ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного забезпечення комплексного моніторингу та корекції індивідуальних поведінкових патернів користувача»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Нікіта ВОЛОШИН
(підпис)

Виконав: здобувач вищої освіти групи ПД-43
Нікіта ВОЛОШИН

Керівник: Владислав ЯСКЕВИЧ
канд. техн. наук, доц.

Рецензент:_____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти Бакалавр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Ірина ЗАМРІЙ
« ____ » _____2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Волошину Нікіті Назарійович

1. Тема кваліфікаційної роботи: «Розробка програмного забезпечення комплексного моніторингу та корекції індивідуальних поведінкових патернів користувача»
керівник кваліфікаційної роботи канд. техн. наук, доцент Владислав ЯСКЕВИЧ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи моніторингу та корекції індивідуальних поведінкових патернів користувача, опис підходів до формування та відстеження звичок, технічна документація з описом фреймворків і бібліотек для розробки клієнт-серверних вебзастосунків.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної галузі моніторингу та корекції індивідуальних поведінкових патернів користувача, аналіз існуючих програмних рішень та обґрунтування технологічного стеку для розробки вебзастосунку HabitFlow.
2. Проєктування вебзастосунку HabitFlow: визначення вимог, розробка архітектури, проєктування бази даних, REST API, основних алгоритмів та інтерфейсу користувача.

3. Програмна реалізація вебзастосунку HabitFlow: реалізація серверної та клієнтської частин, модулів автентифікації, керування звичками, журналу виконання, статистики, рекомендацій, нагадувань і CSV-експорту.
4. Тестування вебзастосунку HabitFlow та оцінювання ефективності програмного продукту.

5. Перелік графічного матеріалу: *презентація*
    1. Аналіз аналогів.
    2. Вимоги до програмного забезпечення.
    3. Програмні засоби реалізації.
    4. Діаграма варіантів використання.
    5. Алгоритм роботи вебзастосунку.
    6. Діаграма класів.
    7. Структура бази даних.
    8. Екранні форми.
    9. Апробація результатів дослідження.

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-07.03.2026	
3	Огляд підходів до моніторингу звичок, регулярних дій та поведінкових патернів користувача	08.03-18.03.2026	
4	Визначення вимог і проєктування вебзастосунку HabitFlow	19.03-26.03.2026	
5	Програмна реалізація серверної та клієнтської частин застосунку	27.03-24.04.2026	
6	Тестування вебзастосунку HabitFlow	25.04-30.04.2026	
7	Оформлення роботи: вступ, висновки, реферат, перелік посилань і додатки	01.05-05.05.2026	
8	Розробка демонстраційних матеріалів	06.05-10.05.2026	
9	Попередній захист роботи	11.05-22.05.2026	

Здобувач вищої освіти _____ Нікіта ВОЛОШИН
(підпис)

Керівник
кваліфікаційної роботи _____ Владислав ЯСКЕВИЧ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 96 стор., 8 табл., 17 рис., 31 джерел.

Мета роботи – підвищення зручності контролю та корекції індивідуальних поведінкових патернів користувача шляхом проєктування та розробки вебзастосунку HabitFlow з модульною архітектурою, підсистемою моніторингу та побудовою поведінкових метрик.

Об'єкт дослідження – програмне забезпечення для комплексного моніторингу та корекції індивідуальних поведінкових патернів користувача.

Предмет дослідження – архітектура, функціональні модулі та алгоритми вебзастосунку HabitFlow для створення звичок, фіксації виконання і пропусків, розрахунку поведінкових метрик, формування статистики та рекомендацій.

Короткий зміст роботи: В роботі проаналізовано підходи до цифрового моніторингу регулярних дій користувача та підтримки процесу формування стабільних звичок. Розглянуто програмні рішення, які використовуються для відстеження звичок, планування активностей, фіксації виконання та нагадувань, зокрема Loop Habit Tracker, Habitica, Todoist, Streaks і Google Calendar. Під час аналізу визначено їхні переваги та обмеження з погляду комплексного моніторингу поведінкових патернів, збереження історії, побудови статистики та надання рекомендацій користувачу.

Сферою використання застосунку є персональний моніторинг регулярних дій, контроль виконання звичок, аналіз поведінкової стабільності та підтримка користувача в процесі формування корисних індивідуальних патернів.

КЛЮЧОВІ СЛОВА: ВЕБЗАСТОСУНОК, ПОВЕДІНКОВІ ПАТЕРНИ, ЗВИЧКИ, МОНІТОРИНГ, КОРЕКЦІЯ, SPRING BOOT, REACT, POSTGRESQL, JWT, REST API, DOCKER, CSV.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

API – Application Programming Interface

CI/CD – Continuous Integration / Continuous Delivery

CRUD – Create, Read, Update, Delete

CSS – Cascading Style Sheets

CSV – Comma-Separated Values

DTO – Data Transfer Object

ER – Entity-Relationship

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

IDE – Integrated Development Environment

JPA – Java Persistence API

JS – JavaScript

JSON – JavaScript Object Notation

JWT – JSON Web Token

ORM – Object-Relational Mapping

REST – Representational State Transfer

SPA – Single Page Application

SQL – Structured Query Language

UI – User Interface

UML – Unified Modeling Language

UX – User Experience

ВСТУП

Повторювані дії займають помітне місце у повсякденному житті людини. Це може бути навчання після роботи, фізична активність кілька разів на тиждень, контроль сну, споживання води, читання або коротке планування дня. Окремо така дія здається незначною. Але якщо вона повторюється протягом певного часу, вона поступово формує індивідуальний поведінковий патерн.

На практиці користувач не завжди бачить власну регулярність точно. Може здаватися, що звичка виконується стабільно, хоча фактично вона пропускається кілька разів на тиждень. Буває й навпаки: користувач недооцінює прогрес, бо не має під рукою історії виконання. Один пропуск ще не показує проблему. Проте кілька пропусків поспіль або повторення невиконання в одні й ті самі дні вже дають підставу переглянути графік.

Звичайний список справ не повністю вирішує цю задачу. Він показує, що потрібно зробити, але не завжди дає відповідь, як часто дія справді виконувалася, коли були пропуски і чи формується стабільна серія. Для роботи зі звичками потрібна система, яка не тільки нагадує про дію, а й зберігає результат: виконано, пропущено або немає запису. Саме ці дані надалі потрібні для історії, статистики та рекомендацій.

Існуючі цифрові рішення мають різний підхід до цієї проблеми. Одні застосунки більше підходять для календарного планування, інші — для швидкого відмічання звичок, мотивації або ведення списку задач. Проте в одному інструменті не завжди поєднано все, що потрібно для повного моніторингу: створення звички, розклад, нагадування, фіксація виконання і пропуску, причина пропуску, історія, статистика, рекомендації та експорт даних.

У межах кваліфікаційної роботи розробляється вебзастосунок HabitFlow. Він призначений для моніторингу та корекції індивідуальних поведінкових патернів користувача через роботу зі звичками. У застосунку користувач створює звичку, задає її назву, категорію, опис, дні виконання, цільову кількість днів на тиждень і

час нагадування. Після цього звичка з’являється на головній сторінці у ті дні, коли вона запланована.

У HabitFlow основна взаємодія зроблена простою. Користувач бачить активні звички на сьогодні та може натиснути «Виконано» або «Пропуск». Якщо дія пропущена, можна вказати причину. Далі ці записи потрапляють до журналу виконання. На їх основі формується календарна історія, статистика, поточна серія та рекомендації. Окремо передбачено автоматичне визначення пропусків для запланованих днів, у які користувач не зробив жодної відмітки.

Актуальність роботи полягає в необхідності створення зручного вебінструменту, який допомагає користувачу бачити не тільки список бажаних дій, а й реальний стан їх виконання. Для цього важливо поєднати планування, щоденну фіксацію, історію, статистику й короткі практичні підказки в одному застосунку.

Мета роботи – підвищення зручності контролю та корекції індивідуальних поведінкових патернів користувача шляхом проєктування та розробки вебзастосунку HabitFlow з модульною архітектурою, підсистемою моніторингу та побудовою поведінкових метрик.

Об’єкт дослідження – програмне забезпечення для комплексного моніторингу та корекції індивідуальних поведінкових патернів користувача.

Предмет дослідження – архітектура, функціональні модулі та алгоритми вебзастосунку HabitFlow для створення звичок, фіксації виконання і пропусків, розрахунку поведінкових метрик, формування статистики та рекомендацій.

Для досягнення мети потрібно виконати такі завдання:

1. Проаналізувати предметну галузь моніторингу та корекції поведінкових патернів користувача.
2. Розглянути існуючі програмні рішення для відстеження звичок, планування регулярних дій і нагадувань.
3. Визначити функціональні та нефункціональні вимоги до HabitFlow.
4. Обґрунтувати вибір технологій для серверної та клієнтської частин застосунку.

5. Спроєктувати архітектуру системи, структуру бази даних, основні сутності та сценарії взаємодії.
6. Реалізувати backend-частину з REST API, авторизацією, роботою зі звичками, журналом виконання, статистикою, рекомендаціями та CSV-експортом.
7. Реалізувати frontend-частину зі сторінками входу, реєстрації, огляду, звичок, історії, рекомендацій і статистики.
8. Додати автоматичну фіксацію пропусків для запланованих днів без відміток.
9. Реалізувати браузерні нагадування та експорт статистики у форматі CSV.
10. Провести тестування основних функцій вебзастосунку.

Під час виконання роботи використано аналіз предметної галузі, порівняння аналогів, об’єктно-орієнтоване проєктування, проєктування реляційної бази даних, розробку REST API, функціональне та модульне тестування. Для опису системи застосовано UML-діаграми, ER-діаграму, таблиці вимог і сценарії взаємодії.

Серверну частину HabitFlow реалізовано мовою Java з використанням Spring Boot, Spring Security, JWT, PostgreSQL, Liquibase та Swagger/OpenAPI. Клієнтську частину створено на React із використанням Vite, JavaScript, Axios, React Router і Recharts. Для запуску компонентів застосунку використовується Docker Compose.

Практична значущість роботи полягає у створенні вебзастосунку для персонального контролю звичок і регулярних дій. HabitFlow зберігає історію виконання, враховує пропуски, формує статистику та рекомендації. Розроблений застосунок може використовуватися користувачем для щоденного самоконтролю, аналізу регулярності власних дій і корекції графіка виконання звичок. Окремі результати дослідження було апробовано на Всеукраїнській науково-технічній конференції «Технології цифрового розвитку: програмні системи та децентралізація», 1–3 травня 2026 року, м. Київ, ДУІКТ, а також на VI Всеукраїнській науково-практичній конференції «Сучасні інтелектуальні

інформаційні технології в науці та освіті», 15 травня 2026 року, м. Київ, ДУІКТ. У майбутньому систему можна розширити мобільною версією, push-сповіщеннями й детальнішою аналітикою.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, переліку посилань і додатків. У роботі розглянуто предметну галузь, засоби реалізації, проєктування системи, програмну реалізацію, екранні форми та результати тестування HabitFlow.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ОБҐРУНТУВАННЯ ЗАСОБІВ РОЗРОБКИ ВЕБЗАСТОСУНКУ HABITFLOW

Предметна галузь моніторингу та корекції індивідуальних поведінкових патернів пов’язана з відстеженням регулярних дій користувача, аналізом їх виконання та виявленням повторюваних закономірностей у повсякденній поведінці. До таких дій можуть належати навчання, фізична активність, контроль режиму сну, споживання води, читання, виконання робочих завдань, відпочинок або інші процеси, які користувач прагне зробити стабільною частиною власного дня.

На відміну від звичайного списку справ, моніторинг поведінкових патернів передбачає не лише створення переліку дій, а й накопичення даних про їх фактичне виконання. Користувачу важливо бачити, які звички виконуються стабільно, які часто пропускаються, як змінюється прогрес протягом тижня або місяця, а також чи є потреба скоригувати графік. Саме тому програмне забезпечення для такої предметної галузі має поєднувати планування, фіксацію результатів, історію, статистику, нагадування та рекомендації.

У межах цієї кваліфікаційної роботи розглядається вебзастосунок HabitFlow. Він призначений для комплексного моніторингу поведінкових патернів користувача через роботу зі звичками. Система дозволяє створювати регулярні дії, задавати категорію, опис, частоту виконання, цільову кількість днів на тиждень, конкретні дні виконання та час нагадування. Після цього користувач може фіксувати виконання або пропуск, переглядати історію, аналізувати статистику та отримувати рекомендації для покращення регулярності.

1.1 Поняття поведінкових патернів користувача

Поведінковий патерн користувача можна розглядати як дію або групу дій, які повторюються протягом певного часу. Це не обов’язково щось складне. Наприклад, людина навчається після роботи, п’є воду вранці, кілька разів на тиждень займається спортом або планує день перед початком занять. Якщо така дія повторюється регулярно, вона поступово стає частиною звичного ритму.

Одна виконана дія ще не є патерном. Вона може бути випадковою. Патерн з’являється тоді, коли повторення можна простежити в часі. Саме тому для аналізу звичок недостатньо дивитися лише на поточний день. Потрібна історія: що було виконано вчора, які дні були пропущені, чи зберігається серія, чи змінюється регулярність протягом тижня або місяця. Питання опису поведінкових змін і технік формування регулярної поведінки розглядається у роботі S. Michie та співавторів [1].

У межах цієї роботи поведінковий патерн розглядається через звичку. Звичка має конкретні параметри: назву, категорію, опис, дні виконання, цільову кількість днів на тиждень і час нагадування. Наприклад, для звички “Навчання Java” можна задати виконання у понеділок, середу та п’ятницю о 19:00. Після цього її вже можна не просто згадувати, а нормально відстежувати.

Для аналізу важливо фіксувати не тільки виконання, а й пропуски. Якщо дія не була виконана один раз, це ще не означає проблему. Але якщо пропуски повторюються в одні й ті самі дні, це вже дає підставу переглянути графік. Наприклад, якщо користувач постійно не виконує звичку в п’ятницю ввечері, можливо, цей час просто невдалий. У такому випадку краще змінити день або зменшити навантаження.

У HabitFlow для цього використовується журнал виконання. Кожна відмітка зберігається окремо: дія може бути виконана, пропущена або залишитися без запису. Стан “немає запису” також важливий. Він означає, що користувач не зробив жодної відмітки в запланований день. Щоб такі дні не випадали зі статистики, у застосунку передбачено автоматичну фіксацію пропусків.

Поведінкові патерни можуть бути позитивними й проблемними. Позитивний патерн видно тоді, коли користувач стабільно виконує заплановану дію: регулярно навчається, підтримує водний баланс, читає або займається фізичною активністю. Проблемний патерн проявляється інакше: дія часто відкладається, пропускається або виконується нерівномірно. У такій ситуації важливо побачити не просто факт невиконання, а повторюваність проблеми.

Корекція поведінкового патерну в цій роботі не розглядається як медичний або психологічний вплив. Йдеться про прикладну програмну підтримку. Застосунок аналізує виконання звички, кількість пропусків і поточну серію, після чого формує коротку рекомендацію. Наприклад, можна запропонувати почати з меншої кількості днів на тиждень або відновити серію з найближчого запланованого дня.

Такий підхід відрізняє HabitFlow від звичайного списку справ. У списку справ дія або виконана, або ні. У HabitFlow звичка має історію, статистику й контекст. Користувач бачить, як змінюється регулярність, які дії виконуються стабільно, а які потребують перегляду. Це дає змогу працювати не з окремими відмітками, а з поведінкою в динаміці.

У межах кваліфікаційної роботи поведінковий патерн розуміється як регулярна дія користувача, яку можна запланувати, зафіксувати, переглянути в історії, проаналізувати та за потреби скоригувати. Саме на цій логіці побудовано вебзастосунок HabitFlow.

1.2 Особливості формування та підтримки регулярних дій

Формування регулярної дії починається з конкретного опису. Якщо користувач записує звичку занадто загально, її важко перевірити. Наприклад, фраза «займатися спортом» не показує, коли саме треба виконати дію і скільки разів на тиждень вона має повторюватися. Інший варіант — «займатися спортом тричі на тиждень о 19:00» — уже має графік, час і зрозумілий результат.

У HabitFlow звичка створюється як структурований запис. Для неї задаються назва, опис, категорія, дні виконання, цільова кількість днів на тиждень і час нагадування. Це потрібно не лише для зручного відображення в інтерфейсі. Такі параметри дають змогу системі зрозуміти, коли звичка має бути активною, коли її потрібно показати на головній сторінці та які дні враховувати в історії.

Регулярна дія відрізняється від разового завдання. Разову справу можна виконати один раз і закрити. Звичка повторюється багато разів, тому для неї важливий не тільки поточний статус, а й накопичена історія. Наприклад, «підготувати презентацію» — це окреме завдання, а «навчатися Java у понеділок, середу та п’ятницю» — регулярна дія, яку потрібно відстежувати протягом певного часу.

Графік звички має бути реальним. Якщо користувач одразу ставить складну дію на кожен день, кількість пропусків може швидко зрости. Це не завжди означає, що дія непотрібна. Часто графік просто не підходить. Тому в HabitFlow можна вибрати конкретні дні тижня: для одних звичок залишити щоденне виконання, а для інших — тільки кілька днів.

Важливо, щоб фіксація результату була короткою. Користувач відкриває головну сторінку, бачить актуальні звички на сьогодні й натискає «Виконано» або «Пропуск». Якщо для однієї відмітки потрібно багато дій, користувач поступово перестає вносити дані, а статистика стає неповною.

Пропуск також є важливою частиною аналізу. Він може виникнути через втому, брак часу, зміну планів або невдало підібраний день. У HabitFlow користувач може не лише зафіксувати пропуск, а й указати його причину. Це допомагає пізніше зрозуміти, чи проблема в самій звичці, чи в її розкладі.

Окремо враховуються дні без відміток. Якщо звичка була запланована, але користувач не натиснув ні «Виконано», ні «Пропуск», такий день не повинен зникати зі статистики. Для цього в застосунку передбачено автоматичну фіксацію пропусків для запланованих днів без запису.

Підтримка регулярних дій у HabitFlow будується навколо кількох елементів: графіка, швидкої відмітки, журналу виконання, історії, статистики, нагадувань і

рекомендацій. Разом вони дозволяють користувачу бачити не лише список звичок, а й реальну регулярність їх виконання.

1.3 Роль цифрових інструментів у моніторингу звичок

Цифрові інструменти допомагають користувачу не тримати всі регулярні дії в пам’яті. Більшість звичок не має чіткої кінцевої дати. Вони повторюються щодня, кілька разів на тиждень або за індивідуальним графіком. Без окремого інструменту складно швидко зрозуміти, що вже виконано, що пропущено, а які дії залишилися без уваги.

Для звичок звичайного списку справ недостатньо. У списку задача найчастіше має один стан: виконана або не виконана. Звичка працює інакше, бо її потрібно оцінювати в часі. Наприклад, якщо користувач планує навчатися Java у понеділок, середу і п’ятницю, важливо бачити не тільки поточну відмітку, а й попередні результати. Саме історія показує, чи дія справді стає регулярною.

На практиці цифровий моніторинг складається з кількох етапів. Спочатку користувач створює звичку і задає її параметри. Потім застосунок показує цю дію у потрібний день або нагадує про неї. Після цього користувач фіксує результат: «Виконано» або «Пропуск». На основі таких записів формується історія, статистика і рекомендації.

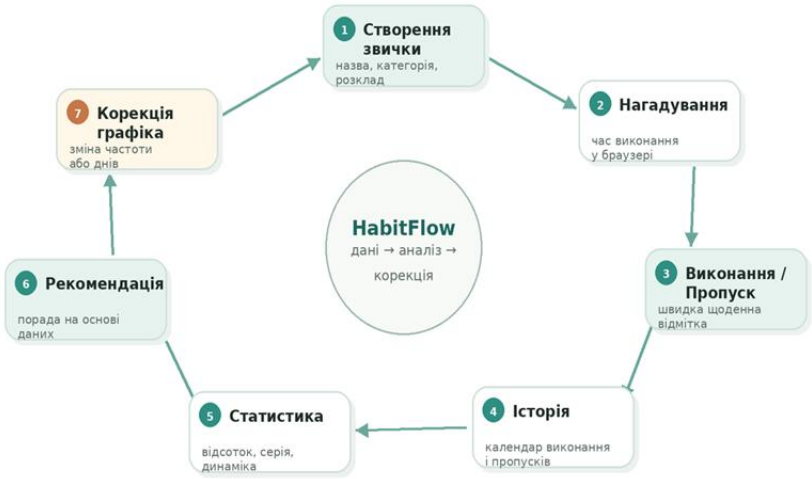


Рис. 1.1 Цикл цифрового моніторингу звички

У цьому циклі кожен етап має свою роль. Створення звички визначає, яку дію потрібно контролювати. Нагадування допомагає не забути про неї. Відмітка виконання або пропуску перетворює дію на конкретний запис у системі. Історія дає змогу повернутися до попередніх днів, а статистика показує загальну картину. Рекомендації потрібні вже після цього, коли накопичено достатньо даних для простого висновку.

Цифровий інструмент зменшує суб’єктивність оцінки. Користувачу може здаватися, що він регулярно виконує певну дію, але календар або статистика покажуть інший результат. Наприклад, звичка могла виконуватися лише двічі на тиждень замість п’яти разів. Буває й навпаки: користувач думає, що прогрес слабкий, хоча історія показує стабільну серію.

Окрема перевага таких інструментів — робота з різними типами звичок. Одні дії справді зручно виконувати щодня. Інші краще планувати тільки на окремі дні. У HabitFlow це враховано через налаштування днів тижня. Якщо звичка активна тільки в понеділок, середу і п’ятницю, вона не повинна впливати на статистику в інші дні.

Для користувача важливо швидко бачити поточний стан. Коли застосунок відкривається лише для короткої відмітки, інтерфейс не має змушувати проходити зайві кроки. На головній сторінці HabitFlow показуються активні звички на сьогодні, прогрес дня, кількість виконаних і пропущених дій. Це дає змогу одразу зрозуміти ситуацію без переходу між багатьма сторінками.

Ще одна роль цифрових інструментів — збереження історії. Якщо кожна відмітка потрапляє до журналу, система поступово накопичує матеріал для аналізу. Через тиждень або місяць користувач може побачити, які звички стали стабільнішими, а які досі часто пропускаються. Без журналу такі висновки довелося б робити вручну або взагалі на рівні відчуттів.

У HabitFlow окремо враховано ситуацію, коли користувач не зробив жодної відмітки. У реальному використанні це трапляється часто: людина могла не відкрити застосунок, забути поставити результат або не мати часу. Якщо звичка

була запланована, такий день не варто просто ігнорувати. Тому в застосунку передбачено автоматичну фіксацію пропусків для запланованих днів без записів.

Ручний пропуск теж залишається важливим. Користувач може сам натиснути «Пропуск» і вказати причину. Це дає більше контексту для подальшого аналізу. Наприклад, пропуск через втому і пропуск через нестачу часу мають різне значення. Якщо одна причина повторюється часто, користувач може зрозуміти, що проблема не в самій звичці, а в її графіку або складності.

У межах HabitFlow цифровий моніторинг реалізується через кілька пов’язаних частин: модуль звичок, журнал виконання, головну сторінку, історію, статистику, рекомендації та нагадування. Разом вони переводять самоконтроль із приблизних відчуттів у більш зрозумілу форму, де користувач бачить не тільки план, а й фактичний результат.

1.4 Аналіз існуючих програмних рішень для відстеження звичок і регулярних дій

У сфері відстеження звичок уже існує багато цифрових рішень. Одні з них більше орієнтовані на щоденну відмітку, інші — на календарне планування, статистику, нагадування або мотивацію. Для розробки HabitFlow було розглянуто кілька таких застосунків: Loop Habit Tracker, Habitica, Todoist, Streaks і Google Calendar.

Під час аналізу увага приділялася не всім можливостям цих сервісів, а тим функціям, які важливі для теми роботи: створення звичок, вибір графіка, фіксація виконання і пропусків, історія, статистика, нагадування, рекомендації та вебдоступність. Такий підхід дозволяє зрозуміти, які ідеї можна врахувати в HabitFlow, а які обмеження потрібно закрити власною реалізацією. Для порівняння було обрано Loop Habit Tracker, Habitica, Todoist, Streaks і Google Calendar, оскільки ці рішення представляють різні підходи до контролю звичок, задач, календарного планування та нагадувань [2–6].

1.4.1 Аналіз застосунку Loop Habit Tracker

Loop Habit Tracker — це мобільний застосунок для відстеження звичок, орієнтований переважно на Android-користувачів. Функціональні можливості застосунку підтверджуються його офіційним описом і документацією про відстеження регулярності, історії та статистики звичок [2]. Його основна логіка проста: користувач створює звичку, задає регулярність і щоденно відмічає виконання. Для такого типу застосунків це важливо, бо сама відмітка не повинна займати багато часу.

Сильною стороною Loop Habit Tracker є статистика. У застосунку є історія виконання, графіки та показник сили звички. Користувач бачить не лише поточну серію, а й загальну динаміку: якщо дія виконується стабільно, показники покращуються, а при пропусках поступово знижуються.

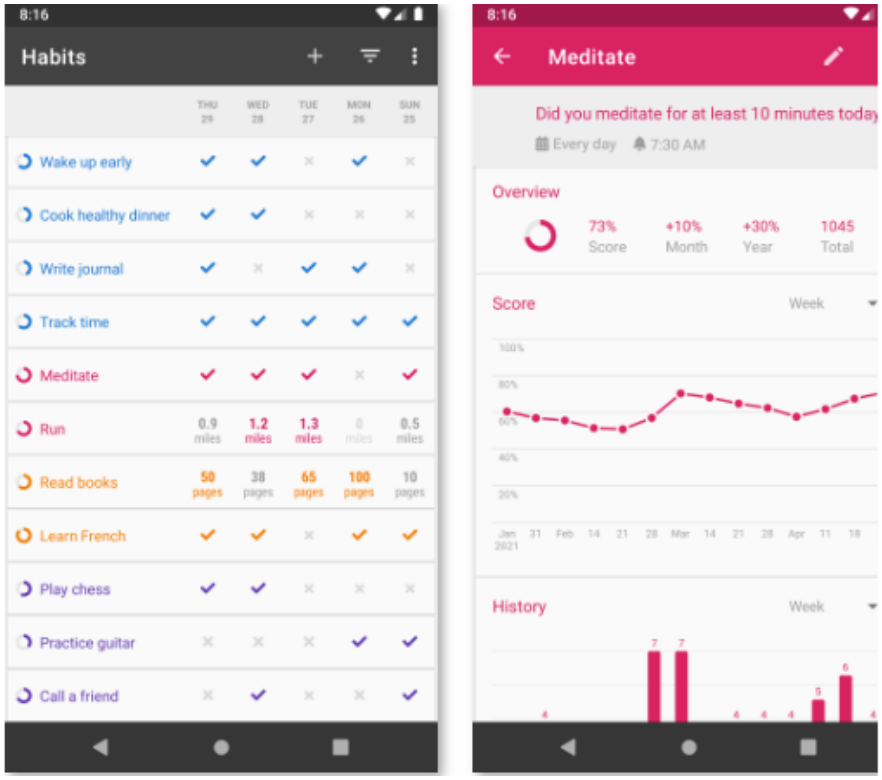


Рис. 1.2 Головний екран застосунку Loop Habit Tracker

До переваг Loop Habit Tracker можна віднести:

– простий інтерфейс для щоденного використання;

– гнучке налаштування регулярності;

– наявність історії, статистики та графіків;

– можливість працювати без обов’язкової реєстрації;

– локальне збереження даних.

Для задачі цієї кваліфікаційної роботи Loop має й обмеження. Насамперед це мобільний застосунок, тоді як HabitFlow розробляється як вебзастосунок із серверним збереженням даних. Крім того, Loop добре показує статистику, але не робить основний акцент на рекомендаціях і автоматичній обробці пропусків.

Loop Habit Tracker можна вважати вдалим прикладом класичного трекера звичок. У HabitFlow було доцільно врахувати його просту щоденну відмітку, історію та статистику, але доповнити ці можливості вебдоступом, персональним обліковим записом, автоматичною фіксацією пропусків, рекомендаціями та CSV-експортом.

1.4.2 Аналіз застосунку Habitica

Habitica — це застосунок для звичок і задач, який використовує ігрову механіку. Замість звичайного списку справ користувач отримує персонажа, досвід, золото та винагороди. Виконання корисних дій покращує прогрес персонажа, а невиконання регулярних задач може мати негативний ефект. Офіційний опис Habitica визначає застосунок як інструмент керування звичками та задачами з використанням ігрових механік [3].

У застосунку є кілька типів задач: звички, щоденні задачі та разові справи. Завдяки цьому Habitica можна використовувати не тільки для контролю регулярних дій, а й для загальної самоорганізації. Такий підхід добре працює для користувачів, яким потрібна додаткова мотивація через гру.

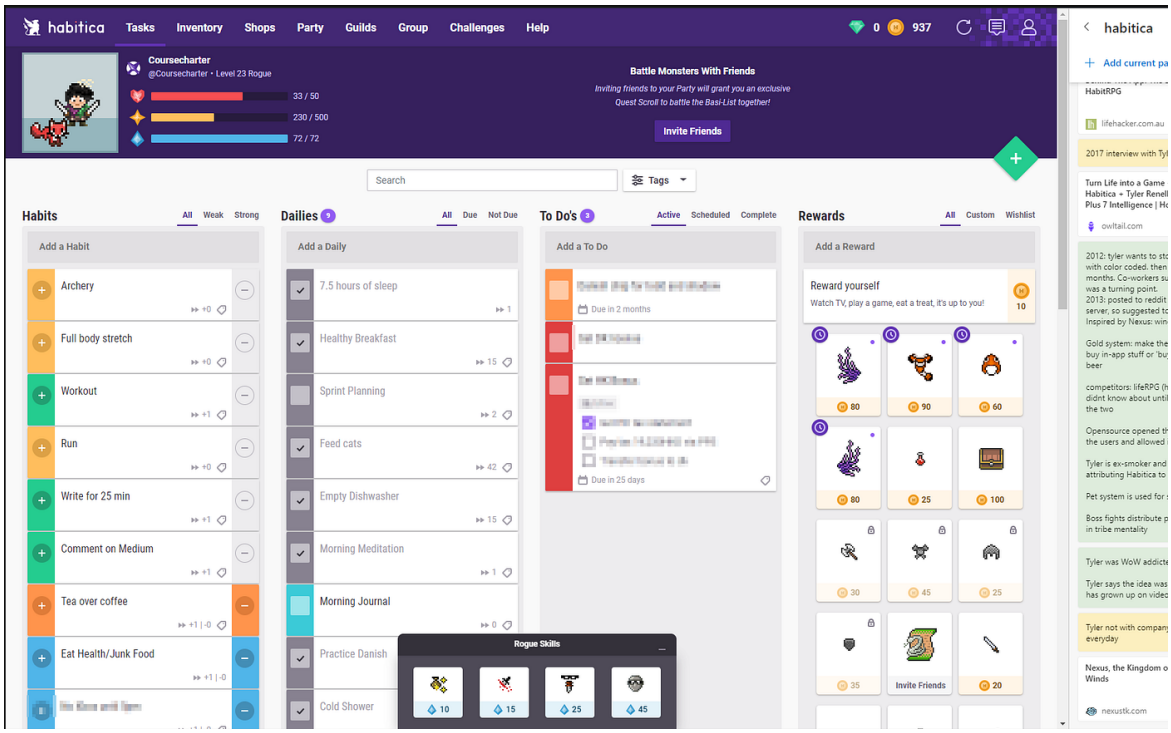


Рис. 1.3 Приклад гейміфікованого списку завдань у Habitica

До переваг Habitica можна віднести:

– ігрову мотивацію;

– поділ задач на звички, щоденні задачі та разові справи;

– систему винагород, рівнів і персонажа;

– можливість використовувати застосунок для навчання, роботи, спорту та побутових справ.

Разом з тим, гейміфікація підходить не всім користувачам. Для одних вона підвищує зацікавленість, а для інших може відволікати від головної мети — спокійного контролю регулярних дій. Крім того, Habitica більше орієнтована на мотивацію, ніж на детальний аналіз причин пропусків або корекцію графіка.

У контексті HabitFlow цей застосунок корисний як приклад зворотного зв’язку після виконання дії. У Habitica таким зворотним зв’язком є ігрова винагорода. У HabitFlow цю роль виконують прогрес дня, серія виконання, статистика та рекомендації. Тому для власного застосунку було обрано менш ігровий, але більш аналітичний підхід.

1.4.3 Аналіз платформи Todoist

Todoist — це вебплатформа та мобільний застосунок для керування задачами. На відміну від спеціалізованих трекерів звичок, він орієнтований не тільки на регулярні дії, а й на робочі справи, проєкти, дедлайни, пріоритети, мітки та фільтри. Тому Todoist можна розглядати як універсальний інструмент для організації задач. Підтримка повторюваних задач, проєктів, міток і фільтрів описана в офіційних матеріалах Todoist [4].

Для теми цієї роботи Todoist цікавий підтримкою повторюваних задач. Користувач може створити дію, яка повторюється щодня, щотижня або за власним правилом. Після виконання така задача переноситься на наступну дату. Це зручно для регулярних дій, які мають чіткий день або час виконання.

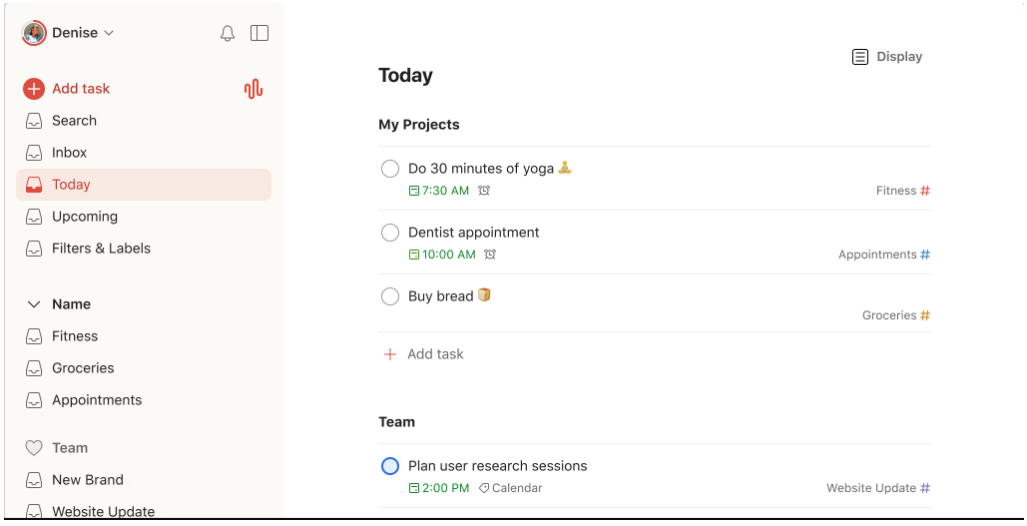


Рис. 1.4 Інтерфейс керування задачами у Todoist

До переваг Todoist можна віднести:

– швидке створення задач;

– підтримку повторюваних дат;

– нагадування для задач із конкретним часом;

– мітки, фільтри та пріоритети;

– доступність через вебверсію та мобільні застосунки.

Разом з тим, Todoist не є спеціалізованою системою для аналізу поведінкових патернів. Повторювана задача показує, що потрібно зробити, але не дає повної картини звички: скільки разів вона виконувалася, коли були пропуски, яка поточна серія і чи потрібно змінити графік. Для звичайного task management цього достатньо, але для моніторингу звичок — ні.

У HabitFlow основний акцент зроблено не на проєктах і пріоритетах, а на регулярності. Застосунок має зберігати історію виконання, фіксувати пропуски, рахувати статистику та формувати рекомендації. Тому Todoist можна використати як приклад зручного планування повторюваних дій, але для задачі цієї роботи потрібне більш спеціалізоване рішення.

1.4.4 Аналіз застосунку Streaks

Streaks — це застосунок для відстеження звичок, який робить акцент на серіях виконання. Його основна ідея полягає в тому, щоб користувач підтримував безперервний ланцюг дій і намагався не пропускати заплановані звички. Особливості Streaks, зокрема робота із серіями виконання, нагадуваннями та інтеграцією з Apple Health, наведені в офіційному описі застосунку [5].

Інтерфейс Streaks побудований навколо карток звичок. Користувач бачить, які дії вже виконані, а які ще залишаються активними. Такий підхід зручний для щоденного використання, бо відмітка займає небагато часу. Також застосунок підтримує нагадування, негативні задачі та, в екосистемі Apple, інтеграцію з Apple Health.

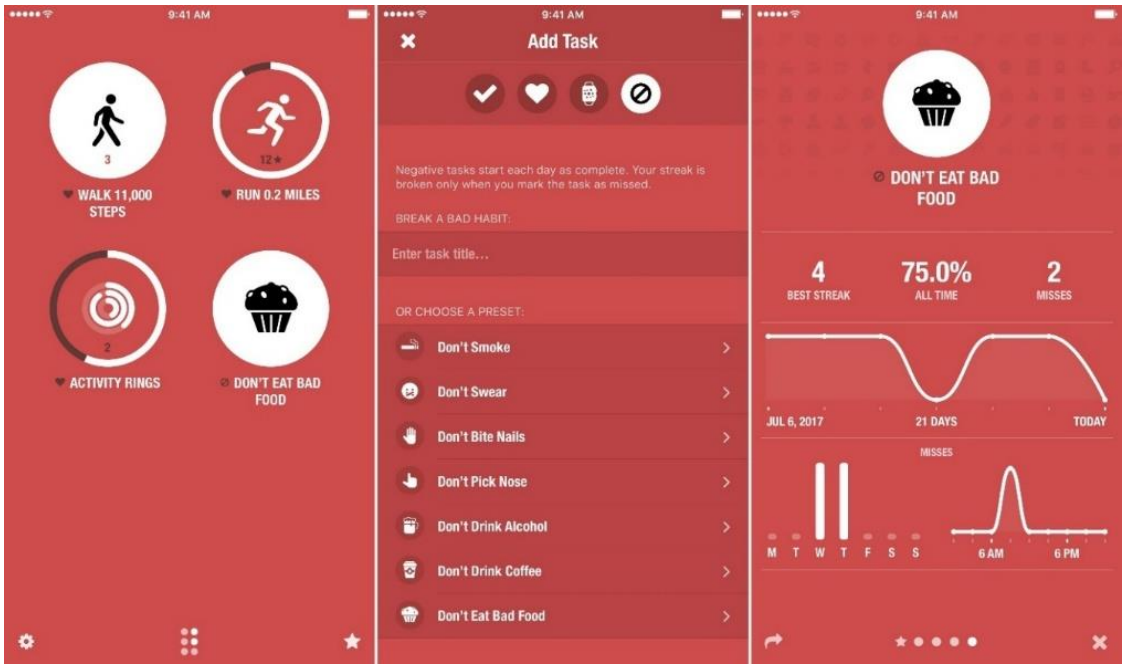


Рис. 1.5 Приклад відстеження серії у Streaks

До переваг Streaks можна віднести:

– простий інтерфейс для щоденного використання;

– сильний акцент на серіях виконання;

– автоматичні нагадування;

– підтримку негативних задач;

– можливість створення задач із таймером;

– інтеграцію з Apple Health;

– зручну візуальну подачу прогресу.

Разом з тим, Streaks має обмеження для задачі цієї кваліфікаційної роботи. Насамперед застосунок орієнтований переважно на користувачів екосистеми Apple. Якщо користувач хоче працювати через браузер на ноутбуці або комп’ютері, такий формат може бути менш зручним. HabitFlow, навпаки, розробляється як вебзастосунок, доступний через браузер.

Ще одне обмеження пов’язане з самим підходом до серій. Серія добре мотивує, але вона не завжди пояснює причину проблеми. Якщо користувач розірвав серію, він бачить факт пропуску, але не завжди розуміє, чому це сталося і

що краще змінити. Для HabitFlow важливо не тільки показати серію, а й зберегти історію, кількість пропусків, причину пропуску та сформувати рекомендацію.

У контексті розробки HabitFlow застосунок Streaks є корисним аналогом через візуальний підхід до звичок. Круглі картки, швидкий доступ до відмітки та зрозуміла подача серії можуть бути вдалими рішеннями для інтерфейсу. У HabitFlow ця ідея частково врахована через картки звичок, поточний статус, прогрес дня і відображення серії.

Streaks зручний для швидкого щоденного контролю. Його переваги полягають у простоті, візуальній зрозумілості, акценті на серіях і тісній інтеграції з Apple Health. Проте для комплексного моніторингу поведінкових патернів потрібне рішення з вебдоступом, серверним збереженням даних, детальнішою історією, обробкою пропусків, рекомендаціями та експортом статистики. Саме ці можливості передбачено у HabitFlow.

1.4.5 Аналіз сервісу Google Calendar

Google Calendar — це сервіс календарного планування, який дозволяє створювати події, нагадування та повторювані записи. Він не є спеціалізованим трекером звичок, але може використовуватися для планування регулярних активностей. Механізм створення повторюваних подій і нагадувань описано в довідкових матеріалах Google Calendar [6].

Для теми цієї роботи Google Calendar цікавий підтримкою повторюваних подій. Наприклад, користувач може створити подію «Навчання Java» на 19:00 і налаштувати повторення у понеділок, середу та п’ятницю.

Google Calendar також підтримує нагадування про події. Для користувача це означає, що сервіс не тільки зберігає план, а й може вчасно нагадати про заплановану дію. Наприклад, перед початком навчання або тренування користувач отримує повідомлення. У цьому Google Calendar добре виконує роль інструмента планування.

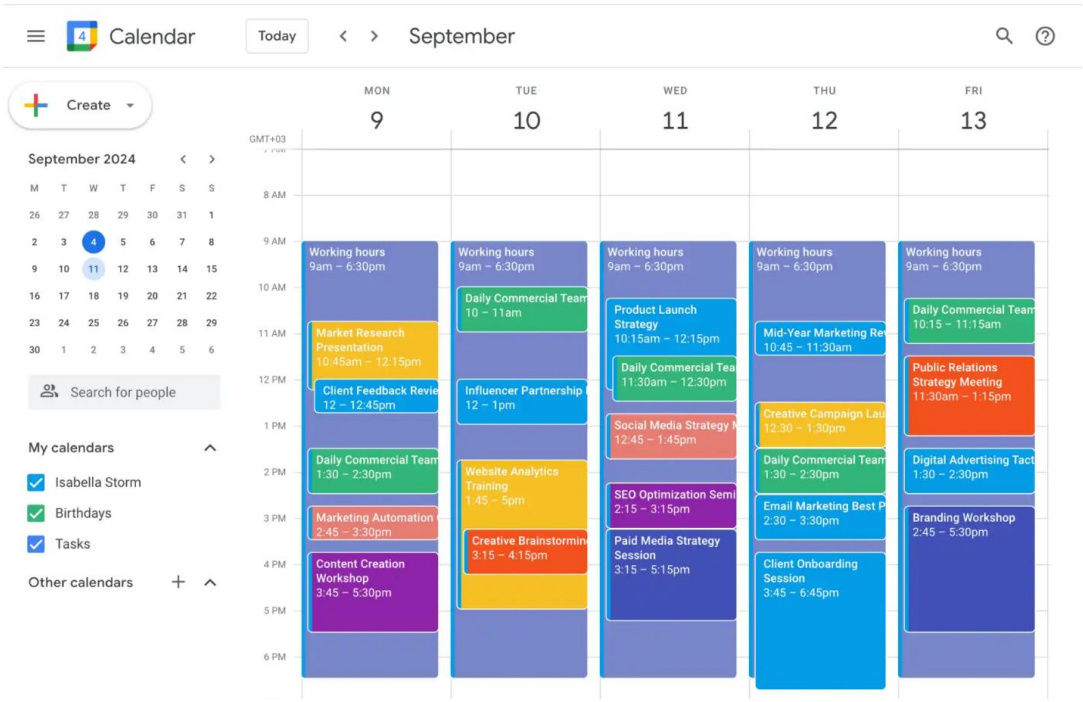


Рис. 1.6 Приклад повторюваної події у Google Calendar

До переваг Google Calendar можна віднести:

– зручне календарне планування;

– підтримку повторюваних подій;

– нагадування;

– синхронізацію між пристроями;

– доступність через вебверсію та мобільні застосунки.

Головне обмеження Google Calendar полягає в тому, що він показує план, але не аналізує фактичне виконання звички. Подія може бути створена в календарі, але сервіс не рахує відсоток виконання, не формує серію, не відокремлює виконання від пропуску і не дає рекомендацій щодо корекції графіка.

У контексті HabitFlow Google Calendar можна розглядати як приклад зручного планування та нагадувань. Проте для комплексного моніторингу поведінкових патернів цього недостатньо. HabitFlow доповнює планування журналом виконання, історією, статистикою, пропусками, рекомендаціями та CSV-експортом.

1.5 Порівняльний аналіз існуючих програмних рішень

Після розгляду окремих програмних рішень доцільно порівняти їх за функціями, які важливі для HabitFlow. Для цього обрано такі критерії: створення звичок, налаштування регулярності, фіксація виконання, робота з пропусками, історія, статистика, нагадування, рекомендації, вебдоступність та експорт даних.

До порівняння включено Loop Habit Tracker, Habitica, Todoist, Streaks, Google Calendar і розроблюваний вебзастосунок HabitFlow. Такий підхід дозволяє побачити, які можливості вже реалізовані в існуючих рішеннях, а які потрібно було врахувати під час розробки власної системи.

Таблиця 1.1

Порівняльний аналіз програмних рішень для моніторингу звичок і регулярних дій

Функціональність / Показник	**Loop Habit Tracker**	**Habitica**	**Todoist**	**Streaks**	**Google Calendar**	**HabitFlow**
Створення звичок	+	+	-	+	-	+
Фіксація виконання	+	+	+	+	-	+
Фіксація пропуску	+	+	+	+	-	+
Збереження причини пропуску	-	-	-	+	-	+
Автоматичне визначення пропущених днів	+	+	-	+	-	+
Календарна історія виконання	+	+	-	+	+	+
Статистика виконання	+	+	+	+	-	+
Відображення серії виконання	+	+	-	+	-	+
Рекомендації щодо корекції звички	-	-	-	-	-	+
Гейміфікація	-	+	-	-	-	+
Вебдоступ через браузер	-	+	+	-	+	+
Серверне збереження даних користувача	-	+	+	+	+	+
Експорт статистики у CSV	-	-	-	-	-	+
Орієнтація саме на поведінкові патерни	+	-	-	+	-	+

За результатами порівняння видно, що кожне рішення має власний сильний бік. Loop Habit Tracker добре підходить для класичного відстеження звичок і статистики. Habitica використовує ігрову мотивацію. Todoist зручний для повторюваних задач. Streaks робить акцент на серіях виконання. Google Calendar добре працює з календарним плануванням.

Водночас жодне з розглянутих рішень не поєднує всі функції, які потрібні для HabitFlow. Для цієї роботи важливими є не тільки створення звички й нагадування, а й фіксація пропусків, причина пропуску, автоматична обробка днів без відмітки, статистика, рекомендації, вебдоступність і CSV-експорт.

Порівняльний аналіз підтверджує доцільність розробки власного вебзастосунку. HabitFlow має враховувати переваги існуючих аналогів, але водночас закривати їхні обмеження за рахунок історії виконання, пропусків, рекомендацій, статистики та доступу через браузер.

1.6 Визначення проблем предметної галузі

Після аналізу існуючих рішень можна виділити кілька проблем, які залишаються актуальними для моніторингу поведінкових патернів. Більшість розглянутих застосунків вирішує лише частину задачі: одні краще працюють із календарем, інші — зі статистикою, мотивацією або швидкою щоденною відміткою.

Першою проблемою є розділеність функцій. Google Calendar добре підходить для планування, але не аналізує фактичне виконання звички. Todoist підтримує повторювані задачі, однак більше орієнтований на керування справами. Loop Habit Tracker і Streaks краще працюють зі звичками, але мають обмеження щодо вебдоступу, рекомендацій або серверного збереження даних.

Друга проблема пов’язана з пропусками. Для аналізу звички важливо знати не тільки те, що дія виконана, а й коли вона була пропущена. Якщо застосунок не відрізняє ручний пропуск від відсутності запису, статистика може бути неточною.

У реальному використанні користувач часто не відкриває застосунок у запланований день, і такий випадок також потрібно враховувати.

Ще одна проблема — відсутність причин пропуску та рекомендацій. Сам факт невиконання не завжди пояснює, що саме завадило користувачу. Причина може бути в нестачі часу, втомі або неправильно вибраному графіку. Без таких даних користувач бачить лише результат, але не завжди розуміє, що потрібно змінити.

Також важливими залишаються вебдоступність і можливість експорту даних. Частина трекерів орієнтована на мобільні платформи або конкретну екосистему. HabitFlow розробляється як вебзастосунок, тому користувач може працювати з ним через браузер. CSV-експорт потрібен для збереження статистики поза системою.

Головна проблема предметної галузі полягає не у відсутності застосунків для звичок, а в тому, що більшість із них не поєднує планування, історію, пропуски, статистику, рекомендації, вебдоступ і експорт даних в одному рішенні. Саме це обґрунтовує розробку HabitFlow.

1.7 Обґрунтування технологічного стеку для розробки HabitFlow

Для реалізації HabitFlow обрано клієнт-серверну архітектуру, оскільки застосунок має працювати з персональними даними користувача, зберігати історію виконання звичок, перевіряти права доступу та формувати статистичні показники на основі накопичених записів. У такій архітектурі клієнтська частина відповідає за інтерфейс і взаємодію з користувачем, а серверна частина виконує бізнес-логіку, авторизацію, роботу з базою даних і підготовку відповідей для frontend-частини.

Серверну частину доцільно реалізувати мовою Java, оскільки вона добре підходить для побудови структурованих backend-систем із чітким поділом на контролери, сервіси, репозиторії та моделі даних [7]. Для HabitFlow це має практичне значення: сутності користувача, звички, категорії, днів виконання та журналу відміток зручно описуються як окремі класи з визначеними зв’язками. Сувора типізація Java також зменшує ризик помилок під час роботи з DTO-

об’єктами, параметрами REST-запитів і відповідями, які передаються на клієнтську частину.

Для створення backend-частини використовується Spring Boot. Його вибір обґрунтований тим, що застосунок потребує швидкої реалізації REST API, підключення до бази даних, конфігурації безпеки, тестування та розділення логіки на окремі шари [8]. У HabitFlow контролери приймають HTTP-запити від клієнта, сервіси виконують основну бізнес-логіку, а репозиторії забезпечують доступ до даних. Така структура спрощує подальший супровід системи, оскільки кожен модуль відповідає за окрему частину функціональності.

Для захисту доступу до персональних даних використано Spring Security та JWT. HabitFlow працює з даними конкретного користувача, тому кожна операція зі звичками, історією, статистикою та рекомендаціями має виконуватися лише після перевірки авторизації. Spring Security дозволяє організувати фільтрацію запитів і перевірку прав доступу на рівні серверної частини [9]. JWT використовується для передачі інформації про авторизованого користувача між клієнтом і сервером без збереження стану сесії на backend-стороні [10]. Для цього застосунку такий підхід зручний, оскільки frontend може додавати токен до захищених API-запитів, а сервер — визначати поточного користувача під час обробки кожного запиту.

Для збереження даних обрано PostgreSQL. У HabitFlow дані мають чітку реляційну структуру: користувач має набір звичок, звичка належить до категорії, має заплановані дні виконання та пов’язана із записами журналу виконання. PostgreSQL підходить для такої моделі, оскільки підтримує первинні й зовнішні ключі, обмеження цілісності, індексацію та роботу зі структурованими таблицями [11]. Завдяки цьому можна коректно зберігати зв’язки між сутностями User, HabitCategory, Habit, HabitScheduleDay і HabitLog.

Для керування змінами структури бази даних використано Liquibase. Під час розробки структура таблиць може змінюватися: додаються нові поля, уточнюються зв’язки, створюються окремі таблиці для категорій або днів виконання. Liquibase дозволяє описувати такі зміни у вигляді міграцій і застосовувати їх послідовно [12].

Це зменшує ризик розбіжностей між схемою бази даних і кодом застосунку, а також спрощує повторний запуск проєкту в іншому середовищі.

Клієнтську частину HabitFlow реалізовано з використанням React. Цей вибір обумовлений компонентним підходом до побудови інтерфейсу [13]. У застосунку є кілька повторюваних елементів: картки звичок, форми створення і редагування, блоки статистики, календарна історія та картки рекомендацій. React дозволяє винести такі елементи в окремі компоненти й повторно використовувати їх на різних сторінках. Для швидкого запуску та збірки frontend-частини використовується Vite, що спрощує локальну розробку і скорочує час оновлення інтерфейсу під час внесення змін [14].

Для взаємодії frontend-частини з backend використано Axios. Він застосовується для надсилання HTTP-запитів до REST API, передачі JSON-даних і додавання JWT-токена до захищених запитів [15]. React Router використовується для маршрутизації між сторінками входу, реєстрації, головного огляду, керування звичками, історії, статистики та рекомендацій [16]. Для візуалізації статистичних показників застосовано Recharts, оскільки статистика виконання звичок краще сприймається користувачем у вигляді графіків і діаграм, а не лише як числові значення [17].

Для документування REST API використано Swagger/OpenAPI. У HabitFlow backend має кілька груп endpoint-ів: автентифікація, керування звичками, журнал виконання, статистика, рекомендації та експорт CSV. Документування API дозволяє швидше перевіряти доступні запити, їхні параметри, формат відповіді та можливі помилки [18]. Це також спрощує узгодження між серверною і клієнтською частинами під час розробки.

Для запуску компонентів системи використано Docker Compose. HabitFlow складається з кількох частин: backend, frontend і PostgreSQL. Docker Compose дозволяє описати їхній запуск в одному конфігураційному файлі та підняти всі необхідні сервіси однією командою [19]. Такий підхід зменшує залежність від локальних налаштувань комп’ютера розробника та спрощує перевірку працездатності застосунку після перенесення проєкту в інше середовище.

Обраний технологічний стек відповідає задачам HabitFlow. Java та Spring Boot забезпечують реалізацію серверної логіки, Spring Security і JWT — захищений доступ до персональних даних, PostgreSQL і Liquibase — структуроване та контрольоване збереження інформації, React із супутніми бібліотеками — зручний клієнтський інтерфейс, а Docker Compose — відтворюваний запуск усіх компонентів системи. Разом ці засоби дозволяють реалізувати модульний вебзастосунок для створення звичок, фіксації виконання і пропусків, побудови поведінкових метрик, формування статистики та рекомендацій.

1.8 Постановка завдання на розробку програмного забезпечення

На основі аналізу предметної галузі та існуючих аналогів визначено завдання розробити вебзастосунок HabitFlow для моніторингу та корекції індивідуальних поведінкових патернів користувача. Застосунок має допомагати користувачу створювати звички, фіксувати виконання і пропуски, переглядати історію, аналізувати статистику та отримувати рекомендації.

У системі потрібно реалізувати такі основні можливості:

– реєстрацію та авторизацію користувача;

– створення, редагування, призупинення та видалення звичок;

– вибір категорії, днів виконання і часу нагадування;

– відображення звичок, актуальних на поточний день;

– фіксацію виконання та пропуску;

– збереження причини пропуску;

– автоматичне визначення запланованих днів без відміток;

– перегляд календарної історії;

– формування статистики та рекомендацій;

– експорт статистичних даних у форматі CSV.

Окрему увагу потрібно приділити персональності даних. Кожна звичка та кожен запис журналу мають бути пов’язані з конкретним користувачем. Це

потрібно для того, щоб користувач бачив тільки власні звички, історію, статистику та рекомендації.

Для реалізації поставленого завдання передбачається використання клієнт-серверної архітектури. Серверна частина відповідає за бізнес-логіку, безпеку, роботу з базою даних і REST API. Клієнтська частина забезпечує взаємодію користувача із системою через браузер.

У результаті має бути створено вебзастосунок, який не обмежується простим списком звичок. HabitFlow повинен підтримувати повний цикл роботи з регулярною дією: створення, планування, фіксацію результату, аналіз історії та корекцію поведінкового патерну на основі статистики й рекомендацій.

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ HABITFLOW

2.1 Загальна архітектура програмного забезпечення HabitFlow

Програмне забезпечення HabitFlow спроєктовано як клієнт-серверний вебзастосунок. Його основні частини — frontend, backend і база даних. Frontend відповідає за інтерфейс користувача, backend обробляє запити та виконує бізнес-логіку, а PostgreSQL зберігає користувачів, звички й журнал виконання. Поділ системи на окремі модулі спрощує супровід, тестування та подальше розширення функціональності, що відповідає загальним принципам побудови модульних програмних систем [28; 31].

Користувач працює із застосунком через браузер. Після входу він бачить головну сторінку з активними звичками на поточний день. Звідти можна швидко натиснути «Виконано» або «Пропуск», перейти до списку всіх звичок, переглянути історію, рекомендації або статистику. Усі ці дії передаються на backend через REST API.

Серверна частина побудована навколо кількох основних модулів. Модуль автентифікації відповідає за реєстрацію, вхід і видачу JWT-токена. Модуль звичок обробляє створення, редагування, призупинення та видалення звичок. Журнал виконання зберігає відмітки про виконання або пропуски. Окремо працюють модулі статистики, рекомендацій, автоматичного визначення пропусків і експорту даних у CSV.

Основна логіка застосунку зосереджена на backend-рівні. Наприклад, frontend тільки надсилає запит про виконання звички, а backend перевіряє користувача, знаходить потрібну звичку, створює запис у журналі та повертає оновлений результат. Завдяки цьому важливі правила не дублюються в інтерфейсі й краще контролюються на сервері.

База даних зберігає інформацію у зв’язаній структурі. Користувач має власні звички, а кожна звичка має записи журналу. Така модель потрібна для історії,

статистики й рекомендацій. Якщо користувач відкриває сторінку статистики, backend бере дані саме з журналу виконання, рахує виконані та пропущені дні, визначає серію й формує відповідь для frontend.

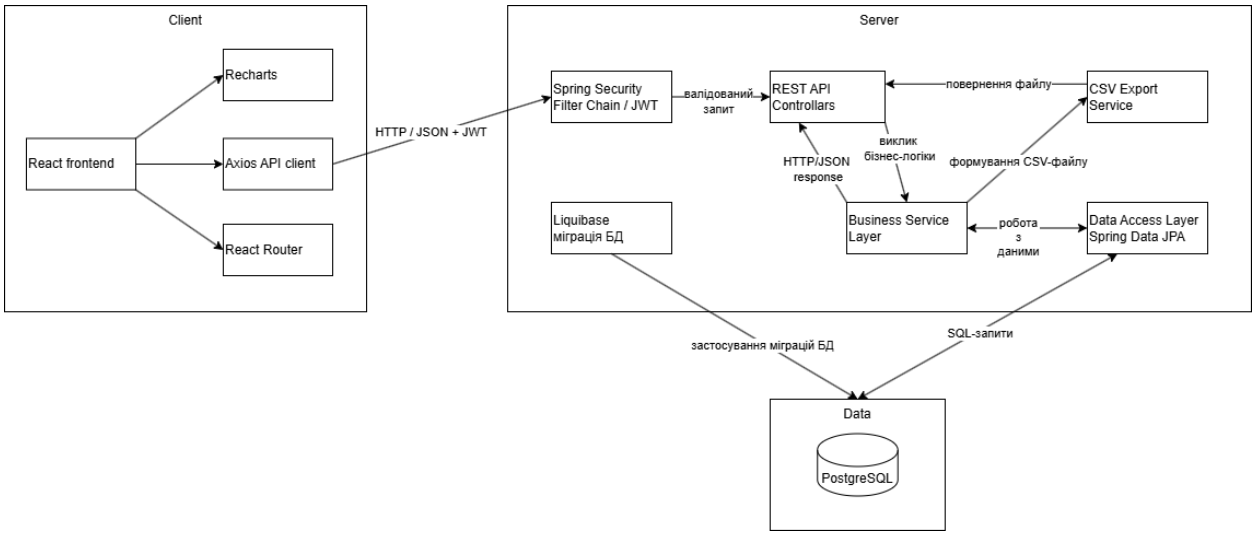


Рис. 2.1 Архітектура застосунку HabitFlow

У загальному вигляді архітектура HabitFlow складається з таких частин:

– клієнтська частина на React;

– REST API на Spring Boot;

– модуль автентифікації та JWT-авторизації;

– сервіс роботи зі звичками;

– сервіс журналу виконання;

– сервіс статистики;

– сервіс рекомендацій;

– модуль експорту CSV;

– база даних PostgreSQL.

Такий поділ робить систему зрозумілою для подальшої підтримки. Якщо потрібно змінити вигляд сторінки статистики, це можна зробити у frontend-частині. Якщо потрібно змінити алгоритм рекомендацій, достатньо оновити відповідний backend-сервіс. База даних при цьому залишається окремим рівнем, який відповідає за збереження даних.

Архітектура HabitFlow побудована так, щоб розділити інтерфейс, бізнес-логіку та збереження даних. Це дозволяє підтримувати застосунок у структурованому вигляді, розширювати його функціональність і не змішувати різні частини системи між собою. Клієнт-серверний підхід відповідає загальним принципам побудови вебсистем, у яких клієнтська частина відповідає за інтерфейс, а серверна — за обробку запитів, бізнес-логіку та роботу з даними [20].

2.2 Визначення функціональних вимог до системи

Функціональні вимоги описують, які дії має виконувати вебзастосунок HabitFlow. У цьому проєкті вони пов’язані з основним сценарієм користувача: зареєструватися, увійти в систему, створити звичку, відмітити виконання або пропуск, переглянути історію, статистику та рекомендації.

Під час визначення вимог враховувалося, що застосунок має бути простим для щоденного використання. Користувач не повинен проходити багато зайвих кроків, щоб поставити одну відмітку. Основна дія має виконуватися швидко: відкрити головну сторінку, знайти звичку на сьогодні й натиснути «Виконано» або «Пропуск».

Функціональні вимоги:

1. Реєстрація та автентифікація користувача через JWT-токени.
2. Створення, редагування, призупинення та видалення звичок.
3. Налаштування категорії, графіка виконання та часу нагадування.
4. Фіксація виконання або пропуску звички з можливістю збереження причини пропуску.
5. Автоматичне формування записів про пропущені дні.
6. Перегляд історії виконання звичок за календарними датами.
7. Формування статистики: відсоток виконання, кількість пропусків, поточна серія.
8. Генерація рекомендацій на основі журналу виконання.

9. Експорт статистичних даних у формат CSV.

Окремо слід зазначити вимогу щодо персональності даних. Усі звички та записи журналу мають належати конкретному користувачу. Якщо один користувач створює звичку «Навчання Java», інший користувач не повинен бачити її у своєму списку або статистиці. Ця вимога реалізується через авторизацію та зв’язок даних із обліковим записом.

Функціональні вимоги HabitFlow охоплюють повний цикл роботи зі звичкою: створення, планування, щоденну відмітку, обробку пропусків, перегляд історії, аналіз статистики та отримання рекомендацій. Саме ці вимоги використовуються далі під час проєктування структури системи, бази даних і REST API.

2.3 Визначення нефункціональних вимог до системи

Нефункціональні вимоги описують не окремі дії користувача, а якість роботи застосунку. Для HabitFlow важливо, щоб система була зручною, стабільною, безпечною і не вимагала складних дій під час щоденного використання.

Оскільки застосунок працює зі звичками та персональною історією користувача, окрему увагу потрібно приділити захисту даних. Користувач має бачити тільки власні звички, записи журналу, статистику та рекомендації. Для цього в системі використовується авторизація через JWT, а паролі зберігаються у хешованому вигляді.

Інтерфейс також має бути простим. Основний сценарій не повинен займати багато часу: користувач відкриває головну сторінку, бачить звички на сьогодні й натискає «Виконано» або «Пропуск». Якщо для цієї дії потрібно багато кроків, застосунок стає незручним для щоденного використання.

Нефункціональні вимоги:

1. Захист доступу до API забезпечується Spring Security та JWT-авторизацією.
2. Паролі користувачів зберігаються у вигляді BCrypt-хешів.

3. REST API підтримує обмін даними у форматі JSON через HTTP.
4. Час відповіді основних API-запитів не перевищує 1–2 секунд при стандартному навантаженні.
5. Дані користувачів ізольовані через прив’язку записів до user_id.
6. Система підтримує контейнеризований запуск через Docker Compose.
7. База даних PostgreSQL забезпечує збереження історії виконання та цілісність зв’язків між сутностями.
8. Frontend адаптований для роботи у сучасних веббраузерах.
9. Архітектура застосунку підтримує подальше розширення функціональності без зміни основних модулів.

Нефункціональні вимоги впливають на те, наскільки зручно буде користуватися HabitFlow у реальних умовах. Наприклад, якщо застосунок має хорошу статистику, але незручний інтерфейс, користувач може перестати регулярно вносити дані. Якщо є історія, але немає авторизації, виникає проблема з персональністю інформації.

Для HabitFlow найбільш важливими є безпека, простота щоденної взаємодії та стабільне збереження історії. Саме ці вимоги враховуються під час проєктування архітектури, бази даних, REST API та frontend-частини.

Нефункціональні вимоги доповнюють функціональні й задають якісні характеристики системи. Вони потрібні для того, щоб HabitFlow був не просто набором функцій, а зручним і безпечним вебзастосунком для щоденного моніторингу звичок.

2.4 Діаграма варіантів використання

Діаграма варіантів використання потрібна для того, щоб показати основні сценарії роботи користувача із системою. У HabitFlow користувач є головним учасником процесу, оскільки саме він створює звички, задає графік, відмічає виконання або пропуски та переглядає результати. Для опису взаємодії

користувача із системою використано UML-діаграму варіантів використання, оскільки вона дозволяє показати основні сценарії роботи з програмним забезпеченням і межі системи [21].

Перший сценарій пов’язаний з обліковим записом. Користувач має зареєструватися або увійти до системи. Після входу frontend отримує JWT-токен, який використовується для подальших запитів до захищених ресурсів. Без авторизації користувач не повинен мати доступу до звичок, історії, статистики та рекомендацій.

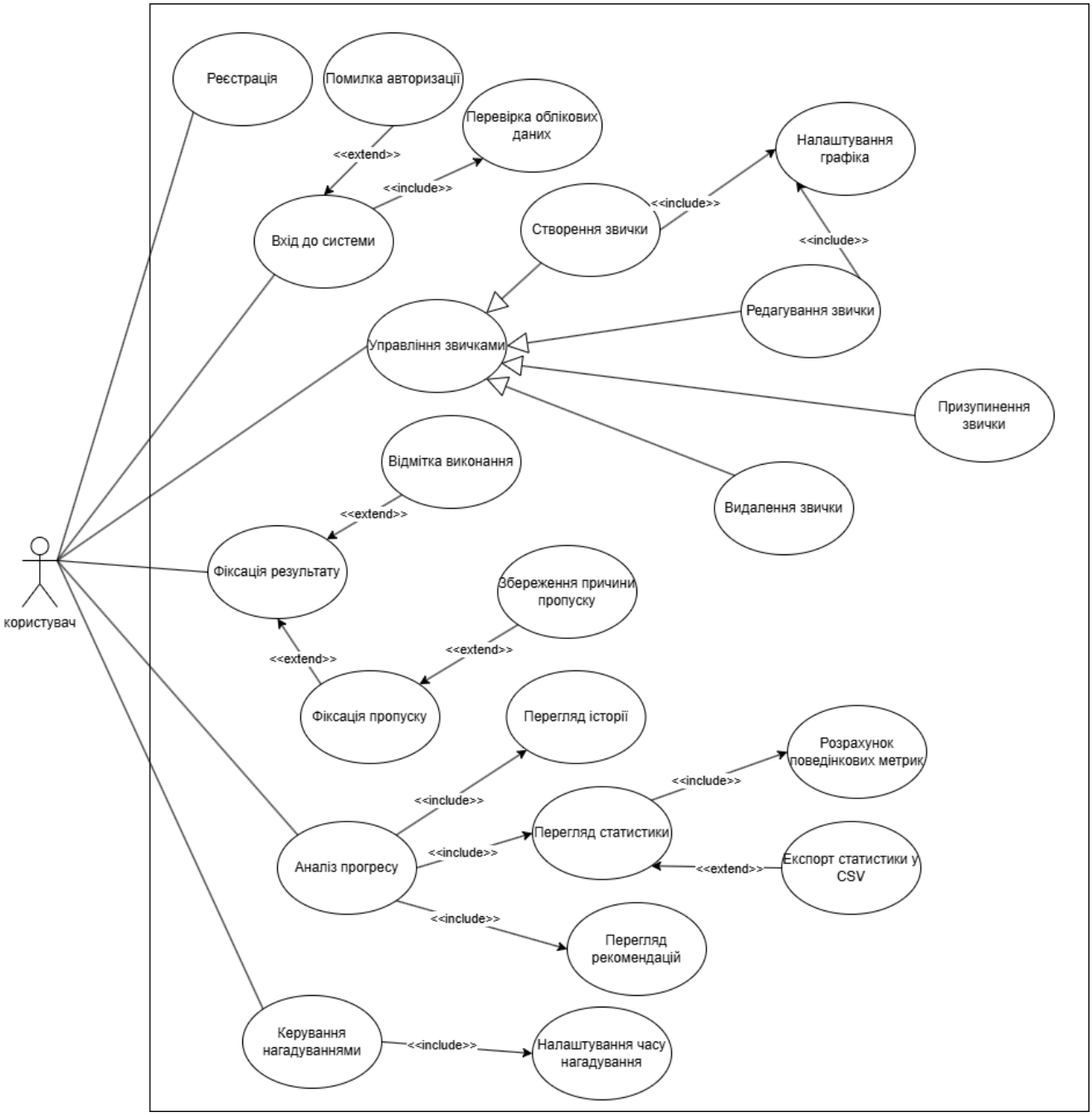


Рис. 2.2 Діаграма варіантів використання вебзастосунку HabitFlow

Другий сценарій стосується керування звичками. Користувач може створити нову звичку, вказати її назву, категорію, опис, дні виконання, цільову кількість днів на тиждень і час нагадування. Якщо звичка більше не підходить, її можна відредагувати, призупинити або видалити. Це потрібно, тому що графік користувача може змінюватися.

Третій сценарій пов’язаний із щоденною роботою. На головній сторінці користувач бачить звички, заплановані на поточний день. Для кожної з них можна натиснути «Виконано» або «Пропуск». Якщо обрано пропуск, користувач може додати причину. Після цього запис зберігається в журналі виконання.

Окремий сценарій — перегляд результатів. Користувач може відкрити історію, статистику або рекомендації. Історія показує виконання у календарному вигляді. Статистика відображає кількість виконаних і пропущених днів, відсоток виконання та серію. Рекомендації допомагають зрозуміти, що можна змінити у графіку або частоті звички.

Також у системі передбачено експорт статистики у форматі CSV. Цей сценарій потрібен для того, щоб користувач міг зберегти дані поза застосунком або переглянути їх у таблиці.

Діаграма варіантів використання показує основні можливості HabitFlow з боку користувача. Вона відображає не внутрішню структуру коду, а саме те, як користувач взаємодіє із застосунком: входить у систему, керує звичками, фіксує результати та переглядає аналітику.

2.5 Сценарії взаємодії користувача із системою

Сценарії взаємодії описують, як користувач працює із вебзастосунком HabitFlow у типових ситуаціях. Вони допомагають зрозуміти не тільки перелік функцій, а й послідовність дій: що користувач робить першим, як система реагує і який результат має бути отриманий.

Основний сценарій починається з входу до системи. Користувач проходить авторизацію, після чого отримує доступ до сторінок застосунку. Далі він може створити звичку, задати її параметри й почати щоденно відмічати результат. Усі записи зберігаються в журналі виконання, а потім використовуються для історії, статистики та рекомендацій.

Таблиця 2.1

Основні сценарії взаємодії користувача із HabitFlow

№	Сценарій	Дії користувача	Результат
1	Реєстрація	Користувач вводить ім’я, email і пароль	Створюється новий обліковий запис
2	Вхід до системи	Користувач вводить email і пароль	Система видає JWT-токен і відкриває головну сторінку
3	Створення звички	Користувач заповнює форму звички: назву, опис, категорію, дні виконання і час нагадування	Нова звичка зберігається в базі даних
4	Редагування звички	Користувач змінює параметри вже створеної звички	Система оновлює дані звички
5	Призупинення звички	Користувач натискає кнопку паузи	Звичка перестає відображатися серед активних
6	Відмітка виконання	Користувач натискає «Виконано»	У журналі створюється запис про виконання
7	Фіксація пропуску	Користувач натискає «Пропуск» і може вказати причину	У журналі створюється запис про пропуск
8	Перегляд історії	Користувач обирає звичку і місяць	Система показує календар виконання
9	Перегляд статистики	Користувач відкриває сторінку статистики	Відображаються виконані дні, пропуски, відсоток виконання і серія
10	Перегляд рекомендацій	Користувач відкриває сторінку рекомендацій	Система показує короткі поради на основі виконання і пропусків
11	Експорт CSV	Користувач натискає кнопку експорту	Завантажується файл зі статистичними даними

Найчастіше користувач взаємодіє із головною сторінкою. На ній відображаються звички, актуальні саме на поточний день. Це скорочує кількість зайвих дій: користувачу не потрібно самостійно шукати, що він має виконати сьогодні. Він одразу бачить потрібні картки та може швидко зафіксувати результат.

Окремий сценарій пов’язаний із пропуском. Якщо користувач не виконав заплановану дію, він може вручну позначити її як пропущену. За потреби додається

причина. Це важливо для подальшого аналізу, бо сам факт пропуску не завжди пояснює проблему. Наприклад, причина може бути пов’язана з браком часу, втомою або невдалим графіком.

Також у HabitFlow передбачено автоматичну обробку днів без відміток. Якщо звичка була запланована, але користувач не натиснув ні «Виконано», ні «Пропуск», система має врахувати такий день як пропущений. Завдяки цьому статистика не ігнорує дні, коли користувач не взаємодіяв із застосунком.

Після накопичення записів користувач може перейти до історії, статистики або рекомендацій. Історія показує календар виконання, статистика дає числову оцінку регулярності, а рекомендації допомагають зрозуміти, що можна змінити. Наприклад, якщо звичка часто пропускається, користувачу може бути доцільно зменшити кількість активних днів.

Сценарії взаємодії HabitFlow побудовані навколо повного циклу роботи зі звичкою: створення, планування, фіксація результату, перегляд історії, аналіз і корекція. Такий підхід дозволяє розглядати звичку не як разову задачу, а як поведінковий патерн, який можна спостерігати й поступово змінювати.

2.6 Проєктування структури бази даних

Структура бази даних HabitFlow проєктується відповідно до логіки роботи вебзастосунку: користувач створює обліковий запис, додає звички, налаштовує графік виконання, фіксує результати та переглядає історію. Для збереження цих даних використовується реляційна модель, у якій кожна основна сутність винесена в окрему таблицю. Під час проєктування враховано принципи нормалізації, зокрема уникнення збереження списків значень в одному полі та винесення повторюваних даних у довідкові таблиці [22].

Основною таблицею є User. Вона зберігає дані облікового запису користувача: ідентифікатор, ім’я, email, хешований пароль і дату створення запису. Пароль не зберігається у відкритому вигляді. У базі даних використовується поле

password_hash, у якому зберігається результат хешування пароля. Це потрібно для захисту облікових даних користувача.

Для опису категорій звичок використовується окрема таблиця HabitCategory. Винесення категорій в окрему таблицю дозволяє уникнути дублювання текстових значень у таблиці звичок. Наприклад, без довідкової таблиці одна й та сама категорія могла б бути записана як «Навчання», «навчання» або «Освіта», що ускладнило б фільтрацію та статистичний аналіз. Таблиця категорій містить ідентифікатор і назву категорії.

Сутність Habit зберігає основні дані про звичку.. До неї належать назва, опис, цільова кількість днів на тиждень, час нагадування, статус активності, дата створення та дата оновлення. Кожна звичка пов’язана з конкретним користувачем через поле user_id, а також із категорією через поле category_id. Завдяки цьому система може відокремлювати звички різних користувачів і групувати їх за категоріями.

Графік виконання звички не зберігається одним текстовим полем. Для цього створено окрему таблицю HabitScheduleDay. Вона містить зв’язок із таблицею Habit і номер дня тижня у полі day_of_week. Такий підхід відповідає першій нормальній формі, оскільки в одному полі не зберігається перелік значень. Якщо звичка виконується у понеділок, середу та п’ятницю, у таблиці буде створено три окремі записи для цієї звички.

Історія виконання зберігається в таблиці HabitLog. Один запис цієї таблиці відповідає одній звичці та одній даті. У журналі зберігаються дата виконання, статус, причина пропуску, ознака автоматичного створення запису та дата створення. Поле status може містити значення, які позначають виконання або пропуск звички. Якщо користувач фіксує пропуск вручну, у полі skip_reason може зберігатися причина.

Між таблицями передбачено такі зв’язки: User має багато записів Habit, HabitCategory може використовуватися багатьма звичками, Habit має багато записів HabitScheduleDay і багато записів HabitLog. Така структура дозволяє зберігати

персональні звички користувача, окремо описувати їхній графік і накопичувати історію виконання без дублювання даних.

Окреме збереження журналу виконання важливе для статистики й рекомендацій. Backend може отримати записи за певний період, порахувати кількість виконаних і пропущених днів, визначити поточну серію та сформувати рекомендації для користувача. Якщо звичка була запланована, але користувач не зробив жодної відмітки, система може створити запис у HabitLog з ознакою автоматичного пропуску.

Проєктована структура бази даних підтримує основні сценарії роботи HabitFlow: реєстрацію користувача, створення звичок, налаштування категорій і графіка, фіксацію виконання або пропуску, перегляд історії, формування статистики, рекомендацій і CSV-експорт. Така модель є достатньо гнучкою для подальшого розширення, наприклад додавання нових типів категорій, розширеної аналітики або складніших правил планування.

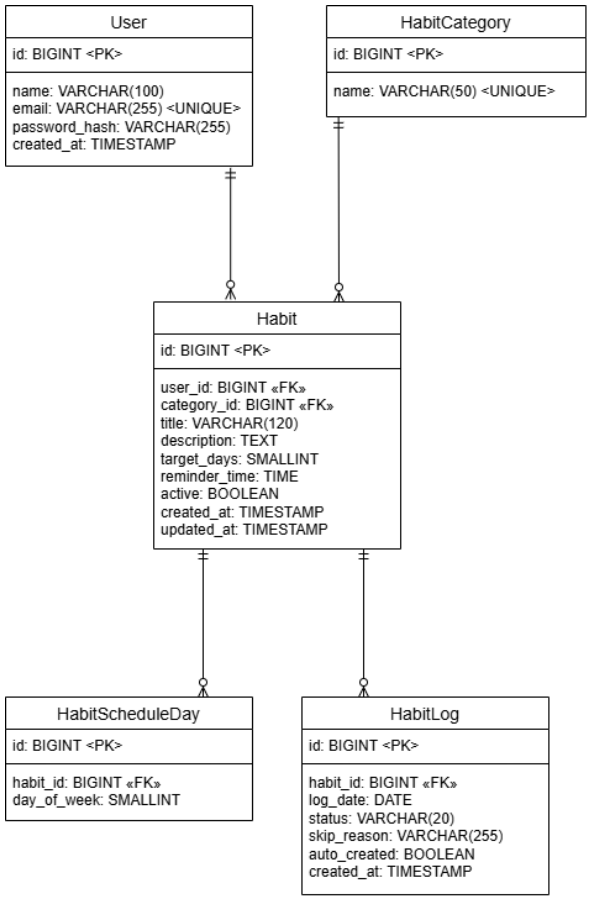


Рис. 2.3 Структура бази даних вебзастосунку HabitFlow

2.7 Опис основних сутностей системи

Основні сутності вебзастосунку HabitFlow визначають структуру даних, з якими працює система під час створення звичок, фіксації виконання, збереження історії та формування статистики. До таких сутностей належать User, HabitCategory, Habit, HabitScheduleDay і HabitLog. Разом вони описують користувача, його звички, категорії, графік виконання та журнал результатів. Виділення основних сутностей системи дозволяє відокремити бізнес-логіку від технічних деталей збереження даних і зробити модель застосунку зрозумілішою для подальшої підтримки [29].

Сутність User відповідає за обліковий запис користувача. Вона зберігає ідентифікатор, ім’я, електронну пошту, хешований пароль і дату створення запису. Email використовується для входу до системи та має бути унікальним. Пароль не зберігається у відкритому вигляді, оскільки для авторизації використовується його хешоване значення.

Сутність HabitCategory використовується для опису категорій звичок. Вона містить ідентифікатор і назву категорії. Окрема таблиця категорій потрібна для того, щоб уникнути дублювання текстових значень у записах звичок. Наприклад, категорії “Навчання”, “Здоров’я” або “Робота” можуть використовуватися багатьма звичками, але зберігатися в одному довіднику.

Сутність Habit описує регулярну дію користувача. Вона містить назву, опис, цільову кількість днів виконання на тиждень, час нагадування, статус активності, дату створення та дату оновлення. Кожна звичка пов’язана з конкретним користувачем через user_id і з категорією через category_id. Завдяки цьому система може відображати лише ті звички, які належать поточному користувачу, а також групувати їх за категоріями.

Сутність HabitScheduleDay відповідає за графік виконання звички. Вона зберігає зв’язок із конкретною звичкою та номер дня тижня. Такий підхід дозволяє не зберігати перелік днів в одному полі таблиці Habit. Якщо звичка має

виконуватися у понеділок, середу та п’ятницю, у таблиці HabitScheduleDay створюються окремі записи для кожного з цих днів.

Сутність HabitLog відповідає за журнал виконання звичок. Вона зберігає не саму звичку, а конкретний результат за певну дату. Один запис HabitLog пов’язаний з однією звичкою та містить дату відмітки, статус виконання або пропуску, причину пропуску, ознаку автоматичного створення запису та дату створення. Якщо користувач вручну фіксує пропуск, система може зберегти причину. Якщо користувач не зробив жодної відмітки у запланований день, система може автоматично створити запис пропуску.

Журнал виконання є основою аналітичної частини HabitFlow. Саме записи HabitLog використовуються для побудови календарної історії, розрахунку відсотка виконання, визначення кількості пропущених днів, поточної серії та формування рекомендацій. Без окремої сутності журналу система могла б показувати лише поточний стан звички, але не її динаміку в часі.

Сутності User, HabitCategory, Habit, HabitScheduleDay і HabitLog утворюють основу моделі даних HabitFlow. Така структура дозволяє зберігати персональні звички користувача, налаштовувати їхній графік, фіксувати виконання або пропуски, аналізувати регулярність і формувати рекомендації на основі накопиченої історії.

2.8 Проєктування REST API вебзастосунку

REST API у HabitFlow використовується для взаємодії між клієнтською та серверною частинами. Frontend надсилає HTTP-запити, а backend обробляє їх, звертається до бази даних і повертає результат у форматі JSON. Під час проєктування REST API враховано підхід до побудови ресурсно-орієнтованих вебінтерфейсів, за яким основні операції системи подаються через ресурси та стандартні HTTP-методи [27]. Такий підхід підходить для вебзастосунку, оскільки інтерфейс і бізнес-логіка залишаються розділеними. REST-підхід використано для організації взаємодії між frontend-частиною та backend-частиною, оскільки він

передбачає роботу з ресурсами через стандартні HTTP-методи та уніфіковані URL-адреси [23].

API поділено на кілька груп. Перша група відповідає за автентифікацію: реєстрацію та вхід користувача. Друга група пов’язана зі звичками: створення, редагування, видалення, призупинення та отримання списку. Окремі endpoint-и використовуються для фіксації виконання, пропуску, перегляду історії, статистики, рекомендацій і експорту CSV.

Публічними залишаються тільки запити реєстрації та входу. Вони потрібні для створення облікового запису й отримання токена. Інші endpoint-и мають бути захищеними, оскільки працюють із персональними даними користувача. Під час таких запитів frontend передає JWT-токен, а backend визначає поточного користувача.

Для фіксації виконання або пропуску використовуються окремі запити. Це зручно, бо така дія має власну логіку. Якщо користувач натискає «Виконано», у журналі створюється запис зі статусом виконання. Якщо він натискає «Пропуск», у записі може додатково зберігатися причина. Надалі ці дані використовуються для історії, статистики та рекомендацій.

Endpoint-и історії та статистики не просто повертають записи з бази даних. Backend попередньо обробляє інформацію: групує записи за датами, рахує кількість виконаних і пропущених днів, визначає відсоток виконання та поточну серію. Frontend отримує вже підготовлені дані й показує їх користувачу у вигляді календаря, карток або графіків.

Таблиця 2.2

Основні REST API endpoint-и вебзастосунку HabitFlow

Метод	**Endpoint**	**Призначення**
POST	`/api/auth/register`	Реєстрація нового користувача
POST	`/api/auth/login`	Вхід до системи та отримання JWT-токена
GET	`/api/habits`	Отримання списку звичок користувача
GET	`/api/habits/today`	Отримання звичок, запланованих на поточний день

POST	/api/habits	Створення нової звички
PUT	/api/habits/{id}	Редагування параметрів звички
PATCH	/api/habits/{id}/active	Зміна статусу активності звички
DELETE	/api/habits/{id}	Видалення звички
POST	/api/habits/{id}/complete	Фіксація виконання звички
POST	/api/habits/{id}/skip	Фіксація пропуску звички
GET	/api/habits/history	Отримання календарної історії виконання
GET	/api/habits/stats	Отримання статистики виконання
GET	/api/habits/recommendations	Отримання рекомендацій для користувача
GET	/api/habits/export	Експорт статистики у форматі CSV

Продовдження Таблиця 2.2

2.9 Алгоритм створення та редагування звички

Створення звички є одним із головних сценаріїв у HabitFlow. Саме з цього починається подальший моніторинг: система не може формувати історію, статистику або рекомендації, якщо користувач спочатку не задав параметри регулярної дії.

Алгоритм створення звички починається з відкриття форми. Користувач вводить назву, опис, обирає категорію, задає кількість днів на тиждень, конкретні дні виконання та час нагадування. Після натискання кнопки створення frontend передає ці дані на backend через REST API.

На серверній стороні виконується перевірка користувача. Backend отримує JWT-токен, визначає поточний обліковий запис і прив’язує нову звичку саме до нього. Це потрібно для того, щоб звички різних користувачів не змішувалися між собою.

Після цього перевіряються дані форми. Назва звички не повинна бути порожньою, кількість днів має відповідати вибраному графіку, а час нагадування повинен зберігатися у коректному форматі. Якщо дані введено неправильно, система повертає повідомлення про помилку. Якщо все коректно, створюється новий запис у таблиці Habit.

Загальний алгоритм створення звички можна подати так:

1. Користувач відкриває сторінку керування звичками.
2. Натискає кнопку створення нової звички.
3. Заповнює назву, опис, категорію, дні виконання та час нагадування.
4. Frontend надсилає дані на backend.
5. Backend перевіряє авторизацію користувача.
6. Система перевіряє коректність введених даних.
7. Якщо дані правильні, створюється запис у базі даних.
8. Frontend оновлює список звичок.

Редагування звички працює за схожою логікою. Користувач відкриває вже створену звичку, змінює потрібні параметри й зберігає результат. Наприклад, він може змінити дні виконання, категорію, опис або час нагадування. Це важливо, бо графік користувача не завжди залишається однаковим.

Під час редагування backend додатково перевіряє, чи належить звичка поточному користувачу. Якщо користувач намагається змінити чужу звичку, система не повинна дозволити таку дію. Якщо звичка належить поточному користувачу, її параметри оновлюються в базі даних.

Окремо враховується статус активності звички. Якщо користувач більше не хоче бачити певну дію серед активних, він може поставити її на паузу. У такому випадку звичка не видаляється повністю, а лише перестає відображатися в актуальному списку. Це зручно, якщо користувач планує повернутися до неї пізніше.

Видалення звички використовується тоді, коли дія більше не потрібна. У цьому випадку backend також перевіряє власника запису, після чого видаляє звичку або робить її недоступною для подальшого використання. Така перевірка потрібна для безпеки й коректної роботи з персональними даними.

Алгоритм створення та редагування звички в HabitFlow побудований навколо кількох основних дій: введення параметрів, перевірка авторизації, валідація даних,

збереження або оновлення запису в базі даних. Такий підхід дозволяє користувачу гнучко керувати звичками й змінювати їх відповідно до власного графіка.

2.10 Алгоритм фіксації виконання та пропуску звички

Фіксація виконання та пропуску є основною щоденною дією користувача в HabitFlow. Після створення звички система повинна не тільки показувати її в потрібний день, а й зберігати результат виконання. Саме ці записи потім використовуються для історії, статистики, серій і рекомендацій.

На головній сторінці користувач бачить звички, які актуальні на поточний день. Для кожної картки доступні дві основні дії: «Виконано» і «Пропуск». Якщо користувач виконав заплановану дію, він натискає «Виконано». Після цього frontend надсилає запит до backend-частини, а сервер створює запис у журналі виконання.

Алгоритм фіксації виконання можна подати так:

1. Користувач відкриває головну сторінку.
2. Система показує звички, заплановані на сьогодні.
3. Користувач натискає кнопку «Виконано».
4. Frontend надсилає запит на backend.
5. Backend перевіряє JWT-токен і визначає поточного користувача.
6. У таблиці HabitLog створюється запис зі статусом виконання.
7. Frontend оновлює дані на сторінці.

Фіксація пропуску працює схожим чином, але має додатковий крок. Якщо користувач натискає «Пропуск», система може запропонувати вказати причину. Це потрібно не для формальності, а для подальшого аналізу. Наприклад, якщо користувач часто обирає причину, пов’язану з нестачею часу, це може означати, що графік звички потрібно змінити.

Алгоритм фіксації пропуску має такий вигляд:

1. Користувач натискає кнопку «Пропуск».
2. Система відкриває форму або модальне вікно для введення причини.

3. Користувач вказує причину пропуску або залишає поле порожнім.
4. Frontend надсилає запит на backend.
5. Backend перевіряє користувача і належність звички.
6. У HabitLog створюється запис зі статусом пропуску.
7. Якщо причина вказана, вона також зберігається в журналі.
8. Сторінка оновлюється, а звичка отримує статус пропущеної.

Окремо потрібно врахувати повторну відмітку за один і той самий день. Якщо користувач уже позначив звичку як виконану або пропущену, система не повинна створювати дубльований запис без потреби. Для цього backend має перевіряти, чи існує запис HabitLog для цієї звички та поточної дати. Якщо запис уже є, його можна оновити або не дозволити повторне створення залежно від реалізованої логіки.

Записи виконання і пропусків зберігаються окремо від самої звички. Це правильний підхід, бо звичка описує правило, а HabitLog описує фактичний результат за конкретний день. Наприклад, звичка «Навчання Java» може бути запланована на понеділок, середу і п’ятницю, але кожен день виконання або пропуску має зберігатися окремим записом.

Також фіксація результату впливає на статистику. Після створення запису в журналі система може перерахувати кількість виконаних днів, кількість пропусків, відсоток виконання і поточну серію. Frontend після оновлення даних показує користувачу вже актуальний стан.

2.11 Алгоритм автоматичного визначення пропущених днів

Автоматичне визначення пропущених днів у HabitFlow потрібне для того, щоб історія виконання не була занадто “оптимістичною”. У реальному використанні користувач не завжди відкриває застосунок щодня. Він може забути поставити відмітку, бути зайнятим або просто не зайти в систему у запланований

день. Якщо такі дні не враховувати, статистика покаже кращий результат, ніж був насправді.

У застосунку ця логіка пов’язана з графіком звички. Якщо звичка активна і запланована, наприклад, на понеділок, середу та п’ятницю, то для кожного з цих днів має бути результат: виконання або пропуск. Коли користувач сам натискає «Виконано» чи «Пропуск», запис створюється одразу. Проблема виникає тоді, коли день минув, а в журналі HabitLog немає жодної відмітки.

У такій ситуації backend перевіряє, чи була звичка запланована на відповідну дату. Якщо так, він шукає запис у журналі виконання. Якщо запис уже існує, додатково нічого створювати не потрібно. Якщо запису немає, система додає пропуск автоматично. Для такого запису окремо зберігається ознака автоматичного створення, щоб його можна було відрізнити від ручного пропуску.

Загальний алгоритм можна подати так:

1. Отримати активні звички користувача.
2. Перевірити графік виконання для кожної звички.
3. Визначити, чи була звичка запланована на конкретну дату.
4. Перевірити, чи є запис у HabitLog за цей день.
5. Якщо запис існує, залишити його без змін.
6. Якщо запису немає, створити запис зі статусом пропуску.
7. Позначити цей пропуск як автоматично створений.

Ручний і автоматичний пропуск мають різний зміст. Ручний пропуск означає, що користувач сам відкрив застосунок і визнав, що дія не була виконана. У такому випадку він може ще й указати причину. Автоматичний пропуск показує іншу ситуацію: звичка була в плані, але користувач не залишив жодної відмітки.

Цей механізм також впливає на рекомендації. Якщо в історії накопичується багато автоматичних пропусків, це може означати, що користувач рідко відкриває застосунок або вибрав занадто щільний графік. У такому випадку рекомендація може підказати спростити розклад, зменшити кількість активних днів або почати з простішої версії звички.

Автоматичне визначення пропущених днів робить історію HabitFlow повнішою. Воно не замінює ручну відмітку користувача, але допомагає не втрачати ті дні, які були заплановані й залишилися без результату.

2.12 Алгоритм формування статистики виконання

Статистика в HabitFlow будується не за одним поточним днем, а за записами з журналу HabitLog. Це важливо, бо одна відмітка не показує реальну регулярність звички. Користувач може виконати дію сьогодні, але часто пропускати її протягом тижня. Або навпаки — мати один пропуск після довгої стабільної серії. Саме тому для статистики береться історія виконання за певний період.

Коли користувач відкриває сторінку статистики, frontend надсилає запит на сервер. Backend визначає поточного користувача за JWT-токеном і отримує тільки його звички. Далі для кожної звички вибираються записи з HabitLog: виконані дні, пропуски, автоматично створені пропуски та записи з причинами. На основі цих даних формується підсумок, який потім відображається на сторінці.

Загальний алгоритм формування статистики можна подати так:

1. Користувач відкриває сторінку статистики.
2. Frontend надсилає запит до backend.
3. Backend визначає користувача за JWT-токеном.
4. З бази даних отримуються звички цього користувача.
5. Для кожної звички вибираються записи HabitLog.
6. Підраховується кількість виконаних і пропущених днів.
7. Обчислюється відсоток виконання.
8. Визначається поточна серія.
9. Підготовлені дані повертаються на frontend.

Відсоток виконання показує, наскільки стабільно користувач дотримується обраного графіка. Якщо у звички багато пропусків, значення буде нижчим. Якщо користувач регулярно натискає «Виконано» у заплановані дні, відсоток поступово

зростає. Такий показник зручний для швидкої оцінки, бо не потрібно вручну переглядати весь календар.

Поточна серія показує кількість послідовних виконань без перерви. Для користувача це має мотиваційне значення. Наприклад, якщо звичка «Навчання Java» виконується кілька запланованих днів поспіль, серія допомагає побачити, що регулярність уже формується. Якщо серія обривається, це не скасовує попередній прогрес, але показує момент, з якого потрібно почати знову.

Статистика також корисна для порівняння звичок між собою. Одна дія може виконуватися майже без пропусків, а інша — мати низький відсоток виконання. У такому випадку користувач одразу бачить, яка звичка потребує уваги. Наприклад, якщо «Водний баланс» має 90%, а «Спорт» — 40%, проблема, швидше за все, не в самому застосунку, а в складності або графіку другої звички.

Алгоритм формування статистики перетворює окремі записи HabitLog на зрозумілі показники: виконані дні, пропуски, відсоток виконання і серію. Завдяки цьому HabitFlow показує не просто набір відміток, а реальну картину регулярності, з якою вже можна працювати далі.

2.13 Алгоритм формування рекомендацій для користувача

Рекомендації в HabitFlow формуються після того, як у системі вже є певна історія виконання звички. Самі по собі числа не завжди пояснюють, що потрібно змінити. Користувач може бачити низький відсоток виконання, але не одразу розуміти причину: графік занадто щільний, час вибраний невдало або звичка поки що складна для регулярного виконання.

Для формування рекомендацій використовується журнал HabitLog. Backend аналізує записи для кожної звички: скільки разів дія була виконана, скільки разів пропущена, чи є поточна серія та який загальний відсоток виконання. Якщо записів ще мало, рекомендація залишається загальною. Коли історії стає більше, порада може бути точнішою.

Загальний алгоритм формування рекомендацій можна подати так:

1. Користувач відкриває сторінку рекомендацій.
2. Frontend надсилає запит до backend.
3. Backend визначає поточного користувача.
4. Із бази даних отримуються звички та записи HabitLog.
5. Для кожної звички рахується відсоток виконання, кількість пропусків і серія.
6. На основі цих показників обирається тип рекомендації.
7. Готові рекомендації повертаються на frontend.

Наприклад, якщо звичка часто пропускається, система може порадити почати з меншої кількості днів на тиждень. Для звички «Навчання Java» це може означати не п’ять активних днів, а два або три. Якщо серія перервалася, рекомендація може запропонувати відновити її з найближчого запланованого дня. Якщо ж звичка виконується стабільно, порада має підтримувальний характер.

Окремо враховується кількість пропусків. Часті пропуски не завжди означають відсутність мотивації. Іноді проблема в тому, що дія погано вписана в розклад. Наприклад, користувач поставив навчання на вечір, але саме ввечері часто втомлюється після роботи. У такій ситуації корисніше переглянути час або частоту, а не просто “старатися більше”.

Рекомендаційний модуль не приймає рішення замість користувача. Він лише показує можливий напрям зміни. Користувач сам вирішує, що робити далі: залишити звичку без змін, змінити графік, поставити її на паузу або видалити.

Таким чином, рекомендації доповнюють статистику. Статистика показує фактичний стан звички, а рекомендація допомагає зрозуміти наступний крок. Завдяки цьому HabitFlow працює не тільки як журнал виконання, а і як інструмент для поступової корекції поведінкового патерну.

2.14 Проєктування інтерфейсу користувача

Інтерфейс HabitFlow проєктувався з урахуванням того, що користувач буде відкривати застосунок регулярно. Для таких систем важливо не перевантажувати сторінки зайвими діями. Якщо для звичайної відмітки потрібно довго шукати потрібну кнопку або переходити між кількома розділами, користувач швидко перестане вносити дані.

Основною сторінкою є огляд поточного дня. На ній показується загальний прогрес, кількість активних звичок, виконані та пропущені дії. Нижче розміщуються картки звичок, які заплановані саме на сьогодні. У картці видно назву, категорію, короткий опис, графік, час нагадування та кнопки «Виконано» і «Пропуск». Такий формат дозволяє швидко зафіксувати результат без зайвих переходів.

Для керування звичками передбачена окрема сторінка. На ній користувач може створити нову звичку, змінити вже існуючу, поставити її на паузу або видалити. Форма створення містить тільки ті поля, які справді потрібні для роботи застосунку: назву, опис, категорію, дні виконання, цільову кількість днів і час нагадування. Це робить налаштування зрозумілим навіть без додаткових пояснень.

Історія виконання подається у календарному вигляді. Такий варіант зручний, бо користувач одразу бачить, у які дні звичка виконувалася, коли була пропущена, а де запису не було. Для поведінкових патернів це важливіше, ніж простий список дат, оскільки календар краще показує повторюваність.

Сторінка статистики використовується для швидкої оцінки результатів. На ній відображаються відсоток виконання, кількість виконаних днів, пропуски та поточна серія. Додатково застосовуються графіки, щоб користувач міг побачити динаміку за тиждень або інший період. Це допомагає швидше зрозуміти, яка звичка виконується стабільно, а яка потребує перегляду.

Сторінка рекомендацій подає поради у вигляді окремих карток. У них зазначається звичка, коротке пояснення проблеми та можливий напрям зміни. Наприклад, якщо дія часто пропускається, рекомендація може запропонувати

зменшити кількість активних днів або змінити час виконання. Такий формат не змушує користувача одразу змінювати звичку, але дає зрозумілу підказку.

Окремо реалізовано сторінки входу та реєстрації. Вони мають просту структуру: поля для email і пароля, кнопка входу або створення облікового запису. Після успішної авторизації користувач переходить до основної частини HabitFlow і працює вже зі своїми персональними даними.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ HABITFLOW

3.1 Реалізація серверної частини застосунку

Серверну частину реалізовано з використанням Java та Spring Boot, що дозволило побудувати REST API, сервісний шар, репозиторії та конфігурацію застосунку [7; 8]. Саме backend створює звички, зберігає відмітки виконання, обробляє пропуски, формує статистику та рекомендації.

Код серверної частини поділено на кілька логічних рівнів. Контролери приймають HTTP-запити від frontend. Сервіси виконують основну бізнес-логіку. Репозиторії працюють із базою даних через Spring Data JPA. Окремо використовуються DTO-класи, які передають дані між клієнтом і сервером без прямої роботи з внутрішніми сутностями бази даних.

Backend-частина проєкту розміщена в каталозі src/main/java/com/example/habitflow і поділена на кілька пакетів. Пакет controller містить REST-контролери AuthController, HabitController і UserController. Пакет service відповідає за бізнес-логіку та містить AuthService, HabitService, AutoSkipService, RecommendationService і HabitExportService. Пакет repository використовується для роботи з базою даних через Spring Data JPA. У пакеті entity описано сутності User, Habit і HabitLog, а в пакеті security розміщено класи JwtAuthenticationFilter, JwtTokenProvider і UserDetailsServiceImpl.

Основними модулями backend-частини є модуль авторизації, модуль звичок, модуль журналу виконання, модуль статистики, модуль рекомендацій та експорт даних у CSV. Наприклад, коли користувач натискає кнопку «Виконано», frontend надсилає запит на сервер. Backend перевіряє користувача за JWT-токеном, знаходить потрібну звичку, створює запис у HabitLog і повертає оновлені дані.

Окрема увага приділена перевірці належності даних користувачу. Якщо користувач відкриває список звичок або статистику, backend повинен отримати

тільки ті записи, які належать його обліковому запису. Це стосується звичок, журналу виконання, історії та рекомендацій.

Для збереження даних використовується PostgreSQL. Структура бази даних створюється та оновлюється через Liquibase. Завдяки цьому таблиці користувачів, звичок і журналу виконання створюються автоматично під час запуску застосунку, без ручного створення SQL-структури.

Серверна частина також містить логіку автоматичного визначення пропущених днів. Якщо звичка була запланована, але користувач не зробив жодної відмітки, backend може створити запис пропуску. Це потрібно для того, щоб статистика не ігнорувала дні без взаємодії.

У результаті backend-частина HabitFlow виконує роль центрального рівня системи. Вона не тільки приймає дані з інтерфейсу, а й контролює доступ, зберігає історію, обробляє бізнес-правила та готує дані для відображення на frontend.

3.2 Реалізація модуля автентифікації та авторизації користувача

Для захисту доступу до API використано Spring Security та JWT-токени, які дозволяють ідентифікувати користувача під час виконання захищених запитів [9; 10]. Без цього модулю застосунок не міг би коректно розділяти дані різних користувачів. Звички, історія виконання, статистика і рекомендації мають належати тільки тому обліковому запису, який їх створив.

Під час реєстрації користувач вводить ім’я, email і пароль. Backend приймає ці дані, перевіряє їх і створює новий запис у таблиці користувачів. Пароль не зберігається у відкритому вигляді. Перед збереженням він хешується за допомогою BCrypt. Це важливо, бо навіть у разі доступу до бази даних пароль не можна буде просто прочитати як звичайний текст.

Під час входу користувач вводить email і пароль. Backend знаходить користувача за email, перевіряє пароль і після успішної перевірки формує JWT-токен. Далі frontend використовує цей токен для запитів до захищених endpoint-ів.

Наприклад, коли користувач відкриває сторінку звичок або статистики, запит надсилається разом із токеном.

Перевірка JWT-токена реалізована у класі JwtAuthenticationFilter, який наслідує OncePerRequestFilter. Під час кожного HTTP-запиту фільтр перевіряє наявність заголовка Authorization. Якщо заголовок починається з Bearer, із нього виділяється JWT-токен. Після цього JwtTokenProvider перевіряє валідність токена та отримує email користувача. За email завантажуються дані користувача, після чого в SecurityContextHolder встановлюється об’єкт автентифікації. Завдяки цьому контролери можуть отримувати поточного користувача через @AuthenticationPrincipal.

Авторизація потрібна не тільки для входу в систему. Вона використовується майже в кожному основному сценарії HabitFlow. Якщо користувач створює звичку, backend прив’язує її до поточного облікового запису. Якщо він переглядає історію, система повертає тільки його записи. Те саме стосується статистики, рекомендацій і CSV-експорту.

У Spring Security налаштовано поділ endpoint-ів на відкриті та захищені. Відкритими залишаються реєстрація, вхід і документація API. Інші запити доступні тільки після перевірки JWT-токена. Завдяки цьому сторонній користувач не може отримати список чужих звичок або змінити їх через API.

Модуль автентифікації та авторизації забезпечує персональну роботу з HabitFlow. Користувач отримує доступ до власних даних після входу, а backend контролює всі запити до звичок, журналу виконання, статистики та рекомендацій.

3.3 Реалізація модуля керування звичками

Модуль керування звичками є одним із головних у HabitFlow. Саме через нього користувач створює регулярні дії, змінює їхній графік, ставить звички на паузу або видаляє їх. Без цього модуля система не могла б перейти до журналу виконання, статистики чи рекомендацій, бо спочатку потрібно створити саму звичку.

Під час створення звички користувач заповнює форму. У ній вказується назва, опис, категорія, цільова кількість днів на тиждень, конкретні дні виконання та час нагадування. Після збереження frontend надсилає ці дані на backend. Сервер перевіряє користувача за JWT-токеном і створює запис у таблиці Habit.

Редагування працює схожим чином. Користувач відкриває вже створену звичку, змінює потрібні поля й зберігає оновлення. Наприклад, він може перенести «Навчання Java» з вечора на інший час або змінити дні виконання з п’яти на три дні на тиждень. Це важливо, бо реальний графік користувача може змінюватися.

Для кожної операції backend перевіряє, чи належить звичка поточному користувачу. Це потрібно, щоб один користувач не міг змінити або видалити чужі записи через API. Якщо звичка не належить поточному обліковому запису, запит не повинен виконуватися.

У модулі також реалізовано зміну статусу активності. Якщо користувач не хоче видаляти звичку повністю, він може призупинити її. У такому випадку звичка залишається в базі даних, але не відображається серед активних дій на сьогодні. Це зручно для тимчасових перерв.

Видалення використовується тоді, коли звичка більше не потрібна. Перед видаленням backend також перевіряє власника запису. Така перевірка однакова для редагування, призупинення та видалення, бо всі ці дії змінюють персональні дані користувача.

У результаті модуль керування звичками забезпечує повний набір дій над сутністю Habit: створення, перегляд, редагування, призупинення та видалення. Саме ці дані далі використовуються для відображення звичок на головній сторінці, фіксації виконання, побудови історії та формування статистики.

3.4 Реалізація журналу виконання звичок

Журнал виконання в HabitFlow реалізовано через сутність HabitLog. Вона зберігає не саму звичку, а результат за конкретний день. Це може бути виконання або пропуск. Такий підхід зручний, бо одна звичка повторюється багато разів, і кожна відмітка повинна мати окремий запис.

Коли користувач натискає «Виконано», frontend передає на backend ідентифікатор звички. Сервер перевіряє JWT-токен, визначає поточного користувача і перевіряє, чи належить ця звичка саме йому. Після цього в журналі створюється запис зі статусом виконання.

Фіксація пропуску працює схоже, але має додаткове поле для причини. Якщо користувач натискає «Пропуск», він може вказати, чому дія не була виконана. Наприклад, через втому, нестачу часу або зміну планів. Ця причина зберігається в HabitLog і надалі може враховуватися під час перегляду історії або формування рекомендацій.

Окремо в журналі враховується дата запису. Це потрібно для того, щоб не створювати кілька однакових відміток за один день. Якщо для певної звички вже є запис на поточну дату, backend має або оновити його, або не дозволити дублювання. Завдяки цьому статистика залишається коректною.

Журнал виконання також використовується для автоматичної фіксації пропусків. Якщо звичка була запланована, але користувач не зробив жодної відмітки, система може створити запис пропуску автоматично. Такий запис відрізняється від ручного, бо користувач не вказував причину самостійно.

У результаті HabitLog є основою для історії, статистики та рекомендацій. Саме з цих записів система визначає, які дні були виконані, які пропущені, де перервалася серія і які звички потребують корекції. Без журналу HabitFlow міг би показувати тільки поточний стан, але не динаміку поведінкового патерну.

3.5 Реалізація автоматичної фіксації пропущених днів

У HabitFlow реалізовано автоматичну фіксацію пропущених днів. Ця функція потрібна для ситуацій, коли користувач не відкрив застосунок у день, коли звичка була запланована. Якщо такий день просто не враховувати, статистика буде виглядати кращою, ніж є насправді.

Логіка працює з активними звичками користувача. Backend перевіряє, які звички були заплановані на певну дату, а потім шукає відповідний запис у журналі HabitLog. Якщо користувач уже натиснув «Виконано» або «Пропуск», додатковий запис не створюється. Якщо запису немає, система створює пропуск автоматично.

Такий пропуск відрізняється від ручного. Коли користувач сам натискає «Пропуск», він може вказати причину. Автоматичний пропуск означає інше: день був у графіку, але користувач не зробив жодної відмітки. Для цього в записі журналу зберігається окрема ознака автоматичного створення.

Автоматична фіксація пропусків реалізована у класі AutoSkipService. Метод autoSkipForAllUsers() запускається за розкладом за допомогою анотації @Scheduled і перевіряє активні звички користувачів наприкінці дня. Для кожної звички система визначає, чи запланована вона на поточну дату. Якщо звичка активна, запланована на цей день і в таблиці habit_logs ще немає запису за відповідну дату, створюється новий запис зі статусом SKIPPED. Для ручного запуску передбачено метод autoSkipForUser(), який використовується через endpoint /api/habits/auto-skip.

Ця логіка важлива для історії та статистики. Наприклад, якщо звичка «Навчання Java» запланована на понеділок, середу і п’ятницю, але користувач не відкрив застосунок у середу, цей день не повинен зникати з аналізу. Він має бути врахований як пропущений, інакше відсоток виконання буде неточним.

Автоматична фіксація також впливає на рекомендації. Якщо система бачить багато таких пропусків, це може означати, що користувач рідко відкриває застосунок або вибрав занадто складний графік. У такій ситуації рекомендація

може запропонувати зменшити кількість активних днів або переглянути час виконання.

У результаті автоматична фіксація пропущених днів робить журнал HabitLog повнішим. Вона не замінює ручну відмітку користувача, але допомагає не втрачати заплановані дні, які залишилися без результату.

3.6 Реалізація модуля статистики та рекомендацій

Для візуального відображення статистичних показників на клієнтській частині використано бібліотеку Recharts [17]. Коли користувач відкриває сторінку статистики, backend отримує його звички та пов’язані з ними записи виконання. Після цього система рахує кількість виконаних днів, кількість пропусків, відсоток виконання та поточну серію.

Ці показники потрібні для того, щоб користувач бачив не тільки окремі відмітки, а загальну картину. Наприклад, звичка може бути виконана сьогодні, але мати багато пропусків протягом тижня. У такому випадку статистика показує реальний стан, а не тільки поточний день.

На frontend-частині статистика відображається у вигляді карток і графіків. Для побудови графіків використовується бібліотека Recharts. Вона дозволяє показати тижневу динаміку виконання і пропусків у зрозумілому вигляді. Користувачу не потрібно самостійно переглядати всі записи журналу, щоб побачити, у які дні були проблеми.

Модуль рекомендацій використовує ті самі дані, але подає їх інакше. Якщо статистика показує числа, то рекомендація дає короткий практичний висновок. Наприклад, якщо у звички низький відсоток виконання, застосунок може порадити зменшити кількість активних днів. Якщо серія перервалася, користувачу можна запропонувати почати нову з найближчого запланованого дня.

Рекомендації не змінюють звичку автоматично. Вони лише підказують, на що варто звернути увагу. Остаточне рішення залишається за користувачем: змінити графік, залишити все без змін, поставити звичку на паузу або видалити її.

У результаті модулі статистики та рекомендацій доповнюють один одного. Статистика показує фактичний стан звички, а рекомендації допомагають зрозуміти, що з цим станом можна зробити далі. Саме ця частина відрізняє HabitFlow від простого списку регулярних дій.

3.7 Реалізація CSV-експорту статистичних даних

У HabitFlow реалізовано експорт статистичних даних у форматі CSV. Ця функція потрібна для того, щоб користувач міг зберегти результати поза застосунком. Наприклад, статистику можна відкрити в Excel, Google Sheets або іншому табличному редакторі.

Експорт працює на основі тих самих даних, які використовуються для сторінки статистики. Backend отримує звички користувача, записи з HabitLog, рахує виконані дні, пропуски, відсоток виконання та серію. Після цього ці дані формуються у вигляді CSV-файлу.

Frontend викликає експорт через окремий API-запит. Користувач натискає кнопку експорту, після чого браузер завантажує файл. У файлі можуть міститися назви звичок, кількість виконаних днів, кількість пропусків, відсоток виконання та поточна серія. Такий формат зручний, бо його можна переглядати без доступу до самого застосунку.

Окремо важливо, що експорт працює тільки для авторизованого користувача. Backend визначає користувача за JWT-токеном і включає до файлу тільки його дані. Це потрібно, щоб статистика різних облікових записів не змішувалася.

CSV-експорт не є головною функцією HabitFlow, але він доповнює статистичний модуль. Користувач отримує можливість не тільки переглянути результати в інтерфейсі, а й зберегти їх окремим файлом для подальшого аналізу або звітності.

У результаті CSV-експорт робить роботу зі статистикою більш гнучкою. Дані не залишаються тільки всередині застосунку, а можуть бути використані в інших інструментах.

3.8 Реалізація браузерних нагадувань

У HabitFlow реалізовано можливість задавати час нагадування для звички. Це потрібно для тих дій, які користувач хоче виконувати в конкретний момент дня. Наприклад, навчання Java можна запланувати на 19:00, а водний баланс — на ранок або обідній час.

Час нагадування задається під час створення або редагування звички. Користувач обирає потрібне значення у формі, після чого воно зберігається разом з іншими параметрами звички. На backend-рівні це поле належить до сутності Habit, а на frontend використовується для показу часу в картці звички.

Браузерні нагадування працюють на клієнтській стороні. Frontend перевіряє звички, які мають встановлений час нагадування, і може показати повідомлення у браузері. Перед цим користувач має надати дозвіл на сповіщення. Без такого дозволу браузер не покаже повідомлення, навіть якщо час нагадування заданий.

Ця функція не замінює основну логіку моніторингу. Якщо користувач отримав нагадування, але не виконав дію, результат усе одно має бути зафіксований через «Виконано» або «Пропуск». Тобто нагадування лише допомагає не забути про звичку, а журнал HabitLog зберігає фактичний результат.

Для користувача ця функція корисна тим, що зменшує кількість випадкових пропусків. Якщо звичка запланована на певний час, застосунок може нагадати про неї без необхідності постійно відкривати сторінку. Особливо це зручно для дій, які легко пропустити через навчання, роботу або інші справи.

У результаті браузерні нагадування доповнюють основну логіку HabitFlow. Вони допомагають користувачу вчасно згадати про заплановану дію, але не змінюють саму модель роботи зі звичками: результат усе одно фіксується через журнал виконання.

3.9 Реалізація клієнтської частини застосунку

Клієнтська частина HabitFlow реалізована на React. Вона відповідає за сторінки, форми, картки звичок, графіки та взаємодію користувача із застосунком. Через frontend користувач створює звички, відмічає виконання або пропуск, переглядає історію, статистику та рекомендації. Під час реалізації клієнтської частини враховано принципи побудови інтерактивних вебінтерфейсів, де важливими є зрозуміла структура сторінок, швидка реакція на дії користувача та повторне використання компонентів [30].

Структура клієнтської частини розміщена в каталозі frontend/src. Основний файл App.jsx відповідає за маршрутизацію сторінок. У каталозі api зберігаються модулі для роботи з backend API: client.js, auth.js, habits.js і token.js. Каталог components містить повторно використовувані компоненти інтерфейсу, зокрема HabitCard, HabitForm, Navbar, PrivateRoute, ReminderWatcher і SkipReasonModal. У каталозі pages розміщено сторінки Dashboard, Habits, History, Login, Recommendations, Register і Stats.

Для роботи з backend використовується Axios. Коли користувач відкриває сторінку, frontend надсилає запит до REST API і отримує потрібні дані. Якщо користувач натискає «Виконано», «Пропуск» або створює нову звичку, відповідний запит також передається на сервер.

Навігація між сторінками реалізована через React Router. Це дозволяє переходити між головною сторінкою, звичками, історією, рекомендаціями та статистикою без повного перезавантаження застосунку. Для користувача це виглядає як швидкий і цілісний вебінтерфейс.

Керування станом авторизації реалізовано через AuthContext. Після входу користувача JWT-токен зберігається в localStorage під ключем habitflow_token. AuthContext перевіряє наявність токена, зберігає інформацію про стан авторизації та надає її іншим компонентам через хук useAuth. Завдяки цьому компоненти можуть визначати, чи користувач увійшов у систему, і відповідно показувати або приховувати захищені сторінки.Для статистики використовується бібліотека

Recharts. Вона допомагає показати виконані та пропущені дні у вигляді графіків. Це робить сторінку статистики зрозумілішою, бо користувач бачить не тільки числа, а й динаміку.

3.10 Реалізація маршрутизації та захищених сторінок

У клієнтській частині HabitFlow реалізовано маршрутизацію між основними сторінками застосунку. Для цього використовується React Router. Завдяки маршрутизації користувач може переходити між сторінками входу, реєстрації, головного огляду, звичок, історії, рекомендацій і статистики без повного перезавантаження сторінки.

У застосунку є відкриті та захищені маршрути. До відкритих належать сторінки входу та реєстрації. Вони доступні користувачу без JWT-токена, оскільки саме через них він отримує доступ до системи. Після успішного входу frontend зберігає токен і може відкривати основні сторінки HabitFlow.

Захист маршрутів реалізовано через компонент PrivateRoute. Він отримує стан авторизації з AuthContext. Якщо перевірка авторизації ще триває, компонент показує екран завантаження. Якщо користувач не авторизований, виконується перенаправлення на сторінку /login. Якщо токен є і користувач авторизований, PrivateRoute відображає вкладений компонент сторінки.

Захищені сторінки доступні тільки після авторизації. Це головний огляд, список звичок, історія, рекомендації та статистика. Якщо користувач без токена намагається перейти на одну з цих сторінок, його потрібно перенаправити на сторінку входу. Такий підхід не дозволяє відкривати персональні дані без авторизації.

Окремо маршрутизація пов’язана з навігаційним меню. Після входу користувач бачить посилання на основні розділи застосунку. Наприклад, він може швидко перейти з головної сторінки до історії або статистики. Це робить роботу із застосунком простішою, бо всі основні функції доступні з одного інтерфейсу.

Для підключення JWT до HTTP-запитів використано Axios interceptor у файлі api/client.js. Перед кожним запитом frontend перевіряє наявність токена в localStorage. Якщо токен існує, він додається до заголовка Authorization у форматі Bearer token. У разі відповіді сервера зі статусом 401 frontend генерує подію auth:unauthorized, після чого користувача можна перенаправити на сторінку входу.

Маршрутизація також допомагає розділити код frontend-частини. Кожна сторінка має власний компонент і відповідає за окремий сценарій: огляд дня, керування звичками, перегляд історії або аналіз статистики. Завдяки цьому код легше підтримувати й змінювати.

У результаті реалізована маршрутизація забезпечує зрозумілу структуру клієнтської частини HabitFlow. Користувач отримує швидкі переходи між сторінками, а застосунок розділяє відкриті та захищені частини відповідно до стану авторизації.

3.11 Опис екранних форм вебзастосунку HabitFlow

Побудова клієнтського інтерфейсу виконана за компонентним підходом React, що дозволяє повторно використовувати елементи сторінок і спростити підтримку frontend-частини [13]. Через інтерфейс він проходить реєстрацію, входить у систему, створює звички, відмічає виконання або пропуск, переглядає історію, статистику та рекомендації.

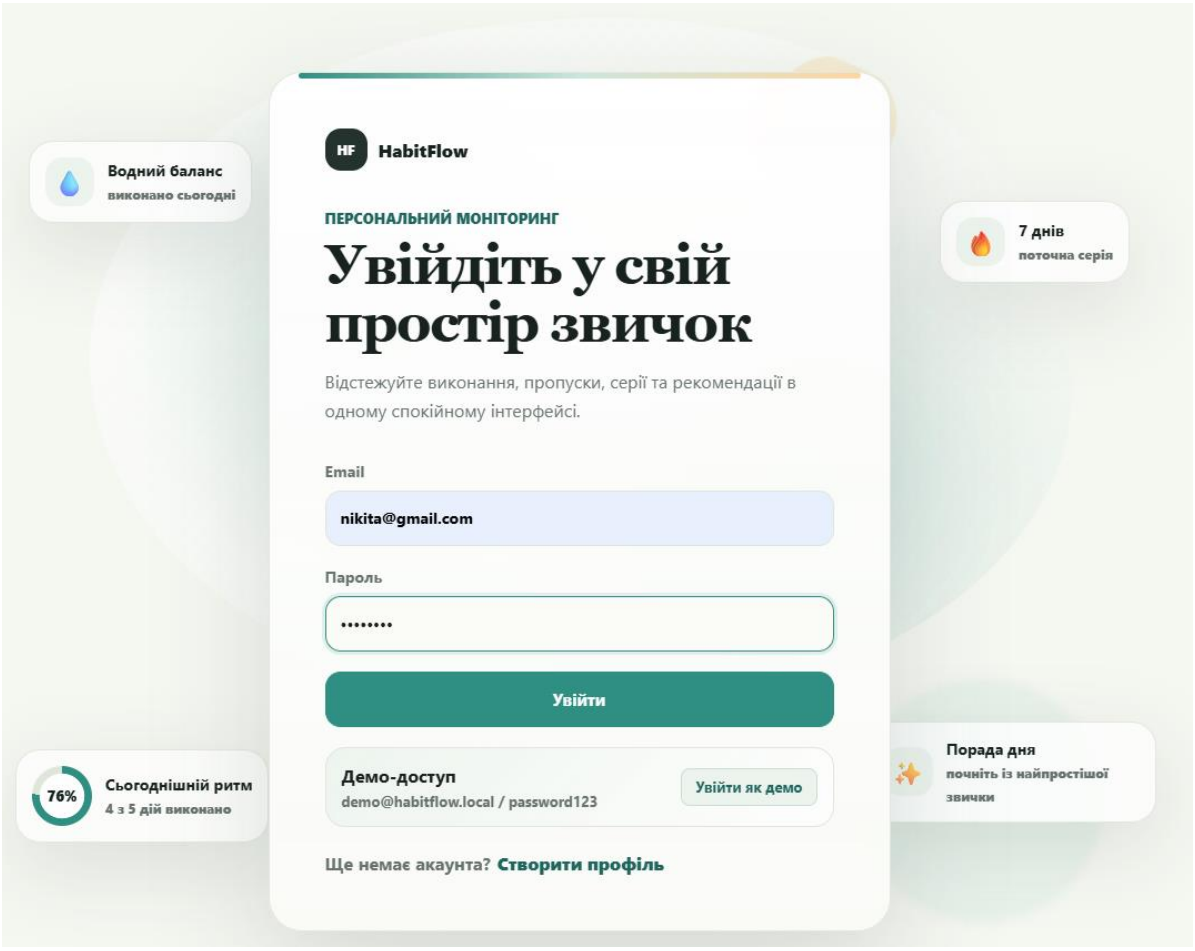


Рис. 3.1 Сторінка входу до вебзастосунку HabitFlow

Сторінка входу використовується для авторизації користувача. Вона містить поля для email і пароля, а також кнопку входу. Після успішної авторизації frontend отримує JWT-токен і відкриває основну частину застосунку. Якщо користувач ще не має облікового запису, він може перейти до сторінки реєстрації.

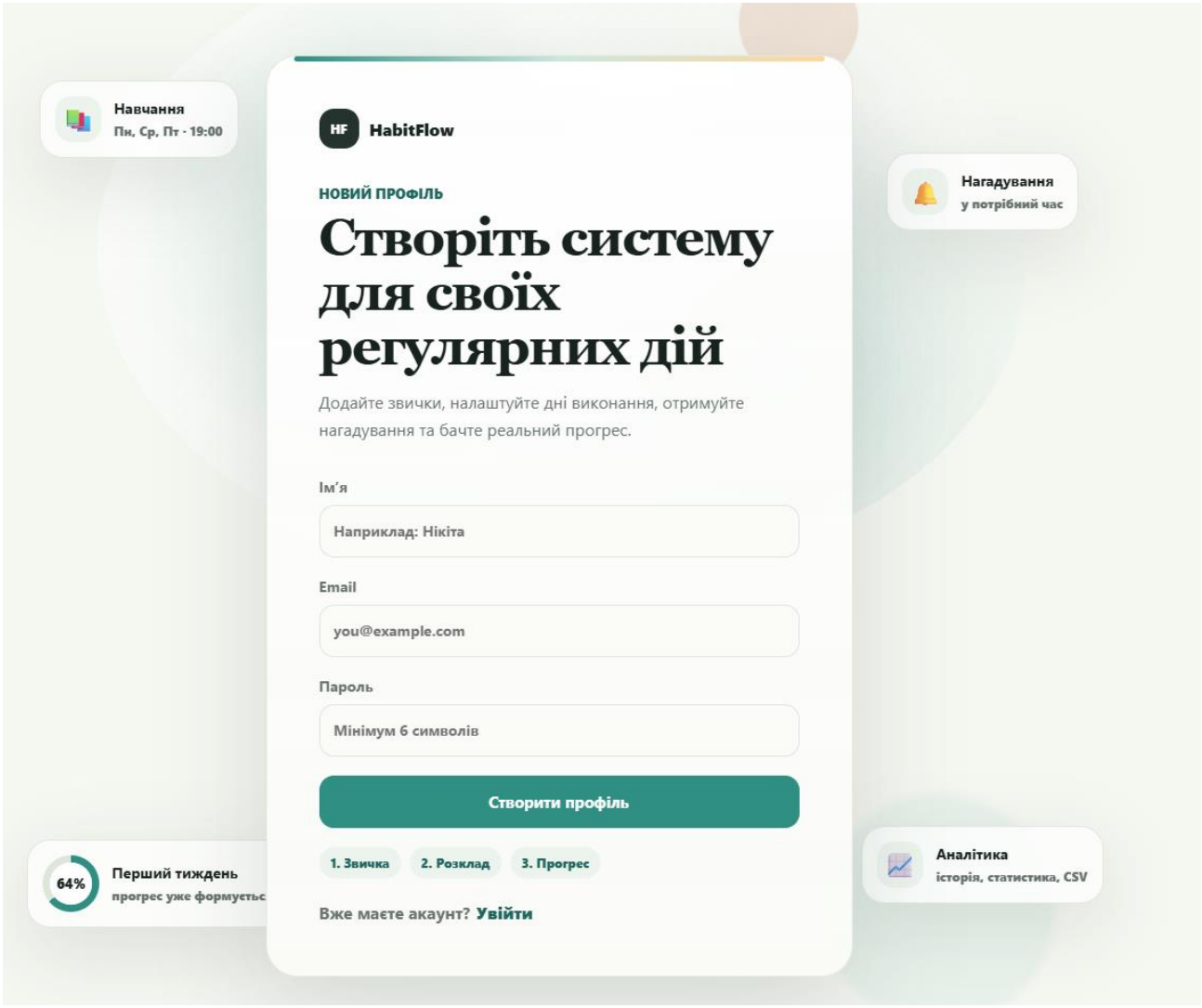


Рис. 3.2 Сторінка реєстрації користувача в HabitFlow

Сторінка реєстрації призначена для створення нового облікового запису. Користувач вводить ім'я, email і пароль. Після збереження дані передаються на backend, де пароль хешується, а користувач додається до бази даних.

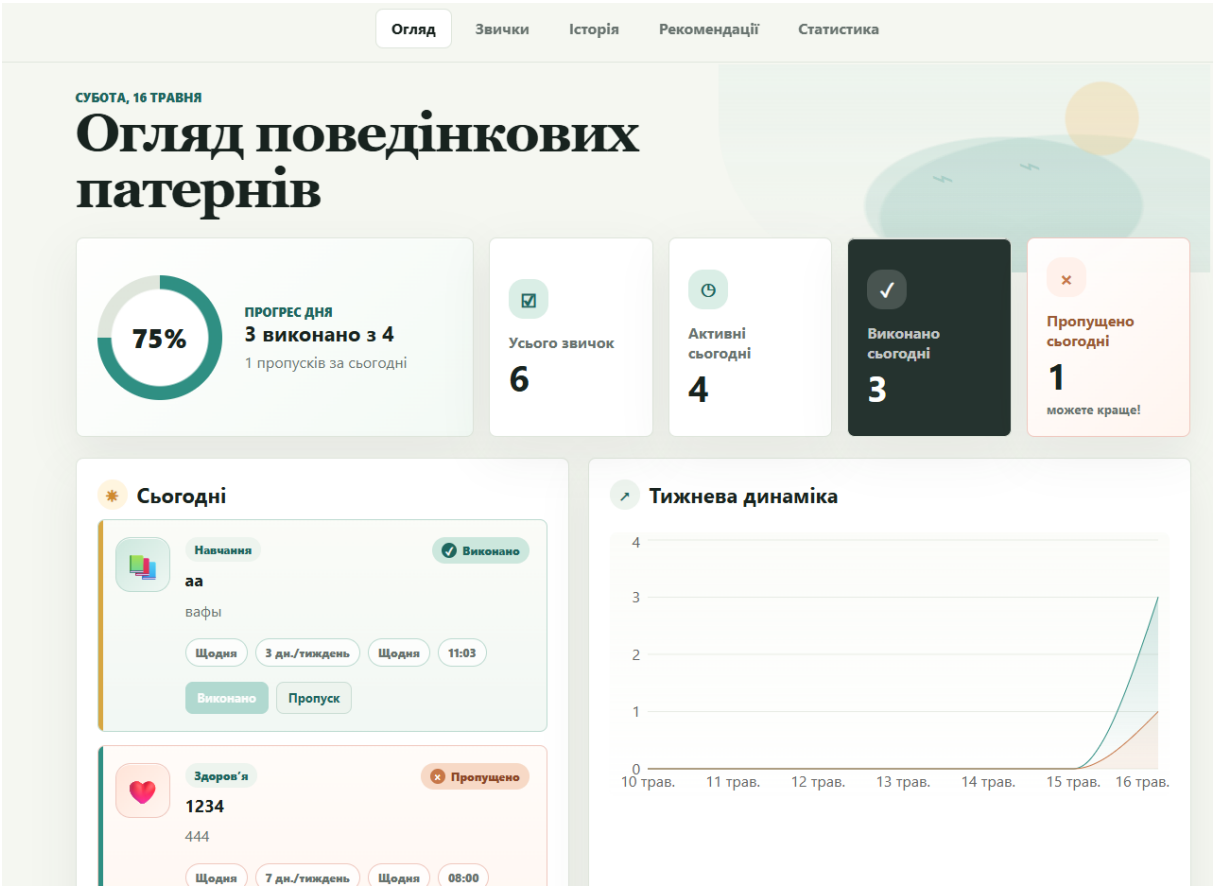


Рис. 3.3 Головна сторінка огляду звичок у HabitFlow

Головна сторінка показує стан користувача за поточний день. На ній відображається прогрес, кількість активних звичок, виконані та пропущені дії. Нижче розміщуються картки звичок, які заплановані саме на сьогодні. Користувач може одразу натиснути «Виконано» або «Пропуск».

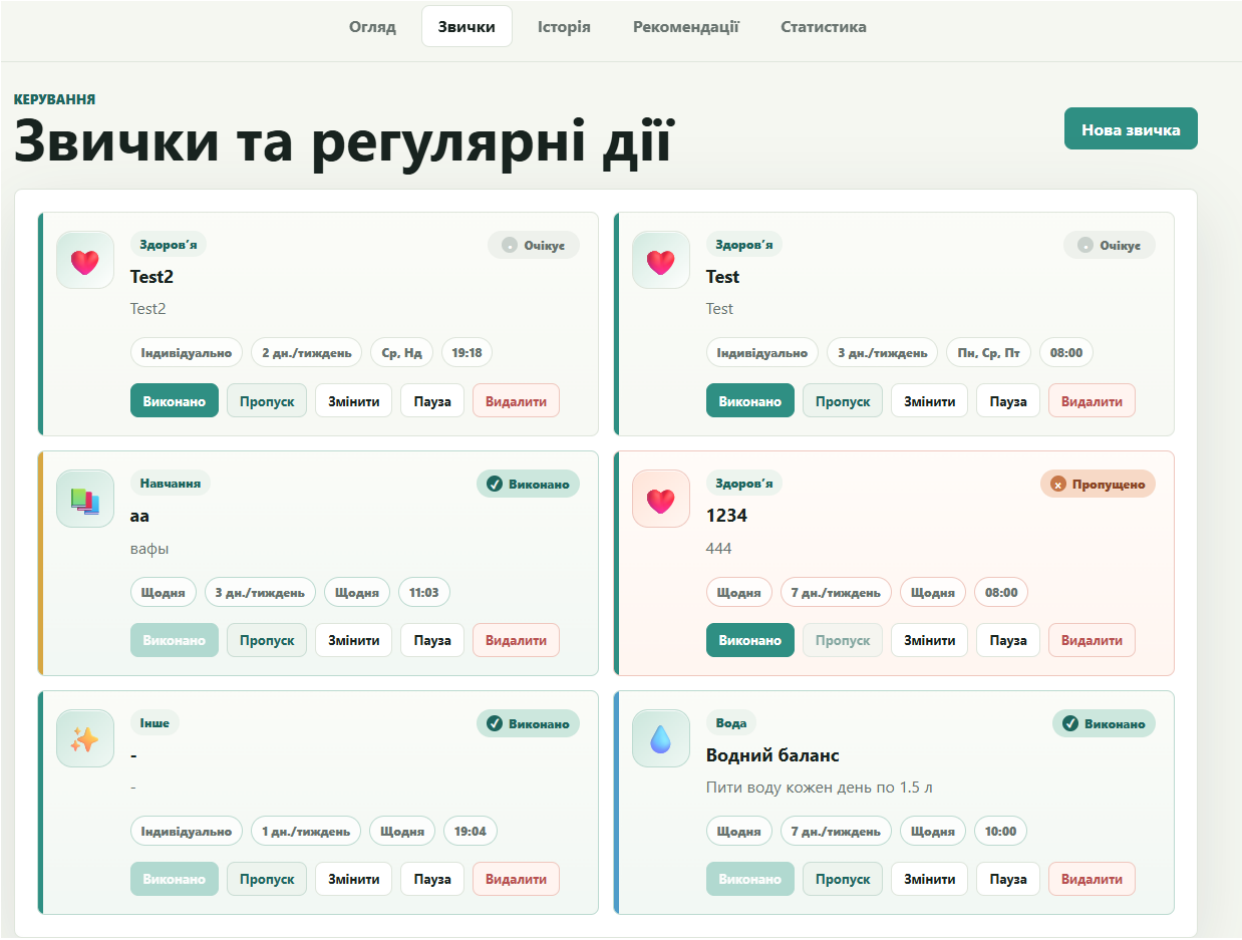


Рис. 3.4 Сторінка керування звичками

Сторінка керування звичками використовується для створення, редагування, призупинення та видалення звичок. У формі користувач задає назву, опис, категорію, дні виконання, цільову кількість днів і час нагадування. Після збереження звичка з'являється в загальному списку.

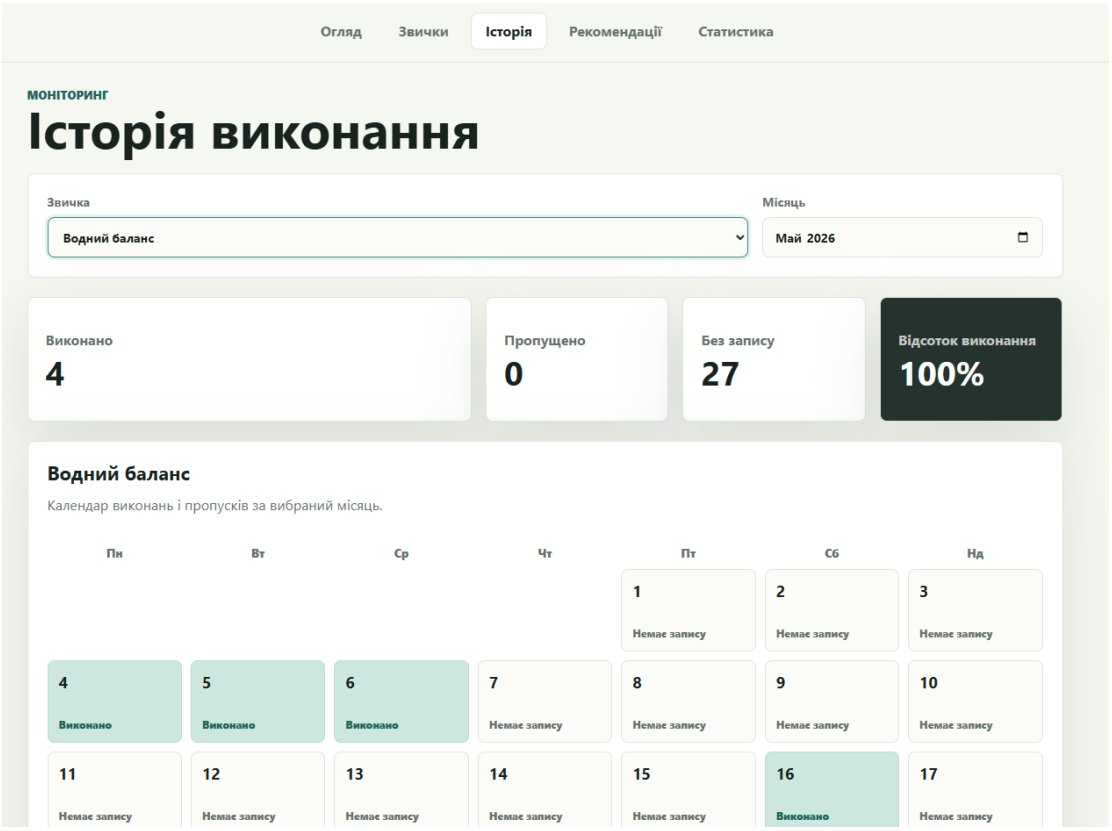


Рис. 3.5 Сторінка історії виконання звичок

Сторінка історії показує виконання звичок у календарному вигляді. Користувач може побачити, у які дні дія була виконана, пропущена або залишилася без запису. Такий формат зручний для швидкого перегляду регулярності.

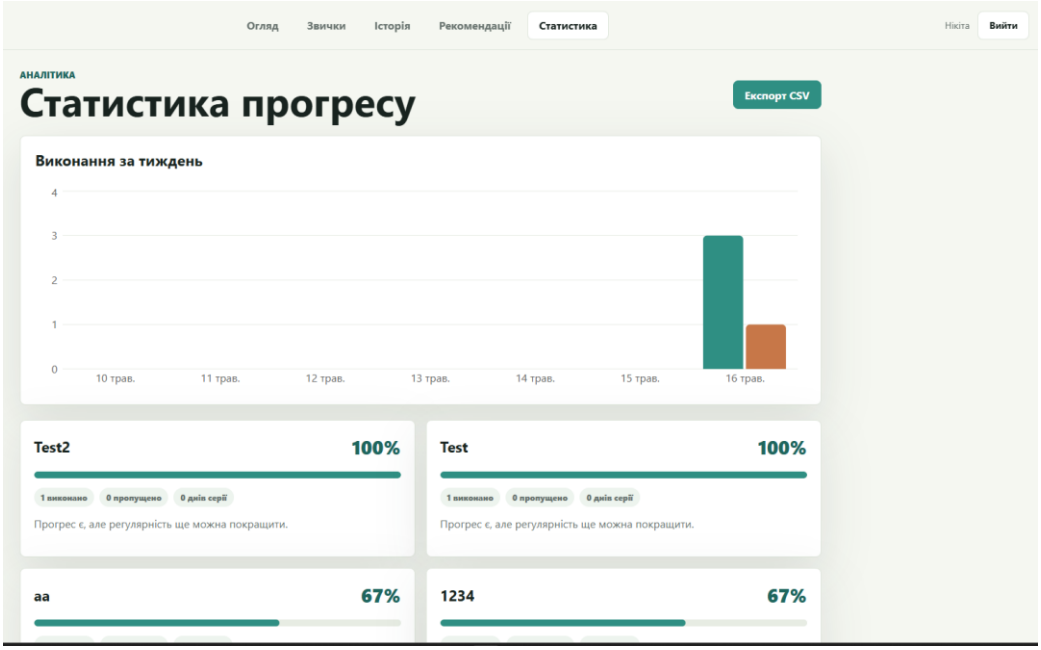


Рис. 3.6 Сторінка статистики виконання звичок

Сторінка статистики містить узагальнені показники: кількість виконаних днів, пропуски, відсоток виконання та поточну серію. Також на сторінці можуть відображатися графіки, які показують тижневу динаміку виконання і пропусків.

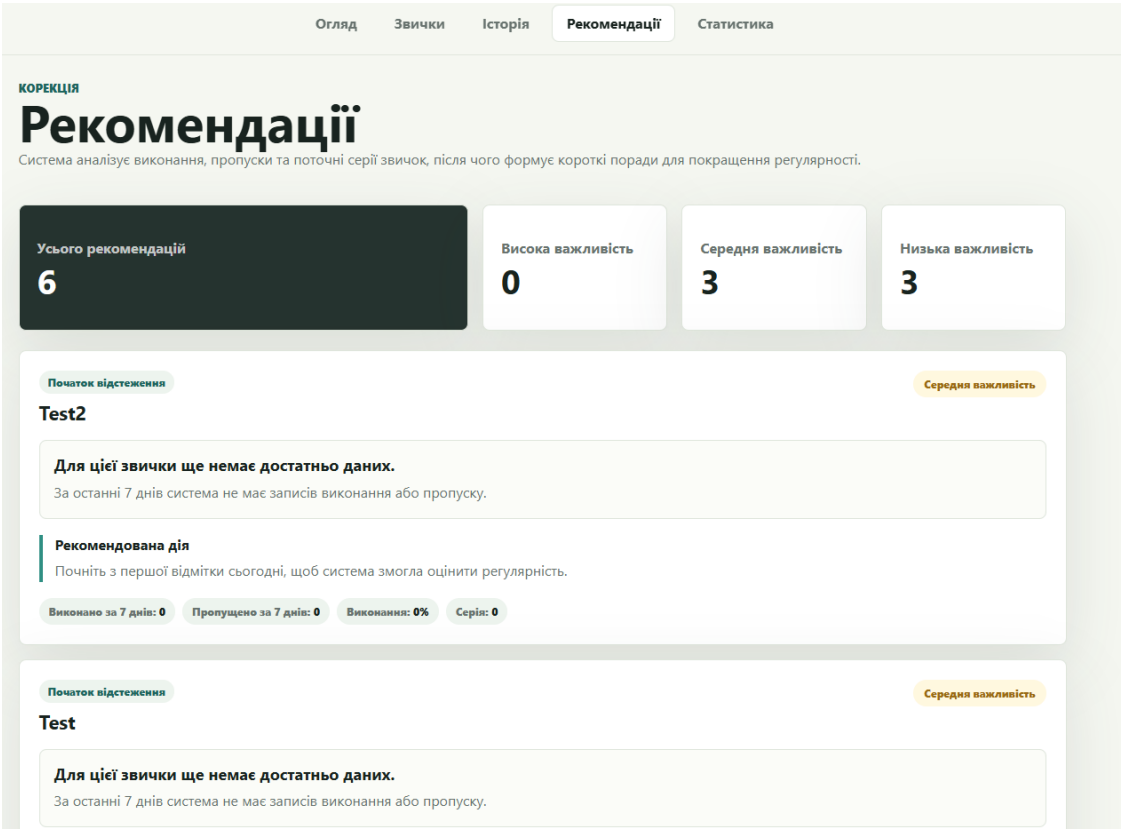


Рис. 3.7 Сторінка рекомендацій для користувача

Сторінка рекомендацій показує короткі поради на основі історії виконання. Якщо звичка часто пропускається, застосунок може запропонувати змінити графік або зменшити кількість активних днів. Якщо звичка виконується стабільно, рекомендація має підтримувальний характер.

Окремо в інтерфейсі передбачено модальне вікно для фіксації причини пропуску. Воно відкривається після натискання кнопки «Пропуск». Користувач може вказати причину, після чого вона зберігається в журналі HabitLog.

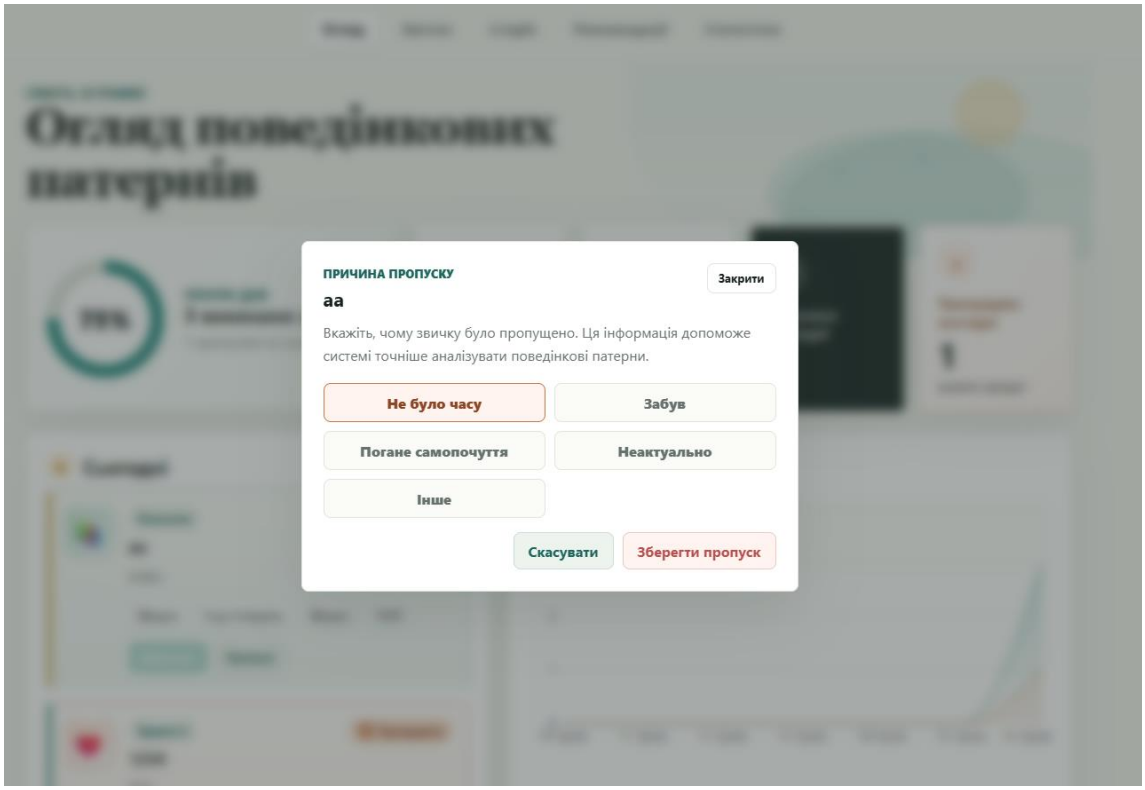


Рис. 3.8 Модальне вікно фіксації причини пропуску звички

У результаті екранні форми HabitFlow покривають основні сценарії роботи користувача: вхід, реєстрацію, перегляд актуальних звичок, керування ними, фіксацію результатів, перегляд історії, статистики та рекомендацій. Інтерфейс побудований так, щоб користувач міг швидко виконувати щоденні дії без зайвих переходів між сторінками.

4 ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ HABITFLOW

4.1 Обґрунтування вибору видів тестування

Для перевірки працездатності вебзастосунку HabitFlow було обрано функціональне, модульне та інтерфейсне тестування. Такий набір видів тестування відповідає структурі системи, оскільки застосунок має серверну частину, клієнтську частину, REST API, модулі авторизації, керування звичками, журнал виконання, статистику, рекомендації та CSV-експорт. Вибір функціонального та модульного тестування узгоджується з базовими підходами до перевірки програмного забезпечення, коли окремо перевіряються як користувацькі сценарії, так і внутрішня логіка модулів [24].

Функціональне тестування використовується для перевірки основних сценаріїв роботи користувача: реєстрації, входу до системи, створення звички, редагування графіка, фіксації виконання, фіксації пропуску, перегляду історії, статистики та рекомендацій. Для HabitFlow це основний тип перевірки, оскільки система має коректно виконувати дії, які безпосередньо впливають на дані користувача.

Модульне тестування застосовується для перевірки окремих backend-сервісів. Насамперед це стосується сервісів автентифікації, керування звичками, журналу виконання, автоматичної фіксації пропусків, статистики та рекомендацій. Такий підхід дозволяє перевірити логіку окремих методів без повного проходження користувацького сценарію через інтерфейс.

Функціональне тестування користувацького інтерфейсу потрібне для перевірки взаємодії користувача зі сторінками застосунку. У межах цього тестування перевіряється коректність переходів між сторінками, відображення карток звичок, робота форм, кнопок виконання і пропуску, сторінок історії, статистики та рекомендацій.

Обрані види тестування дозволяють перевірити HabitFlow на трьох рівнях: окремі backend-сервіси, основні функціональні сценарії та взаємодію користувача з інтерфейсом. Це дає змогу оцінити не лише працездатність окремих модулів, а й цілісну роботу вебзастосунку.

4.2 Тестування функціональних можливостей вебзастосунку

Після реалізації основних модулів HabitFlow було проведено перевірку функціональних можливостей застосунку. Тестування виконувалося за основними сценаріями, з якими користувач працює найчастіше: реєстрація, вхід, створення звички, відмітка виконання, фіксація пропуску, перегляд історії, статистики та рекомендацій.

Основна увага приділялася не тільки тому, чи відкриваються сторінки, а й тому, чи правильно зберігаються дані. Наприклад, після натискання кнопки «Виконано» у журналі HabitLog має з’явитися запис зі статусом виконання. Якщо користувач натискає «Пропуск», система повинна зберегти пропуск і, за потреби, причину.

Таблиця 4.1

Результати тестування основних функцій HabitFlow

№	Сценарій тестування	Очікуваний результат	Статус
1	Реєстрація нового користувача	Створюється новий обліковий запис	Виконано
2	Вхід до системи	Користувач отримує доступ до основних сторінок	Виконано
3	Створення звички	Нова звичка зберігається та з’являється у списку	Виконано
4	Редагування звички	Змінені параметри відображаються після збереження	Виконано
5	Призупинення звички	Звичка не показується серед активних	Виконано
6	Фіксація виконання	У журналі створюється запис зі статусом виконання	Виконано
7	Фіксація пропуску	У журналі створюється запис пропуску з причиною або без неї	Виконано
8	Перегляд історії	Календар показує виконані та пропущені дні	Виконано

9	Перегляд статистики	Відображаються відсоток виконання, пропуски та серія	Виконано
10	Перегляд рекомендацій	Користувач бачить поради на основі історії виконання	Виконано
11	CSV-експорт	Завантажується файл зі статистичними даними	Виконано
12	Вихід із системи	Користувач втрачає доступ до захищених сторінок	Виконано

Продовження Таблиця 4.1

Під час тестування також перевірялося, чи не змішуються дані різних користувачів. Для цього можна створити два облікові записи й додати різні звички. Після входу в кожен профіль користувач має бачити тільки власні записи, історію та статистику.

Окремо перевірялася робота захищених сторінок. Якщо користувач не авторизований, він не повинен відкривати головну сторінку, список звичок, історію або статистику. У такому випадку застосунок має перенаправити його на сторінку входу.

За результатами перевірки основні функції HabitFlow працюють коректно. Користувач може створювати звички, відмічати виконання або пропуски, переглядати історію, аналізувати статистику та отримувати рекомендації.

4.3 Модульне тестування backend-сервісів

Для модульного тестування backend-логіки доцільно використовувати JUnit, який є поширеним інструментом перевірки Java-коду [25]. На цьому етапі важливо було переконатися, що основна логіка працює правильно ще до перевірки всього застосунку через інтерфейс. Тестувалися не візуальні елементи, а саме обробка даних на серверній стороні.

Основну увагу було приділено сервісам, які працюють зі звичками, журналом виконання, статистикою та рекомендаціями. Наприклад, під час тестування сервісу звичок перевірялося створення нового запису, редагування параметрів і прив’язка

звички до конкретного користувача. Для журналу виконання перевірялося створення записів зі статусами виконання та пропуску.

Окремо перевірялася логіка автоматичної фіксації пропущених днів. Для цього моделювалася ситуація, коли звичка була запланована на певну дату, але в HabitLog не було жодного запису. Очікуваний результат — створення пропуску з ознакою автоматичного створення.

Під час тестування також перевірялася робота з даними різних користувачів. Це важливо для HabitFlow, бо кожна звичка й кожен запис журналу мають належати конкретному обліковому запису. Якщо сервіс отримує запит від одного користувача, він не повинен повертати або змінювати записи іншого.

Результати модульного тестування показали, що основні backend-сервіси виконують свої задачі коректно. Реєстрація, авторизація, створення звичок, фіксація виконання, пропуски, статистика, рекомендації та CSV-експорт працюють відповідно до очікуваної логіки.

Таблиця 4.2

Результати модульного тестування backend-сервісів HabitFlow

№	Модуль / сервіс	Що перевірялося	Очікуваний результат	Статус
1	AuthService	Реєстрація користувача	Користувач створюється, пароль зберігається у хешованому вигляді	Виконано
2	AuthService	Вхід до системи	Після правильних даних формується JWT-токен	Виконано
3	HabitService	Створення звички	Звичка зберігається і прив’язується до поточного користувача	Виконано
4	HabitService	Редагування звички	Параметри звички оновлюються без зміни власника	Виконано
5	HabitService	Призупинення звички	Звичка отримує неактивний статус	Виконано
6	HabitLogService	Фіксація виконання	У журналі створюється запис зі статусом виконання	Виконано
7	HabitLogService	Фіксація пропуску	У журналі створюється запис пропуску, за потреби з причиною	Виконано

8	HabitLogService	Перевірка дублювання записів	Для однієї звички й дати не створюються зайві дублікати	Виконано
9	StatisticsService	Формування статистики	Рахуються виконані дні, пропуски, відсоток і серія	Виконано
10	RecommendationService	Формування рекомендацій	Для звичок з низьким виконанням створюється відповідна порада	Виконано
11	CsvExportService	Експорт даних	Формується CSV-файл зі статистикою користувача	Виконано

Продовження Таблиця 4.2

4.4 Функціональне тестування користувацького інтерфейсу

Перевірка інтерфейсу користувача виконувалася з урахуванням основних сценаріїв взаємодії: переходів між сторінками, роботи форм, кнопок, повідомлень і відображення результатів дій [26]. На цьому етапі перевірялися не окремі backend-сервіси, а повний шлях користувача: від відкриття сторінки до отримання результату в інтерфейсі.

Основна увага була приділена сторінкам входу, реєстрації, головного огляду, керування звичками, історії, статистики та рекомендацій. Для кожної сторінки перевірялося, чи правильно відображаються дані, чи працюють кнопки, чи оновлюється інтерфейс після дії користувача.

Таблиця 4.3

Результати функціонального тестування користувацького інтерфейсу HabitFlow

№	**Екранна форма**	**Дія користувача**	**Очікуваний результат**	**Статус**
1	Сторінка входу	Ввести email і пароль	Після успішного входу відкривається головна сторінка	Виконано
2	Сторінка реєстрації	Заповнити форму нового користувача	Створюється обліковий запис	Виконано
3	Головна сторінка	Переглянути звички на сьогодні	Відображаються актуальні звички та прогрес дня	Виконано
4	Картка звички	Натиснути **«Виконано»**	Статус звички змінюється, прогрес оновлюється	Виконано
5	Картка звички	Натиснути **«Пропуск»**	Відкривається можливість зафіксувати пропуск	Виконано

6	Сторінка звичок	Створити нову звичку	Нова звичка з’являється у списку	Виконано
7	Сторінка звичок	Відредагувати звичку	Змінені параметри відображаються після збереження	Виконано
8	Сторінка історії	Вибрати звичку для перегляду	Відображається календар виконання	Виконано
9	Сторінка статистики	Переглянути статистику	Показуються показники виконання, пропуски та графік	Виконано
10	Сторінка рекомендацій	Переглянути поради	Відображаються рекомендації на основі історії	Виконано
11	Навігаційне меню	Перейти між розділами	Сторінки відкриваються без повного перезавантаження	Виконано
12	Вихід із системи	Натиснути кнопку виходу	Користувач повертається на сторінку входу	Виконано

Продовження Таблиця 4.3

Під час тестування також перевірялося, чи правильно працюють захищені сторінки. Якщо користувач не авторизований, він не повинен бачити головну сторінку, історію, статистику або рекомендації. У такому випадку застосунок перенаправляє його на сторінку входу.

Окремо перевірялася зручність щоденної взаємодії. Користувач має мати змогу швидко відмітити звичку без зайвих переходів. Для цього на головній сторінці розміщено картки актуальних звичок і кнопки «Виконано» та «Пропуск».

Результати функціонального тестування показали, що основні екранні форми працюють коректно. Користувач може пройти реєстрацію, увійти до системи, створити звичку, зафіксувати виконання або пропуск, переглянути історію, статистику та рекомендації.

4.5 Результати тестування вебзастосунку

Після проведення функціонального та модульного тестування було перевірено основні можливості вебзастосунку HabitFlow. Тестування показало, що реалізовані модулі працюють відповідно до поставлених вимог: користувач може

зареєструватися, увійти до системи, створити звичку, зафіксувати виконання або пропуск, переглянути історію, статистику та рекомендації.

Окремо перевірялася робота з персональними даними. Після входу користувач бачить тільки власні звички та записи журналу. Це підтверджує коректну роботу авторизації через JWT і зв’язку даних із конкретним обліковим записом. Якщо користувач не авторизований, доступ до захищених сторінок обмежується.

Під час тестування журналу виконання було перевірено створення записів зі статусами виконання та пропуску. Після натискання кнопки «Виконано» у журналі створюється відповідний запис. Після натискання «Пропуск» система зберігає пропуск і, за наявності, його причину. Це дозволяє надалі використовувати ці дані для історії та статистики.

Сторінка історії коректно відображає виконані та пропущені дні. Сторінка статистики показує узагальнені показники: кількість виконаних днів, пропуски, відсоток виконання та серію. Рекомендаційний модуль формує короткі поради на основі накопичених даних. Наприклад, якщо звичка часто пропускається, користувачу пропонується переглянути графік або зменшити навантаження.

Також було перевірено CSV-експорт. Після натискання відповідної кнопки користувач отримує файл зі статистичними даними. Це дає змогу переглядати результати не тільки в інтерфейсі HabitFlow, а й у табличних редакторах.

За результатами тестування критичних помилок у роботі основних сценаріїв не виявлено. Реалізований вебзастосунок виконує поставлені задачі та дозволяє користувачу працювати зі звичками в повному циклі: створення, планування, фіксація результатів, перегляд історії, аналіз статистики та отримання рекомендацій.

4.6 Оцінювання ефективності програмного продукту

Ефективність програмного продукту HabitFlow оцінювалася за тим, наскільки реалізований вебзастосунок спрощує процес контролю звичок і дає користувачу більше інформації для корекції власної регулярної поведінки. Для цього враховувалися функціональні можливості системи, структура взаємодії користувача із застосунком, результати тестування та наявність модулів, які автоматизують обробку даних.

До розробки HabitFlow користувач міг контролювати регулярні дії за допомогою звичайного списку справ, календаря або окремого трекера звичок. Такий підхід не завжди дає повну картину, оскільки планування, фіксація виконання, пропуски, історія, статистика та рекомендації часто розділені між різними інструментами. У HabitFlow ці функції об’єднані в одному вебзастосунку, що зменшує кількість ручних дій і спрощує аналіз результатів.

Основний ефект від використання HabitFlow полягає в автоматизації роботи з даними про звички. Користувач створює звичку один раз, після чого система самостійно враховує її графік, показує актуальні дії на поточний день, зберігає виконання або пропуск у журналі, формує історію, статистику та рекомендації. Окремо реалізовано автоматичне визначення пропущених днів, що дозволяє не втрачати заплановані дати без відміток.

Таблиця 4.4

Оцінювання ефективності програмного продукту HabitFlow

Критерій оцінювання	**Реалізація в HabitFlow**	**Очікуваний результат**
Зменшення ручного контролю	автоматична фіксація пропущених днів	користувачу не потрібно вручну перевіряти всі заплановані дати
Централізація даних	звички, історія, статистика й рекомендації зберігаються в одній системі	користувач отримує повну картину виконання в одному інтерфейсі
Швидкість щоденної взаємодії	кнопки «Виконано» і «Пропуск» на головній сторінці	основна дія виконується без переходу між багатьма сторінками
Точність статистики	використання журналу HabitLog і запланованих днів виконання	статистика враховує не лише виконання, а й пропуски
Підтримка корекції поведінки	модуль рекомендацій на основі історії та статистики	користувач отримує підказки щодо зміни графіка або навантаження

Переносимість даних	CSV-експорт статистики	користувач може зберегти або проаналізувати результати поза системою
Захист персональних даних	JWT-авторизація і розмежування даних користувачів	користувач має доступ лише до власних звичок і статистики

Продовження Таблиця 4.4

Результати тестування показали, що основні функціональні модулі HabitFlow працюють коректно: користувач може зареєструватися, увійти до системи, створити звичку, зафіксувати виконання або пропуск, переглянути історію, статистику, рекомендації та експортувати дані у форматі CSV. Це підтверджує, що застосунок виконує основні задачі, поставлені під час проєктування.

Ефективність HabitFlow також забезпечується модульною архітектурою. Окремі модулі автентифікації, керування звичками, журналу виконання, статистики, рекомендацій і CSV-експорту можна змінювати або розширювати без повної перебудови системи. Наприклад, у майбутньому до застосунку можна додати push-сповіщення, мобільну версію або складніші алгоритми персоналізованих рекомендацій.

Розроблений вебзастосунок HabitFlow є ефективним для задачі персонального моніторингу та корекції поведінкових патернів, оскільки поєднує планування, фіксацію результатів, історію, статистику, рекомендації та експорт даних в одному програмному продукті. Це дозволяє користувачу не лише відмічати виконання звичок, а й бачити реальну регулярність власних дій та приймати обґрунтовані рішення щодо зміни графіка або навантаження.

ВИСНОВКИ

У кваліфікаційній роботі розроблено вебзастосунок HabitFlow для моніторингу та корекції індивідуальних поведінкових патернів користувача. На відміну від звичайних менеджерів задач або списків справ, розроблений застосунок орієнтований не лише на фіксацію дій, а й на аналіз регулярності їх виконання, збереження історії та формування поведінкових метрик.

На основі аналізу предметної галузі та існуючих програмних рішень для відстеження звичок було визначено основні проблеми подібних систем: недостатню гнучкість налаштування графіків, обмежені можливості аналізу статистики, слабку підтримку роботи з пропусками та відсутність механізмів корекції поведінкових патернів. У результаті аналізу було встановлено необхідність поєднання історії виконання, статистики, рекомендацій і вебдоступу в межах єдиного програмного рішення.

Під час дослідження було проаналізовано застосунки Loop Habit Tracker, Habitica, Todoist, Streaks і Google Calendar. Проведений аналіз показав, що кожне з рішень має окремі переваги, проте жодне з них повністю не забезпечує комплексний моніторинг регулярних дій користувача з урахуванням статистики, історії виконання, пропусків, рекомендацій та експорту даних. На основі цього було сформовано концепцію вебзастосунку HabitFlow.

У результаті проєктування визначено функціональні та нефункціональні вимоги до системи та обґрунтовано вибір клієнт-серверної архітектури. Для реалізації серверної частини використано Java та Spring Boot, що забезпечують модульність і розширюваність системи. Для захисту доступу застосовано Spring Security, JWT-авторизацію та BCrypt-хешування паролів. Збереження даних реалізовано через PostgreSQL, а керування структурою бази даних — за допомогою Liquibase. Клієнтську частину створено на React із використанням Axios, React Router та Recharts. Для контейнеризованого запуску застосовано Docker Compose.

Під час проєктування було побудовано діаграму варіантів використання, описано сценарії взаємодії користувача із системою та спроєктовано структуру

бази даних. У результаті сформовано структуру, що складається з основних сутностей User, Habit, HabitLog, HabitCategory та HabitScheduleDay. Така модель забезпечує ізоляцію даних користувачів, підтримку графіків виконання звичок і збереження повної історії виконання та пропусків.

У процесі реалізації було створено функціональні модулі реєстрації та авторизації користувачів, керування звичками, журналу виконання, статистики, рекомендацій, нагадувань та CSV-експорту. У результаті система дозволяє користувачу створювати звички, фіксувати виконання або пропуски, переглядати календарну історію, аналізувати відсоток виконання та поточну серію, а також отримувати рекомендації щодо корекції регулярних дій.

Окрему роль у роботі системи виконує журнал HabitLog, який забезпечує збереження історії виконання звичок за конкретні дати. На основі цих даних реалізовано автоматичне формування статистики, визначення серій виконання та механізм автоматичної фіксації пропущених днів. Це дозволяє системі аналізувати не лише окремі дії користувача, а й загальну динаміку поведінки.

У застосунку також реалізовано модуль експорту статистичних даних у форматі CSV, що дає змогу використовувати результати поза межами системи та виконувати додатковий аналіз у табличних редакторах.

Після завершення реалізації проведено тестування основних сценаріїв роботи застосунку. У результаті тестування підтверджено коректність роботи модулів авторизації, керування звичками, журналу виконання, статистики, рекомендацій та CSV-експорту. Також підтверджено стабільність взаємодії між frontend та backend частинами, коректну роботу REST API та розмежування даних між користувачами.

Результатом кваліфікаційної роботи став вебзастосунок HabitFlow, який забезпечує комплексний моніторинг регулярних дій користувача та дозволяє аналізувати індивідуальні поведінкові патерни на основі історії виконання, статистики та рекомендацій. Архітектура системи підтримує можливість подальшого розширення функціональності, зокрема створення мобільної версії, інтеграції push-сповіщень, покращення аналітичного модуля та розвитку механізму рекомендацій.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Волошин Н.Н., Яскевич В.О. Розробка програмного забезпечення комплексного моніторингу та корекції індивідуальних поведінкових патернів користувача. Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація», 1-3 травня 2026 року, Київ, ДУІКТ. Збірник тез. (подано до друку).

2. Волошин Н.Н., Яскевич В.О. Розробка програмного забезпечення комплексного моніторингу та корекції індивідуальних поведінкових патернів користувача. VI Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2026 року, м. Київ, ДУІКТ. Збірник тез. (подано до друку).

ПЕРЕЛІК ПОСИЛАНЬ

1. Michie S., Richardson M., Johnston M., Abraham C., Francis J., Hardeman W., Eccles M. P., Cane J., Wood C. E. The Behavior Change Technique Taxonomy (v1) of 93 Hierarchically Clustered Techniques: Building an International Consensus for the Reporting of Behavior Change Interventions. Annals of Behavioral Medicine. 2013. Vol. 46, No. 1. P. 81–95. DOI: 10.1007/s12160-013-9486-6.

2. Loop Habit Tracker : офіційний вебсайт. URL: https://loophabits.org/ (дата звернення: 14.05.2026).

3. Habitica : офіційний вебсайт. URL: https://habitica.com/ (дата звернення: 14.05.2026).

4. Todoist Features : офіційний вебсайт. URL: https://www.todoist.com/features (дата звернення: 15.05.2026).

5. Streaks : офіційний вебсайт. URL: https://streaksapp.com/ (дата звернення: 15.05.2026).

6. Google Calendar Help. Create a recurring event. URL: https://support.google.com/calendar/answer/37115 (дата звернення: 16.05.2026).

7. Bloch J. Effective Java. 3rd ed. Boston : Addison-Wesley Professional, 2018. 416 p.

8. Walls C. Spring in Action. 6th ed. Shelter Island : Manning Publications, 2022. 520 p.

9. Spilca L. Spring Security in Action. Shelter Island : Manning Publications, 2020. 560 p.

10. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. 2015. URL: https://www.rfc-editor.org/rfc/rfc7519 (дата звернення: 16.05.2026).

11. Obe R. O., Hsu L. S. PostgreSQL: Up and Running. 3rd ed. Sebastopol : O'Reilly Media, 2017. 312 p.

12. Liquibase Documentation : офіційна документація. URL: https://docs.liquibase.com/ (дата звернення: 17.05.2026).

13. Яскевич В. О. JavaScript-бібліотека React : навч. посіб. для студентів спеціальності «Комп'ютерні науки». Київ : Київський університет імені Бориса Грінченка, 2023.

14. Vite Documentation : офіційна документація. URL: https://vite.dev/guide/ (дата звернення: 17.05.2026).

15. Axios Documentation : офіційна документація. URL: https://axios-http.com/docs/intro (дата звернення: 18.05.2026).

16. React Router Documentation : офіційна документація. URL: https://reactrouter.com/ (дата звернення: 18.05.2026).

17. Recharts Documentation : офіційна документація. URL: https://recharts.org/en-US/ (дата звернення: 19.05.2026).

18. OpenAPI Specification : офіційна специфікація. URL: https://spec.openapis.org/oas/latest.html (дата звернення: 20.05.2026).

19. Poulton N. Docker Deep Dive. Leanpub, 2023.

20. Gourley D., Totty B. HTTP: The Definitive Guide. Sebastopol : O'Reilly Media, 2002. 656 p.

21. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley Professional, 2003. 208 p.

22. Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts. 7th ed. New York : McGraw-Hill Education, 2019. 1344 p.

23. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : doctoral dissertation. Irvine : University of California, 2000.

24. Spillner A., Linz T., Schaefer H. Software Testing Foundations: A Study Guide for the Certified Tester Exam. 4th ed. Santa Barbara : Rocky Nook, 2014. 304 p.

25. Langr J., Hunt A., Thomas D. Pragmatic Unit Testing in Java 8 with JUnit. Raleigh : Pragmatic Bookshelf, 2015. 230 p.

26. Nielsen J. Usability Engineering. San Francisco : Morgan Kaufmann, 1993. 362 p.

27. Richardson L., Amundsen M., Ruby S. RESTful Web APIs. Sebastopol : O'Reilly Media, 2013. 406 p.

28. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 616 p.

29. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley Professional, 2003. 560 p.

30. Duckett J. JavaScript and jQuery: Interactive Front-End Web Development. Indianapolis : Wiley, 2014. 640 p.

31. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2016. 816 p.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Розробка програмного забезпечення комплексного моніторингу та корекції індивідуальних поведінкових патернів користувача

Виконав: студент 4 курсу
групи ПД-43
Волошин Нікіта Назарійович
Керівник роботи:
канд.техн.наук, доц. Яскевич Владислав Олександрович

Київ – 2026

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – підвищення зручності контролю та корекції індивідуальних поведінкових патернів користувача за рахунок вебзастосунку HabitFlow.

Об’єкт дослідження – процес комплексного моніторингу та корекції індивідуальних поведінкових патернів користувача.

Предмет дослідження – програмне забезпечення для моніторингу, аналізу та корекції звичок.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати предметну галузь моніторингу та корекції індивідуальних поведінкових патернів користувача.

2. Дослідити існуючі програмні рішення для відстеження звичок, регулярних дій, нагадувань і статистики.

3. Визначити функціональні та нефункціональні вимоги до вебзастосунку HabitFlow.

4. Обгрунтувати вибір технологій для реалізації клієнтської та серверної частин застосунку.

5. Спроєктувати архітектуру системи, структуру бази даних, REST API та основні алгоритми роботи.

6. Реалізувати вебзастосунок HabitFlow із модулями авторизації, керування звичками, журналу виконання, статистики, рекомендацій і CSV-експорту.

7. Провести тестування основних функціональних можливостей вебзастосунку.

3

АНАЛІЗ АНАЛОГІВ

Функціональність / Показник	**Loop Habit Tracker**	**Habitica**	**Todoist**	**Streaks**	**Google Calendar**	**HabitFlow**
Створення звичок	+	+	-	+	-	+
Фіксація виконання	+	+	+	+	-	+
Фіксація пропуску	+	+	+	+	-	+
Збереження причини пропуску	-	-	-	+	-	+
Автоматичне визначення пропущених днів	+	+	-	+	-	+
Календарна історія виконання	+	+	-	+	+	+
Статистика виконання	+	+	+	+	-	+
Відображення серії виконання	+	+	-	+	-	+
Рекомендації щодо корекції звички	-	-	-	-	-	+
Гейміфікація	-	+	-	-	-	+
Вебдоступ через браузер	-	+	+	-	+	+
Серверне збереження даних користувача	-	+	+	+	+	+
Експорт статистики у CSV	-	-	-	-	-	+
Орієнтація саме на поведінкові патерни	+	-	-	+	-	+

4

ВИМОГИ ДО ЗАСТОСУНКУ

Функціональні вимоги:

1. Реєстрація та автентифікація користувача через JWT-токени.
2. Створення, редагування, призупинення та видалення звичок.
3. Налаштування категорії, графіка виконання та часу нагадування.
4. Фіксація виконання або пропуску звички з можливістю збереження причини пропуску.
5. Автоматичне формування записів про пропущені дні.
6. Перегляд історії виконання звичок за календарними датами.
7. Формування статистики: відсоток виконання, кількість пропусків, поточна серія.
8. Генерація рекомендацій на основі журналу виконання.
9. Експорт статистичних даних у формат CSV.

ВИМОГИ ДО ЗАСТОСУНКУ

Нефункціональні вимоги:

1. Захист доступу до API забезпечується Spring Security та JWT-авторизацією.
2. Паролі користувачів зберігаються у вигляді BCrypt-хешів.
3. REST API підтримує обмін даними у форматі JSON через HTTP.
4. Час відповіді основних API-запитів не перевищує 1–2 секунд при стандартному навантаженні.
5. Дані користувачів ізольовані через прив’язку записів до user_id.
6. Система підтримує контейнеризований запуск через Docker Compose.
7. База даних PostgreSQL забезпечує збереження історії виконання та цілісність зв’язків між сутностями.
8. Frontend адаптований для роботи у сучасних веббраузерах.
9. Архітектура застосунку підтримує подальше розширення функціональності без зміни основних модулів.

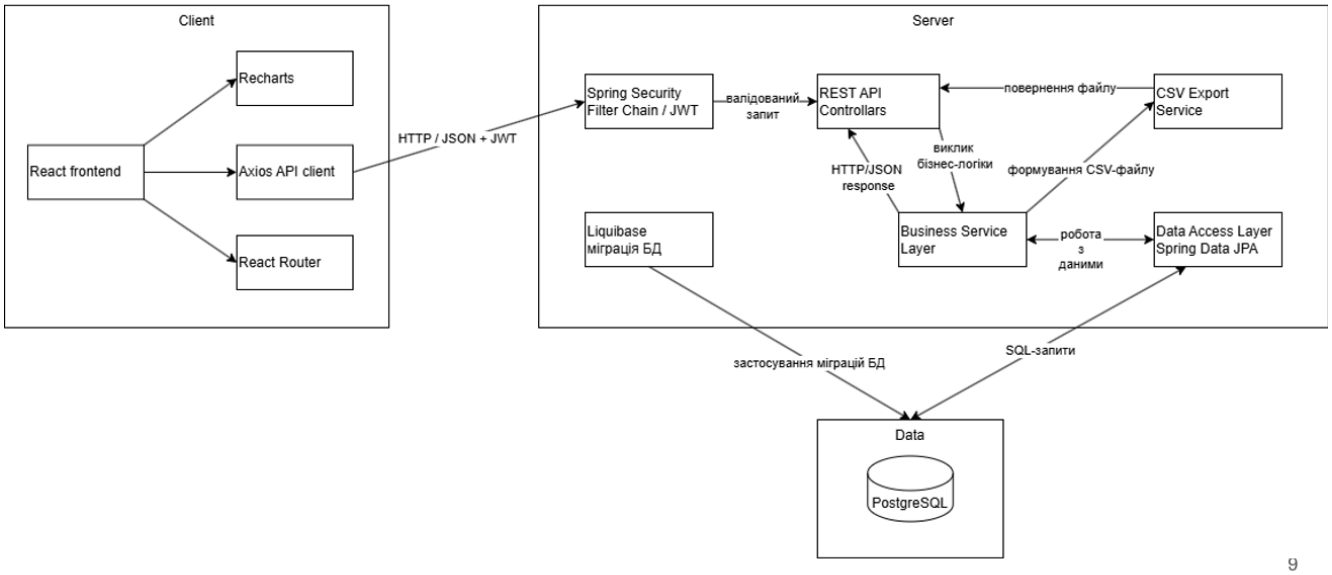
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

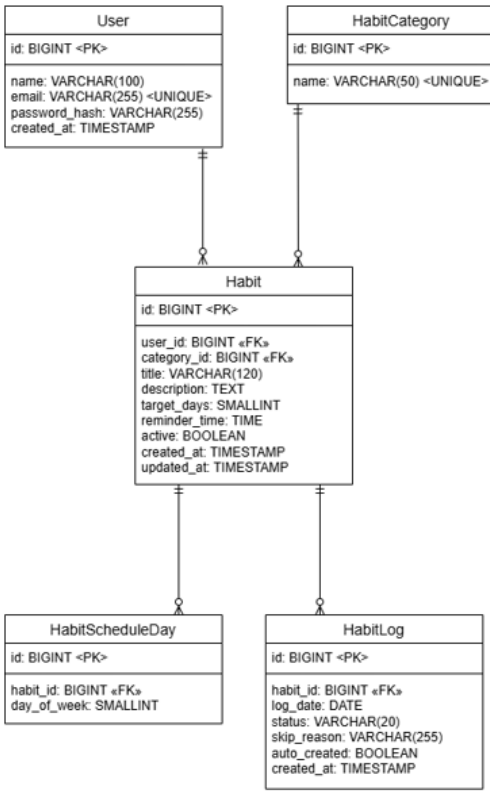
8

АРХІТЕКТУРА ЗАСТОСУНКУ



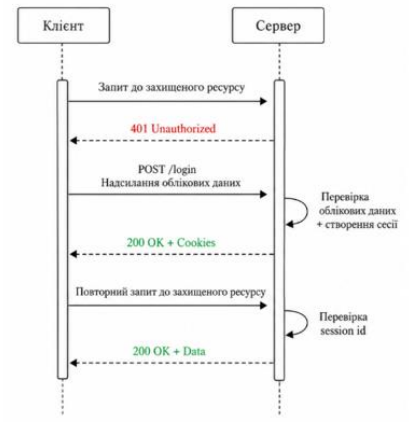
9

СТРУКТУРА БАЗИ ДАННИХ

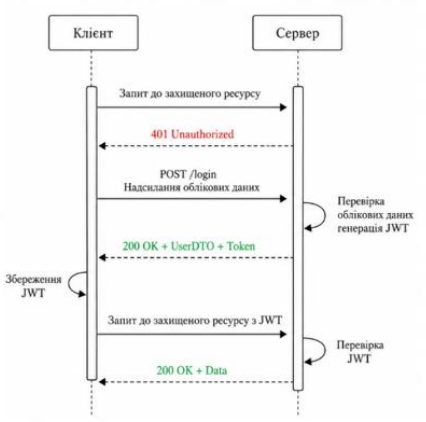


10

ДІАГРАМИ ПОСЛІДОВНОСТІ АВТОРИЗАЦІЇ



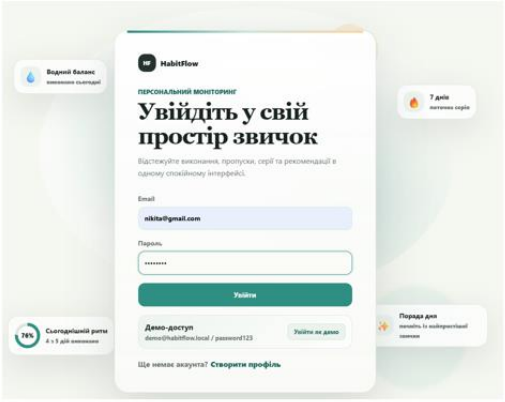
Діаграма послідовності простої авторизації



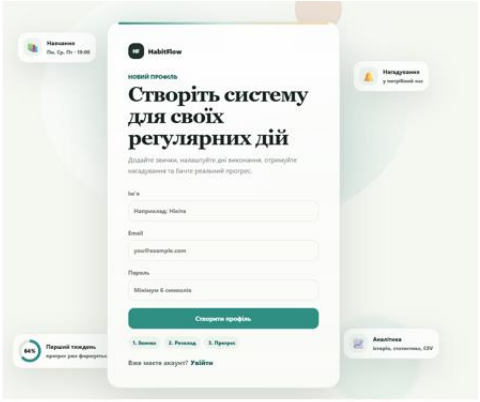
Діаграма послідовності авторизації з використанням JWT

11

ЕКРАННІ ФОРМИ



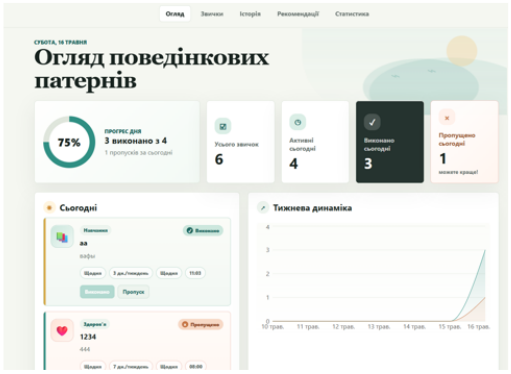
Сторінка входу до web-застосунку HabitFlow



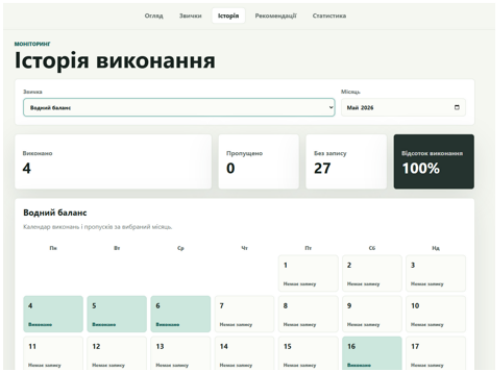
Сторінка реєстрації користувача в HabitFlow

12

ЕКРАННІ ФОРМИ

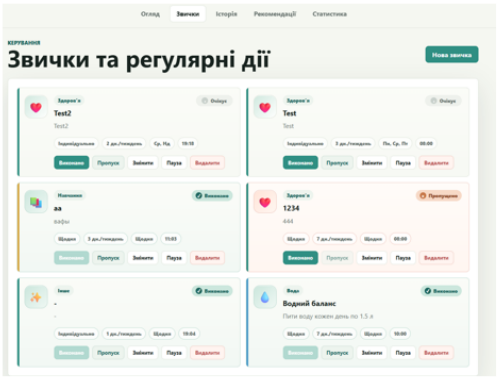


Головна сторінка огляду звичок у HabitFlow

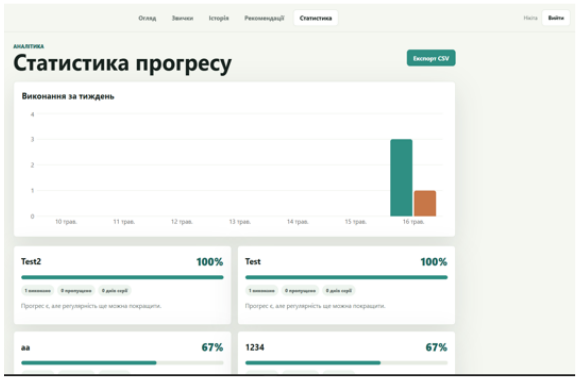


Сторінка історії виконання звичок

ЕКРАННІ ФОРМИ



Сторінка керування звичками



Сторінка статистики виконання звичок

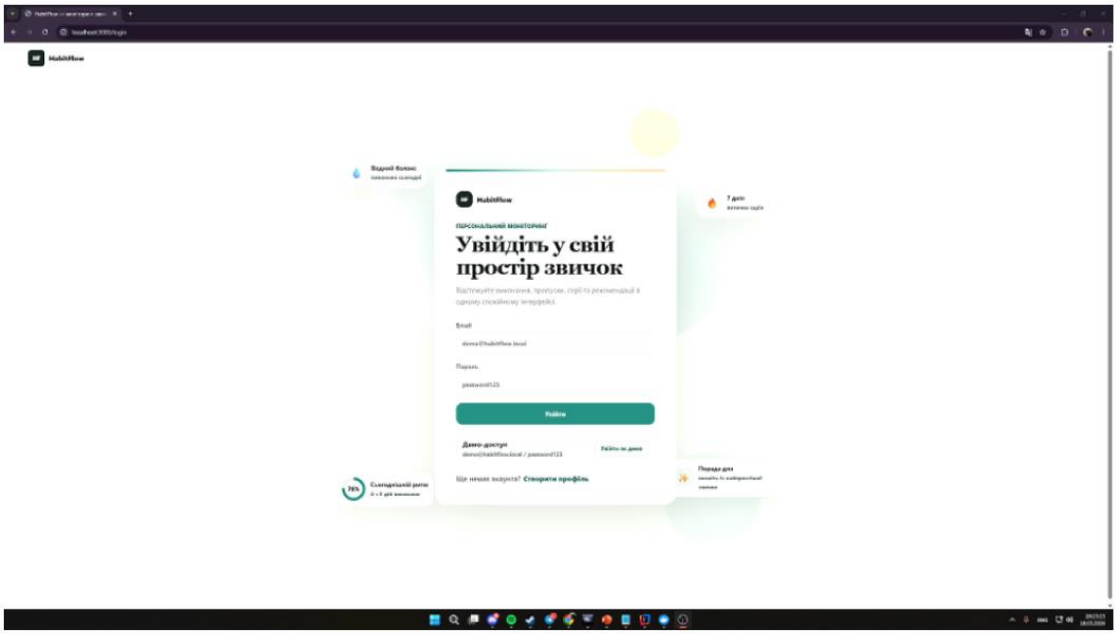
АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Волошин Н.Н., Яскевич В.О. Розробка програмного забезпечення комплексного моніторингу та корекції індивідуальних поведінкових патернів користувача. Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація», 1-3 травня 2026 року, Київ, ДУІКТ. Збірник тез. (подано до друку).

2. Волошин Н.Н., Яскевич В.О. Розробка програмного забезпечення комплексного моніторингу та корекції індивідуальних поведінкових патернів користувача. VI Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2026 року, м. Київ, ДУІКТ. Збірник тез. (подано до друку).

15

ДЕМОНСТРАЦІЯ РОБОТИ HABITFLOW



16

ВИСНОВКИ

1. На основі аналізу предметної галузі та існуючих програмних рішень для відстеження звичок було визначено основні проблеми моніторингу регулярних дій користувача, зокрема недостатню гнучкість налаштування графіків, обмежені можливості аналізу статистики та відсутність механізмів корекції поведінкових патернів.
2. На основі проведеного аналізу було сформовано функціональні та нефункціональні вимоги до вебзастосунку HabitFlow, а також обґрунтовано використання технологічного стеку React, Spring Boot, PostgreSQL, Spring Security та JWT для реалізації клієнт-серверної архітектури.
3. У результаті проєктування було розроблено модульну архітектуру системи, структуру бази даних, основні сутності та алгоритми роботи застосунку, що забезпечують збереження історії виконання звичок, побудову статистики та формування поведінкових метрик.
4. У процесі реалізації було створено функціональні модулі автентифікації, керування звичками, журналу виконання, статистики, рекомендацій, нагадувань та CSV-експорту, що дозволило забезпечити комплексний моніторинг і аналіз регулярних дій користувача.
5. Проведене тестування основних сценаріїв роботи підтвердило коректність функціонування програмного продукту, стабільність взаємодії між frontend та backend частинами, а також працездатність механізмів авторизації, роботи з базою даних і формування статистики.

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ ПРОГРАМНИХ МОДУЛІВ

```
package com.example.habitflow.security;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import
jakarta.servlet.http.HttpServletRequest;
import
jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import
org.springframework.security.authenticatio
n.UsernamePasswordAuthenticationToken;
import
org.springframework.security.core.context.
SecurityContextHolder;
import
org.springframework.security.core.userdeta
ils.UserDetails;
import
org.springframework.security.web.authentic
ation.WebAuthenticationDetailsSource;
import
org.springframework.stereotype.Component;
import
org.springframework.web.filter.OncePerRequ
estFilter;

import java.io.IOException;

@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter
extends OncePerRequestFilter {

    private final JwtTokenProvider
tokenProvider;
    private final UserDetailsServiceImpl
userDetailsService;

    @Override
    protected void
doFilterInternal(HttpServletRequest
request,

HttpServletResponse response,

FilterChain chain)
            throws ServletException,
IOException {

        String header =
request.getHeader("Authorization");

        if (header == null ||
!header.startsWith("Bearer ")) {
            chain.doFilter(request,
response);
            return;
        }

        String token =
header.substring(7);

        if
(tokenProvider.validateToken(token)) {
            String email =
tokenProvider.getEmailFromToken(token);
            UserDetails userDetails =
userDetailsService.loadUserByUsername(emai
l);


UsernamePasswordAuthenticationToken auth =
                    new
UsernamePasswordAuthenticationToken(
                            userDetails,
                            null,

userDetails.getAuthorities()
                    );

            auth.setDetails(
                    new
WebAuthenticationDetailsSource().buildDeta
ils(request)
            );


SecurityContextHolder.getContext().setAuth
entication(auth);
        }

        chain.doFilter(request, response);
    }
}
```

```
package com.example.habitflow.service;

import com.example.habitflow.entity.Habit;
import
com.example.habitflow.entity.HabitLog;
import com.example.habitflow.entity.User;
import
com.example.habitflow.entity.enums.HabitLo
gStatus;
import
com.example.habitflow.exception.ResourceNo
tFoundException;
import
com.example.habitflow.repository.HabitLogR
epository;
import
com.example.habitflow.repository.HabitRepo
sitory;
import
com.example.habitflow.repository.UserRepos
itory;
import lombok.RequiredArgsConstructor;
import
org.springframework.scheduling.annotation.
Scheduled;
import
org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation
.Transactional;

import java.time.DayOfWeek;
import java.time.LocalDate;
import java.time.ZoneId;
import java.util.Arrays;
import java.util.List;

@Service
@RequiredArgsConstructor
public class AutoSkipService {

    private static final ZoneId KYIV_ZONE
= ZoneId.of("Europe/Kyiv");

    private final UserRepository
userRepository;
    private final HabitRepository
habitRepository;
    private final HabitLogRepository
habitLogRepository;

    @Scheduled(cron = "0 59 23 * * *",
zone = "Europe/Kyiv")
    @Transactional
    public void autoSkipForAllUsers() {
        LocalDate today =
LocalDate.now(KYIV_ZONE);

        List<Habit> scheduledHabits =
habitRepository.findAll()
                .stream()
                .filter(Habit::isActive)
                .filter(habit ->
isScheduledForDate(habit, today))
                .toList();

        createSkippedLogs(scheduledHabits,
today);
    }

    @Transactional
    public int autoSkipForUser(String
email, LocalDate date) {
        User user =
userRepository.findByEmail(email)
                .orElseThrow(() ->
                        new
ResourceNotFoundException("Користувача не
знайдено")
                );

        LocalDate targetDate = date ==
null ? LocalDate.now(KYIV_ZONE) : date;

        List<Habit> scheduledHabits =

habitRepository.findByUserAndActiveTrueOrd
erByCreatedAtDesc(user)
                        .stream()
                        .filter(habit ->
isScheduledForDate(habit, targetDate))
                        .toList();

        return
createSkippedLogs(scheduledHabits,
targetDate);
    }

    private int
createSkippedLogs(List<Habit> habits,
LocalDate date) {
        int createdCount = 0;

        for (Habit habit : habits) {
            boolean alreadyHasLog =

habitLogRepository.findByHabitAndLogDate(h
abit, date)
                            .isPresent();

            if (alreadyHasLog) {
                continue;
            }

            HabitLog log =
HabitLog.builder()
                    .habit(habit)
                    .logDate(date)

.status(HabitLogStatus.SKIPPED)
                    .note("Пропуск
автоматично зафіксовано системою")
                    .build();

            habitLogRepository.save(log);
            createdCount++;
        }

        return createdCount;
    }

    private boolean
isScheduledForDate(Habit habit, LocalDate
date) {
        String daysOfWeek =
habit.getDaysOfWeek();

        if (daysOfWeek == null ||
daysOfWeek.isBlank()) {
            return true;
        }
```

```
        DayOfWeek currentDay =
date.getDayOfWeek();

        return
Arrays.stream(daysOfWeek.split(","))
                .map(String::trim)
                .anyMatch(value ->
value.equalsIgnoreCase(currentDay.name()))
;
    }
}
```

```
import axios from 'axios'
import { TOKEN_KEY } from './token'

export const API_BASE_URL =
  import.meta.env.VITE_API_URL ||
'http://localhost:8080/api'

const api = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'application/json',
  },
})

api.interceptors.request.use((config) => {
  const token =
localStorage.getItem(TOKEN_KEY)

  if (token) {
    config.headers.Authorization = `Bearer
${token}`
  }

  return config
})

api.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      window.dispatchEvent(new
Event('auth:unauthorized'))
    }

    return Promise.reject(error)
  },
)

export function getErrorMessage(error) {
  return (
    error.response?.data?.detail ||
    error.response?.data?.message ||
    'Не вдалося виконати запит. Перевірте
підключення до сервера.'
  )
}

export default api
```

```
import { Navigate } from 'react-router-dom'
import { useAuth } from '../context/AuthContext'

export default function PrivateRoute({ children }) {
  const { authReady, isAuthenticated } = useAuth()

  if (!authReady) {
    return (
      <div className="loading-screen">
        <div className="loader" />
      </div>
    )
  }

  if (!isAuthenticated) {
    return <Navigate to="/login" replace />
  }

  return children
}
```