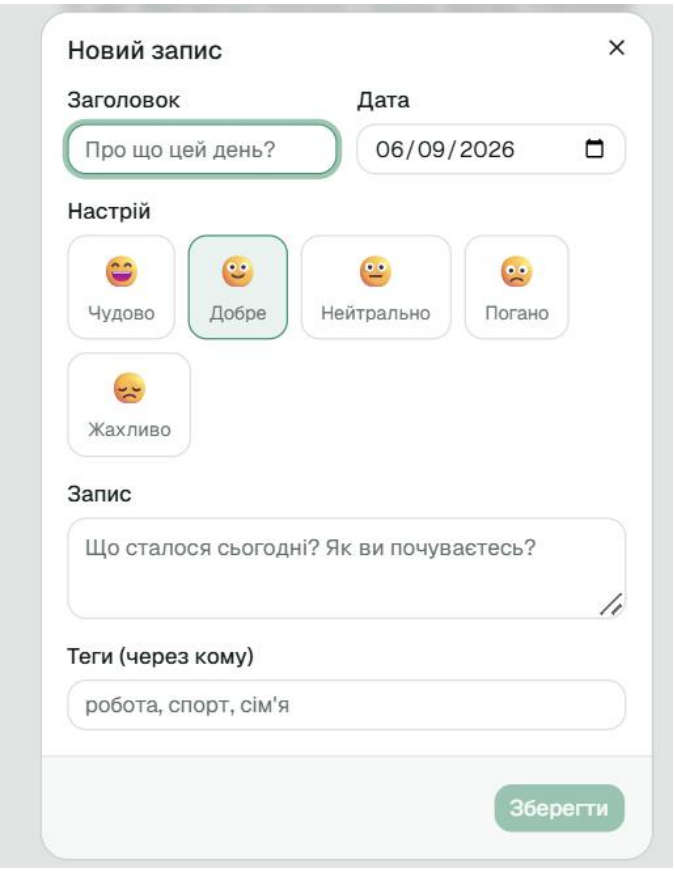


Рисунок В.5 – Вікно додавання запису