НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Мобільний застосунок для відстеження особистих звичок та подолання залежностей

Виконав студент IV курсу, групи ІП-11
(шифр групи)

Печковський Олександр Костянтинович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник ст.викл., д-р філософії Дифучин А.Ю. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант доцент, к.т.н., доц., Ліщук К. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент асистент, д-р філософії Нікітін В.А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Печковському Олександру Костянтиновичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Мобільний застосунок для відстеження особистих звичок та подолання залежностей

керівник проєкту Дифучин Антон Юрійович, д-р філософії
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «23» травня 2025 р. №1705-с

2. Термін подання студентом проєкту «16» червня 2025 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі.
2) Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних.
3) Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення.
4) Технологічний розділ: керівництво користувача, методика випробувань програмного продукту.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використання_____

2) Схема структурна діяльності_____

3) Креслення вигляду екранних форм_____

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «15» березня 2025 року_____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	15.03.2025	
2	Аналіз існуючих методів розв’язання задачі	30.03.2025	
3	Постановка та формалізація задачі	02.04.2025	
4	Розробка інформаційного забезпечення	09.04.2025	
5	Алгоритмізація задачі	20.04.2025	
6	Обґрунтування вибору використаних технічних засобів	22.04.2025	
7	Розробка програмного забезпечення	20.05.2025	
8	Налагодження програми	25.05.2025	
9	Виконання графічних документів	27.05.2025	
10	Оформлення пояснювальної записки	01.06.2025	
11	Подання ДП на попередній захист	02.06.2025	
12	Подання ДП рецензенту	10.06.2025	
13	Подання ДП на основний захист	16.06.2025	

Студент _____ Олександр ПЕЧКОВСЬКИЙ

(підпис) (ініціали, прізвище)

Керівник _____ Антон ДИФУЧИН

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 43 таблиці, 23 рисунки та 33 джерела – загалом 94 сторінки.

Дипломний проєкт присвячений розробці мобільного застосунку для відстеження особистих звичок та подолання залежностей HabitTracker для платформи Android, який забезпечує відстеження особистих звичок і подолання залежностей із застосуванням гнучких цільових періодів, офлайн-підтримки, соціальних функцій та мотиваційних інструментів.

Мета роботи – спрощення та прискорення процесу формування позитивних звичок і боротьби із залежностями шляхом надання персоналізованих інструментів, соціальної взаємодії та доступності для широкої аудиторії, включаючи користувачів без глибоких знань у сфері саморозвитку.

У першому розділі розглянуто предметну область, проведено аналіз існуючих рішень, моделювання бізнес-процесів за нотацією BPMN та обґрунтовано актуальність проєкту.

Другий розділ присвячений розробленню функціональних і нефункціональних вимог, аналізу системних вимог та економічних показників за методом Use Case Points.

У третьому розділі описано архітектуру програмного забезпечення (клієнт-сервер, монолітна архітектура), обґрунтовано вибір засобів розробки (Java, Android Studio, Firebase) та реалізовано структури даних і бази даних.

Четвертий розділ охоплює аналіз якості за допомогою SonarQube Cloud та мануальне тестування, що підтвердило відповідність вимогам.

П’ятий розділ описує розгортання через GitHub Releases та супровід програмного забезпечення.

Програмне забезпечення впроваджено у вигляді прототипу, доступного для тестування через GitHub Releases, із перспективою публікації в Google Play Store.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ДОДАТОК, ANDROID, ANDROID STUDIO, GOOGLE FIREBASE, БАЗА ДАНИХ, ЗВИЧКИ, ЗАЛЕЖНОСТІ, СОЦІАЛЬНІ ФУНКЦІЇ, ОФЛАЙН-ПІДТРИМКА.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 43 tables, 23 figures, and 33 sources - in total 94 pages.

The diploma project is dedicated to the development of the HabitTracker mobile application for the Android platform, which facilitates the tracking of personal habits and overcoming addictions through the use of flexible target periods, offline support, social features, and motivational tools.

The purpose of the project is to simplify and accelerate the process of forming positive habits and combating addictions by providing personalized tools, social interaction, and accessibility for a wide audience, including users without in-depth knowledge in the field of self-development.

The first section examines the subject area, conducts an analysis of existing solutions, models business processes using BPMN notation, and justifies the project's relevance.

The second section is devoted to the development of functional and non-functional requirements, analysis of system requirements, and economic indicators using the Use Case Points method.

The third section describes the software architecture (client-server, monolithic architecture), justifies the choice of development tools (Java, Android Studio, Firebase), and implements data structures and databases.

The fourth section covers quality analysis using SonarQube Cloud and manual testing, which confirmed compliance with the requirements.

The fifth section describes deployment via GitHub Releases and software maintenance.

The software has been implemented as a prototype, available for testing through GitHub Releases, with the prospect of publication on the Google Play Store.

KEYWORDS: MOBILE APPLICATION, ANDROID, ANDROID STUDIO, GOOGLE FIREBASE, DATABASE, HABITS, ADDICTIONS, SOCIAL FEATURES, OFFLINE SUPPORT.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-1120.045490.01.91	Технічне завдання	22	
3	А4	КПІ.ІП-1120.045490.02.81	Пояснювальна записка	94	
4	А4	КПІ.ІП-1120.045490.03.12	Текст програми	30	
5	А4	КПІ.ІП-1120.045490.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-1120.045490.05.34	Керівництво користувача	18	
7	А3	КПІ.ІП-1120.045490.06.99	Графічний матеріал	4	

КПІ.ІП-1120.045490.00.90

Змін.	Арк.	№ докум.	Підп.	Дата
Розроб.		Печковський О.К.		
Перевір.		Дифучин А.Ю.		
Н.контр.		Ліщук К.І.		
Затв.		Жаріков Е.В.		

Відомість дипломного проєкту

Літ.	Аркуш	Аркушів
	1	

КПІ ім. Ігоря Сікорського
ФІОТ каф. ІПІ гр. ІП-11

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ВІДСТЕЖЕННЯ ОСОБИСТИХ ЗВИЧОК ТА ПОДОЛАННЯ ЗАЛЕЖНОСТЕЙ

Технічне завдання

КПІ.ІП-1120.045490.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Антон ДИФУЧИН

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Олександр ПЕЧКОВСЬКИЙ

Київ – 2025

Зміст

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ВІДСТЕЖЕННЯ ОСОБИСТИХ ЗВИЧОК ТА ПОДОЛАННЯ ЗАЛЕЖНОСТЕЙ.

Галузь застосування:

Наведене технічне завдання поширюється на розробку мобільного застосунку для відстеження особистих звичок та подолання залежностей Habit Tracker, котра використовується для підтримки користувачів у формуванні позитивних звичок, відстеженні прогресу їх виконання та подоланні шкідливих залежностей шляхом моніторингу, мотивації та взаємодії зі спільнотою. Застосунок призначений для галузі особистого розвитку, здоров’я та психологічного благополуччя, а його можливими користувачами є особи, які прагнуть покращити свій спосіб життя, включаючи людей із залежностями, що шукають підтримки, а також усіх, хто цікавиться саморозвитком і самодисципліною.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки Habit Tracker є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для відстеження особистих звичок і подолання залежностей шляхом моніторингу прогресу, встановлення цілей, надання аналітичних звітів, надсилання нагадувань та підтримки взаємодії зі спільнотою через групові чати та чати з партнерами підтримки. Експлуатаційне призначення полягає в наданні користувачам зручного інструменту для саморозвитку, який працює на мобільних пристроях під управлінням Android, підтримує офлайн-режим із синхронізацією даних через Firebase та забезпечує інтуїтивний інтерфейс для щоденного використання.

Метою розробки є створення високоякісного програмного продукту, який характеризується надійністю, зручністю використання та ефективністю. Показники якості включають стабільність роботи, безпеку зберігання даних, швидкість відгуку інтерфейсу та швидкість синхронізації даних.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функцій:

4.1.1 Користувацького інтерфейсу

– відображення головного екрану з узагальненою статистикою звичок і залежностей, включаючи середній прогрес, найкращу звичку, мотиваційні цитати та список цілей (Рисунок 4.1);

– навігація між вкладками (Головна, Звички, Залежності, Повідомлення, Налаштування) через нижню панель навігації (Рисунок 4.1);

– додавання звичок, залежностей і цілей, а також зміна частоти нагадувань для звичок через діалогові вікна з вибором попередньо визначених або власних елементів (Рисунки 4.2, 4.3 і 4.4);

– візуалізація статистики звичок і залежностей у вигляді календарів із позначками подій, гістограм за днями тижня та годинами дня (Рисунок 4.5);

– інтерактивний чат для спілкування з партнерами підтримки або учасниками глобальних чатів із підтримкою копіювання повідомлень та надсилання адміністратору скарг на них (Рисунок 4.6);

– відображення анімації “Панічної кнопки” для мотиваційної підтримки при боротьбі із залежностями, із циклічними повідомленнями (Рисунок 4.7);

– діалогове вікно з налаштуваннями (Рисунок 4.8);

– картки для звичок та залежностей з індивідуальними опціями кожної (Рисунки 4.9 і 4.10).

Рисунок 4.1 - Головний екран з узагальненою статистикою звичок, залежностей та списком цілей

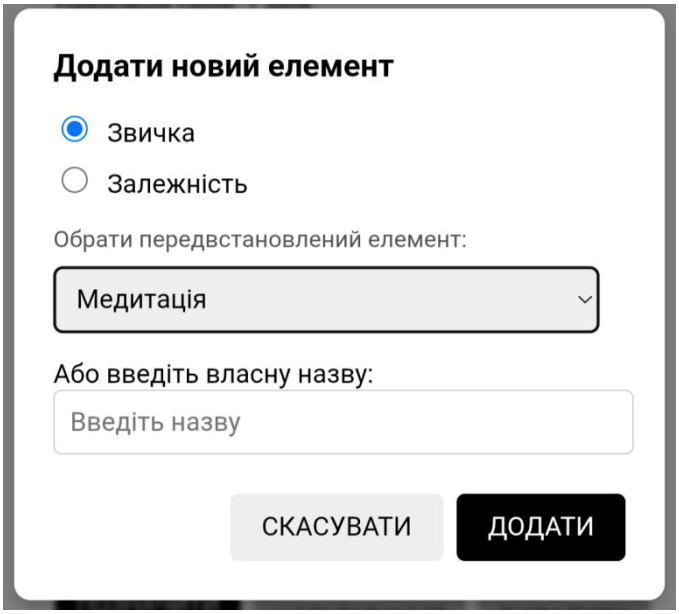


Рисунок 4.2 - Діалогове вікно для додавання звичок та залежностей

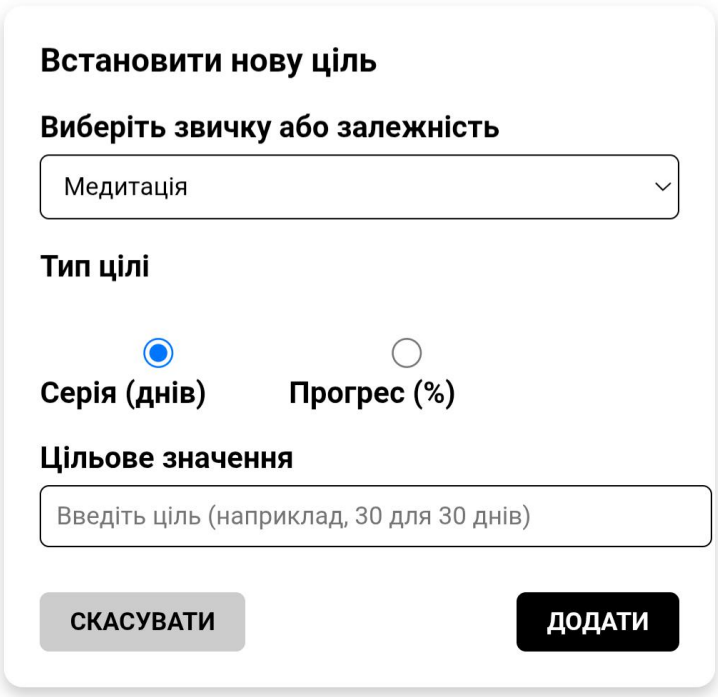


Рисунок 4.3 - Діалогове вікно для додавання цілей

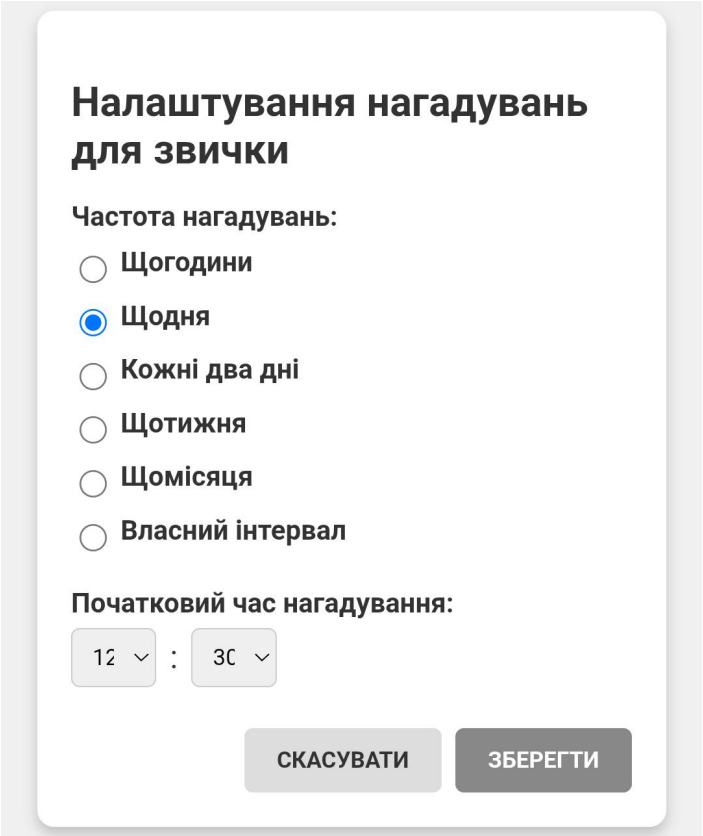


Рисунок 4.4 - Діалогове вікно для налаштування частоти нагадувань для звички

Рисунок 4.5 - Візуалізація статистики звичок і залежностей

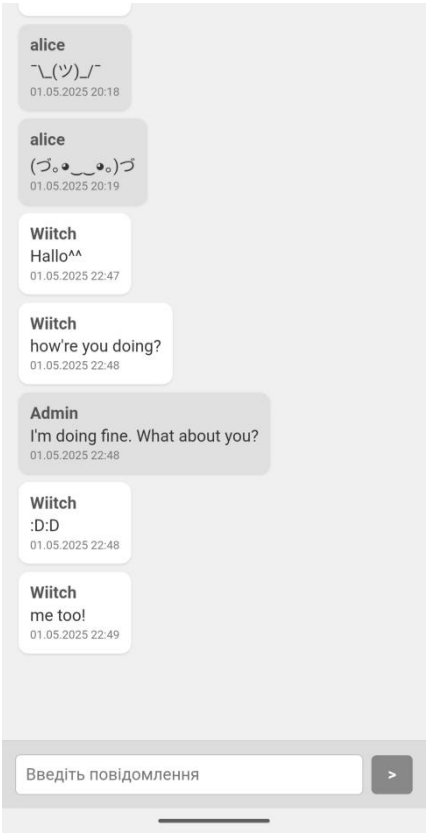


Рисунок 4.6 - Глобальний чат для спілкування з іншими учасниками.

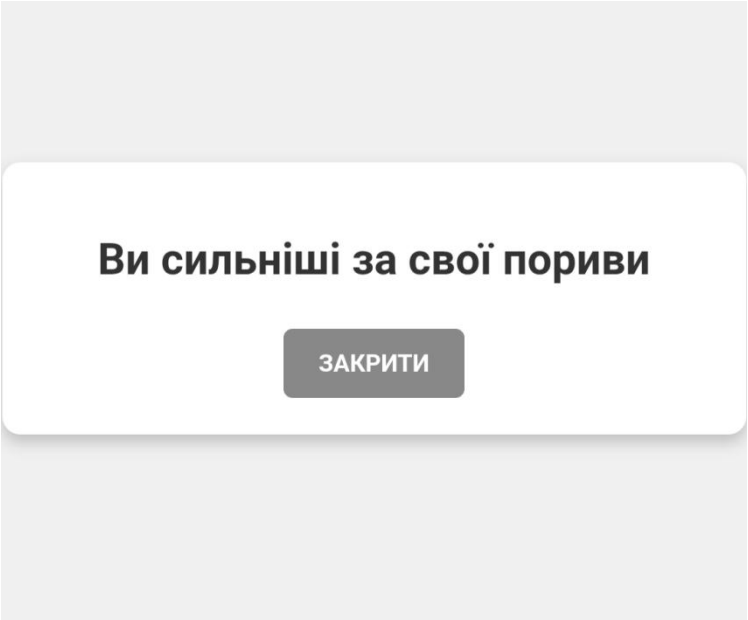


Рисунок 4.7 - Анімація “Панічної кнопки”

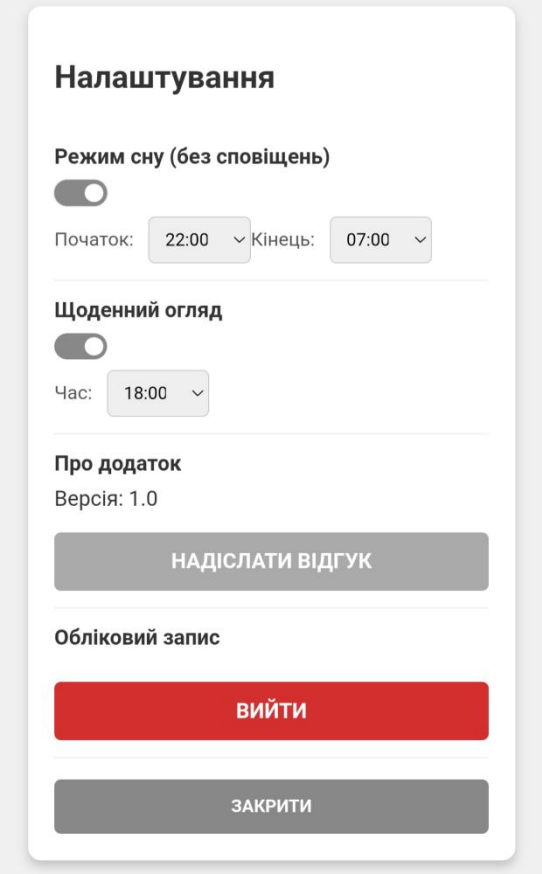


Рисунок 4.8 - Діалогове вікно налаштувань

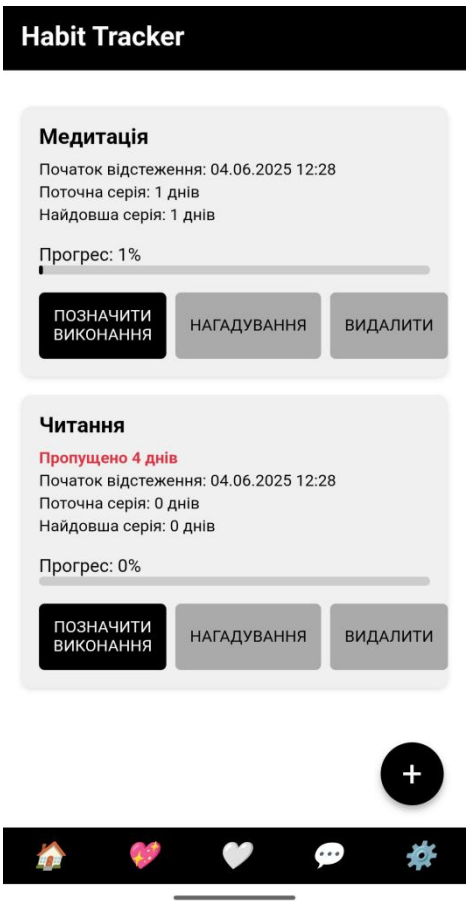


Рисунок 4.9 - Картки для звичок з індивідуальними опціями для кожної

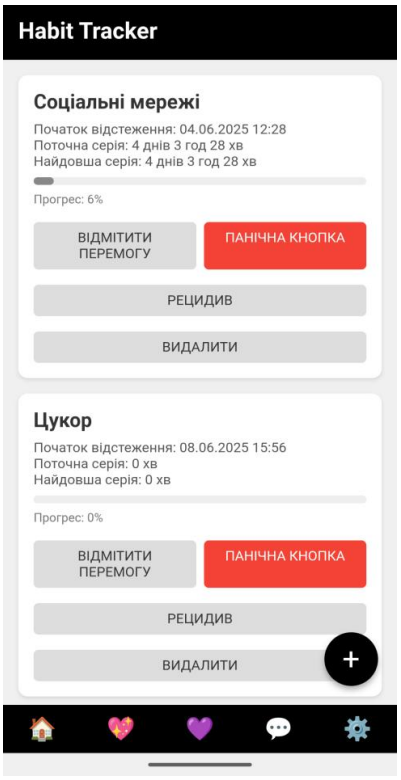


Рисунок 4.10 - Картки для залежностей з індивідуальними опціями для кожної.

4.1.2 Управління звичками та залежностями

4.1.2.1 Для користувача

– додавання/видалення звичок і залежностей через діалогове вікно;

– позначення виконання звички через кнопку “Mark as completed” на картці звички, що оновлює прогрес і серію;

– позначення перемоги над залежністю через кнопку “Mark Victory”, що фіксує день без рецидиву;

– налаштування нагадувань із вибором частоти і початкового часу.

4.1.3 Відстеження рецидивів залежностей

4.1.3.1 Для користувача

– реєстрація рецидиву залежності через діалогове вікно із вибором дати і часу, із валідацією на минулий або теперішній час;

– перегляд історії рецидивів на екрані статистики залежності, відображеної як позначки на календарі.

4.1.4 Встановлення та відстеження цілей

4.1.4.1 Для користувача

– створення цілей через діалогове вікно з вибором звички/залежності, типу мети (streak/progress) та цільового значення (наприклад, 30 днів або 80%);

– перегляд статусу цілей (досягнута/в прогресі) на головному екрані у вигляді карток із описом і датою встановлення;

– видалення цілей через кнопку “Delete” на картці цілі.

4.1.5 Аналіз прогресу

4.1.5.1 Для користувача

– перегляд календаря з позначками виконання звичок або перемог/рецидивів залежностей, із можливістю вибору дня для детальної інформації;

– аналіз розподілу активності через гістограми за днями тижня та годинами дня на екрані статистики;

– отримання текстових даних про поточну та найдовшу серію, прогрес у відсотках;

– перемикання між місяцями для перегляду історичних даних через кнопки “Назад”/“Далі”.

4.1.6 Взаємодія зі спільнотою через чати

4.1.6.1 Для користувача

– створення запитів на партнера підтримки через діалогове вікно із вибором звички/залежності;

– отримання повідомлень у реальному часі в чатах (глобальних або партнерських) із відображенням повідомлення як картки із іменем, текстом і часом;

– відправка повідомлень через поле для введення тексту та кнопку відправлення, із автоматичним прокручуванням донизу;

– отримання push-сповіщень для чатів із партнерами підтримки через FCM, із переходом до чату при натисканні;

– копіювання повідомлень або скарга на повідомлення через контекстне меню із опціями “Copy” та “Report”.

4.1.6.2 Для адміністратора

– створення глобальних чатів через діалог із полем для введення тексту для назви, вибором гендерних обмежень, вибором вікових діапазонів.

4.1.7 Налаштування

4.1.7.1 Для користувача

– увімкнення/вимкнення режиму сну із вибором часу початку та закінчення;

– увімкнення щоденного огляду залежностей та вибір часу, коли будуть надсилатись сповіщення про відмічання перемог над залежностями за поточний день;

– вихід із облікового запису через кнопку “Logout” у налаштуваннях.

4.1.8 Мотиваційна підтримка при залежностях

4.1.8.1 Для користувача

– активація “Панічної кнопки” через кнопку на картці залежності, що відкриває анімацію із мотиваційними повідомленнями;

– закриття діалогу через кнопку “Close” для повернення до екрану залежностей.

4.1.9 Авторизація та ініціалізація профілю

4.1.9.1 Для користувача

– вхід через Google Sign-In із відображенням кнопки на екрані авторизації;

– введення особистих даних (ім’я, вік, стать) після першого входу, із валідацією;

– вибір передвстановлених та/або додавання власних звичок та/або залежностей;

– задання імені користувача для спільноти.

4.1.9.2 Для адміністратора

– позначення профілю як адміністраторського через ім’я користувача Admin, із доступом до адмін-функцій після авторизації.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на пристроях на базі ОС Android.

Мінімальна конфігурація технічних засобів:

– тип процесора: Quad-Core ARM Cortex-A53;
– операційна система: Android 7.0 (API 24);
– об‘єм ОЗП: 2 Гб;
– вільне місце на накопичувачі: 100 МБ;
– підключення до мережі Інтернет зі швидкістю від 5 мегабіт/с (для первинного налаштування та подальшої синхронізації даних);
– екран: роздільна здатність 720x1280 пікселів.

Рекомендована конфігурація технічних засобів:

– тип процесора: Octa-Core ARM Cortex-A73;
– операційна система: Android 11.0 (API 30);
– об‘єм ОЗП: 4 Гб;
– вільне місце на накопичувачі: 500 МБ;
– підключення до мережі Інтернет зі швидкістю від 20 мегабіт/с (для первинного налаштування та подальшої синхронізації даних);
– екран: роздільна здатність 1080x1920 пікселів.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення працює під управлінням операційних систем Android (версії 7.0 і вище).

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в наступному форматі:

– ім’я користувача: текстовий рядок, довжина до 255 символів, обов’язкове поле;

– вік користувача: ціле число, діапазон 14–90, обов’язкове поле;

– стать: попередньо визначене значення (“male”, “female”), обов’язкове поле;

– назва користувацької звички/залежності: текстовий рядок, UTF-8, довжина до 255 символів, обов’язкове поле;

– тип (звичка/залежність): попередньо визначене значення (“habit”, “addiction”), обов’язкове поле;

– час нагадування: дата/час у форматі “yyyy-MM-dd HH:mm”;

– частота нагадувань: попередньо визначене значення (e.g., “hourly”, “daily”, “weekly”) або ціле число хвилин для власного інтервалу;

– позначення виконання звички: мітка часу у форматі “yyyy-MM-dd HH:mm”, обов’язкове поле при натисканні “Mark as completed”;

– реєстрація рецидиву залежності: дата/час у форматі “yyyy-MM-dd HH:mm”, обов’язкове поле;

– звичка/залежність для цілі: ключ (наприклад, “healthy_sleep”, “custom_habit_123”), обов’язкове поле;

– тип цілі: попередньо визначене значення (“streak”, “progress”), обов’язкове поле;

– цільове значення: ціле число (наприклад, 30 днів для streak, 80% для progress), обов’язкове поле;

– повідомлення чату: текстовий рядок;

– мітка часу повідомлення: ціле число, генерується системою, обов’язкове поле;

– режим сну: булеве значення (true/false);

– час початку/закінчення режиму сну: час у форматі “HH:mm”, обов’язкове при увімкненому режимі;

– щоденний огляд залежностей: булеве значення (true/false), необов’язкове поле.

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в наступному форматі:

– прогрес: ціле число (0–100%), відображає відсоток досягнення цільового періоду;

– поточна серія: ціле число (дні для звичок) або текстовий рядок (наприклад, “30 days 5 hours”) для залежностей;

– найдовша серія: ціле число (дні для звичок) або текстовий рядок (наприклад, “30 days 5 hours”) для залежностей;

– позначки календаря: список дат у форматі “yyyy-MM-dd” для виконань, перемог, рецидивів;

– статус цілі: булеве значення (true для досягнутої, false для цілі в прогресі), обчислюється порівнянням прогресу/серії з цільовим значенням.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування Java.

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі Android Studio із Gradle як системою збирання.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді структурованих Java-файлів, організованих у пакеті com.alexpechkin.habittracker.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати інсталяційну версію програмного забезпечення у форматі APK для встановлення на Android-пристрої та AAB для публікації в Google Play Store.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;
– схема структурна діяльності;
– креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проєкту	21.02	
2.	Розробка технічного завдання	15.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.04	Специфікації програмного забезпечення
4.	Проєктування структури програмного забезпечення, проєктування компонентів	30.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проєкту	14.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проєкту	20.05	Графічний матеріал проєкту
9.	Оформлення технічної документації проєкту	29.05	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: **Мобільний застосунок для відстеження особистих звичок та подолання залежностей**

КПІ.ІП-1120.045490.02.81

Київ – 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment, інтегроване середовище розробки.
API	–	Application programming interface, прикладний програмний інтерфейс.
SDK	–	Software development kit.
IT	–	Інформаційні технології.
ER	–	Entity-Relation diagram.
ОС	–	Операційна система.
БД	–	База даних.
BPMN	–	Business Process Model and Notation, нотація моделювання бізнес-процесів.
NoSQL	–	Non-relational SQL, нереляційна база даних.
UI	–	User Interface, користувацький інтерфейс.
OAuth	–	Open Authorization, протокол автентифікації для безпечного доступу.
FCM	–	Firebase Cloud Messaging, система push-сповіщень для мобільних додатків
MVC	–	Model-View-Controller, архітектурний шаблон для організації коду.

ВСТУП

Актуальність тематики дипломного проекту зумовлена зростаючим інтересом до саморозвитку, ментального здоров’я та цифрових інструментів, які допомагають формувати позитивні звички та долати залежності. У сучасному світі люди дедалі частіше стикаються з викликами, пов’язаними з організацією часу, підтримкою мотивації та боротьбою з компульсивними поведінками. Ручне відстеження звичок і залежностей є трудомістким, схильним до пропусків і часто не забезпечує достатньої мотивації для довгострокових змін. Автоматизація цього процесу за допомогою мобільних додатків дозволяє користувачам ефективно керувати своїм прогресом, отримувати персоналізовану підтримку та соціальну взаємодію, що значно підвищує успішність поведінкових змін.

Серед провідних тенденцій розв’язання проблеми виділяються гейміфікація, інтеграція з носимими пристроями, використання штучного інтелекту для персоналізованих рекомендацій та соціальні функції для підвищення мотивації. Сучасні дослідження зосереджені на адаптивних алгоритмах відстеження, психологічних інструментах (наприклад, когнітивно-поведінковій терапії) та забезпеченні офлайн-доступу для користувачів із обмеженим інтернет-з’єднанням. Провідні компанії, такі як Habitica, Streaks та I Am Sober, активно розвивають мобільні додатки, які частково вирішують ці завдання, однак мають обмеження, зокрема недостатню персоналізацію, залежність від інтернету та слабку підтримку складних залежностей.

Станом на сьогодні існують рішення, які дозволяють відстежувати звички чи залежності, але більшість із них не поєднують гнучкі цільові періоди, офлайн-підтримку та соціальні функції в одному додатку. Дослідження у сфері поведінкових змін, проведені такими вченими, як B.J. Fogg (Fogg Behavior Model), підкреслюють важливість мотивації, здатності та тригерів для формування звичок, що враховано в розробці.

У рамках цього проекту буде розроблено мобільний додаток HabitTracker для Android, який забезпечує відстеження звичок і залежностей із гнучкими цільовими періодами (30–90 днів), офлайн-підтримкою, соціальними функціями (чати, партнери підтримки) та режимом паніки для мотивації в кризових ситуаціях. Додаток інтегрується з Firebase для автентифікації, зберігання даних і синхронізації.

Мета роботи – спростити та прискорити процес формування звичок і подолання залежностей, підвищити мотивацію користувачів через персоналізовані інструменти та соціальну взаємодію, а також забезпечити доступність додатка для широкої аудиторії, включаючи користувачів без глибоких знань у сфері саморозвитку.

Можливі сфери застосування включають особистий розвиток, ментальне здоров’я, освіту, корпоративні програми добробуту та психологічну підтримку, де додаток може бути використаний як інструмент для підвищення самодисципліни, мотивації та якості життя.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання дипломного проєктування

Дипломне проєктування передбачає виконання наступних завдань:

– аналіз предметної області та опис її ключових бізнес-процесів, визначення загального завдання розробки у рамках ДП;

– аналіз існуючих рішень (як програмних продуктів, так і алгоритмічних чи технічних рішень) обраного завдання розробки у рамках ДП;

– аналіз та моделювання бізнес-процесів;

– розроблення функціональних, нефункціональних та системних вимог до програмного забезпечення;

– постановка завдання на розробку програмного забезпечення ДП;

– розроблення архітектури програмного забезпечення;

– розроблення архітектурних рішень та обґрунтування вибору засобів розробки програмного забезпечення;

– конструювання та розроблення програмного забезпечення;

– аналіз безпеки даних програмного забезпечення;

– аналіз якості та тестування програмного забезпечення;

– розгортання та супровід програмного забезпечення;

– створення супроводжувальної документації до розробленого програмного забезпечення.

1.2 Аналіз предметної області

Предметна область відстеження звичок та залежностей охоплює процеси формування позитивних поведінкових моделей (звичок) та подолання негативних (залежностей). Звички – це сукупність автоматизованих дій, які виконуються регулярно внаслідок повторення, що сприяють особистісному розвитку та підвищенню якості життя [1]. Наприклад, регулярне читання чи ранкові вправи стають звичками після

певного періоду практики. Залежності, навпаки, є компульсивними поведінками, часто зумовленими фізіологічними чи психологічними факторами, такими як залежність від алкоголю чи соціальних мереж [2]. Формування звичок зазвичай вимагає від 21 до 66 днів, залежно від складності поведінки, тоді як подолання залежностей може займати від 60 до 120 днів через необхідність подолання фізичних і психологічних бар'єрів [3] [4].

Основні напрямки розробок у цій сфері включають:

– мобільні додатки для відстеження звичок: програми, такі як Habitica, Streaks та Way of Life, використовують гейміфікацію, сповіщення та аналітику для мотивації користувачів;

– платформи для боротьби із залежностями: додатки, як-от QuitNow! та I Am Sober, пропонують інструменти для відстеження тверезості та підтримки спільноти;

– інтеграція з носимими пристроями: наприклад, Apple Watch чи Fitbit для автоматичного моніторингу сну, фізичної активності чи стресу;

– психологічні інструменти: вебплатформи та чат-боти, що надають когнітивно-поведінкову терапію (КПТ) для боротьби із залежностями [34].

Ця предметна область набуває популярності завдяки зростанню інтересу до саморозвитку, ментального здоров'я та цифрових технологій, які дозволяють автоматизувати процеси відстеження та надавати персоналізовану підтримку [5]. Теоретичною основою є моделі поведінки, такі як Fogg Behavior Model, яка наголошує на трьох елементах для зміни поведінки: мотивація, здатність та тригери [6].

На поточний момент знання предметної області відстеження звичок та залежностей активно імплементуються в IT через різноманітні програмні рішення. Основні підходи включають:

– мобільні додатки: Використовують інтуїтивні інтерфейси для створення звичок, позначення виконаних дій, відображення прогресу через

графіки та сповіщення для нагадувань. Наприклад, Habitica застосовує RPG-гейміфікацію, де виконання звичок підвищує рівень персонажа [7];

– хмарні сервіси: Платформи, такі як Firebase чи AWS, забезпечують синхронізацію даних між пристроями, зберігання історії прогресу та обробку аналітики;

– штучний інтелект: Алгоритми машинного навчання аналізують поведінку користувачів, пропонуючи персоналізовані рекомендації, наприклад, оптимальний час для виконання звички чи мотиваційні повідомлення [8];

– соціальні функції: Спільноти, чати та партнери по відповідальності (як в I Am Sober) підвищують мотивацію через соціальну взаємодію.

Програмне забезпечення зазвичай реалізує такі функції:

– реєстрація користувачів та налаштування профілів;

– вибір або створення звичок/залежностей із цільовими періодами;

– відстеження прогресу через календарі, лічильники чи відсотки;

– нагадування через push-сповіщення;

– аналітика у вигляді графіків, звітів чи рекомендацій;

– інтеграція з носимими пристроями для автоматичного збору даних (наприклад, сон чи фізична активність).

Бізнес-процеси у сфері відстеження звичок та залежностей включають:

– реєстрація та налаштування профілю: Користувач створює акаунт, вказуючи особисті дані (вік, стать, цілі) для персоналізації досвіду;

– вибір звичок або залежностей: Користувач обирає готові звички (наприклад, “медитація”) чи залежності (наприклад, “відмова від куріння”) або створює власні;

– відстеження прогресу: Щоденне позначення виконаних звичок чи утримання від залежностей, оновлення статистики (строк, прогрес);

– нагадування та мотивація: Надсилання сповіщень, мотиваційних цитат, нагород чи повідомлень від спільноти;

– соціальна взаємодія: Участь у чатах, обмін досвідом із партнерами підтримки чи тематичними групами;

– аналітика та звіти: Генерація графіків прогресу, аналіз шаблонів поведінки, виявлення проблемних зон (наприклад, часті рецидиви залежностей).

Ці процеси підтримуються через мобільні інтерфейси, локальні бази даних (наприклад, SharedPreferences), хмарні сервіси (Firebase) та системи сповіщень (Android AlarmManager).

Незважаючи на значний прогрес, сучасні IT-рішення у сфері відстеження звичок та залежностей мають низку недоліків:

– недостатня персоналізація: більшість додатків застосовують однакові цільові періоди для всіх звичок чи залежностей, ігноруючи їх різну складність (наприклад, формування звички читання vs відмова від алкоголю) [9];

– обмежена офлайн-підтримка: багато додатків, таких як Habitica, потребують постійного інтернет-з’єднання для синхронізації чи повного функціоналу, що створює проблеми для користувачів з нестабільним інтернетом [10];

– слабка підтримка складних залежностей: додатки рідко інтегрують психологічні інструменти, такі як когнітивно-поведінкова терапія, що обмежує їх ефективність для боротьби з наркоманією чи самопошкодженням [11];

– проблеми з безпекою даних: чутливі дані користувачів (наприклад, історія рецидивів) не завжди достатньо захищені, що може призвести до витоків інформації [12];

– низька мотивація для довгострокового використання: одноманітні інтерфейси чи статичні мотиваційні механізми призводять до втрати інтересу користувачів після кількох тижнів [9].

У рамках дипломного проєкту буде розроблено мобільний додаток HabitTracker для Android, який поєднує покращену офлайн-підтримку, гнучку

систему відстеження звичок та залежностей із індивідуальними цільовими періодами та соціальні функції для підвищення мотивації. Унікальною особливістю є гнучкі цільові періоди для досягнення 100% прогресу (30 днів для здорового сну, 66 днів для читання, 90 днів для відмови від алкоголю), які враховують різну складність поведінкових змін. Соціальні функції, такі як групові чати та партнери підтримки, сприяють мотивації через спільноту. Цей підхід вирішує проблеми офлайн-доступу, персоналізації та довгострокової мотивації, забезпечуючи простоту використання та безпеку даних.

1.3 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації мобільного застосунку для відстеження особистих звичок та подолання залежностей Habit Tracker. Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

1.3.1 Аналіз відомих програмних продуктів

У цьому розділі розглядаються готові програмні продукти, які частково або повністю реалізують функціонал відстеження звичок та залежностей, описаний у технічному завданні дипломного проєкту. Для аналізу обрано чотири популярні додатки: Habitica, Streaks, Way of Life та I Am Sober. Ці продукти порівнюються з розробленим додатком HabitTracker, з акцентом на їх функціонал, особливості та обмеження. Кожен опис супроводжується скріншотами, а порівняння представлено у вигляді таблиці.

Habitica — це мобільний та вебдодаток, який використовує гейміфікацію для мотивації користувачів у формуванні звичок та виконанні завдань [13]. Користувачі створюють персонажа, який отримує досвід і золото за виконання звичок, щоденних завдань чи списків справ, але втрачає здоров’я за невиконання. Додаток підтримує соціальні функції, такі як гільдії

та групові квести, що сприяють мотивації через спільноту. Habitica доступна на iOS, Android та вебплатформах, є безкоштовною з преміум-підпискою для додаткових функцій. Вона дозволяє створювати звички, щоденні завдання та списки справ, пропонує аналітику прогресу через статистику виконаних завдань і надсилає нагадування через push-сповіщення. Проте додаток сильно залежить від інтернет-з'єднання, що ускладнює офлайн-використання, а гейміфікація може бути незручною для користувачів, які не люблять ігрові елементи. Крім того, підтримка відстеження залежностей обмежена через відсутність спеціалізованих інструментів для рецидивів [14]. Інтерфейс застосунку Habitica наведений на рисунку 1.1.

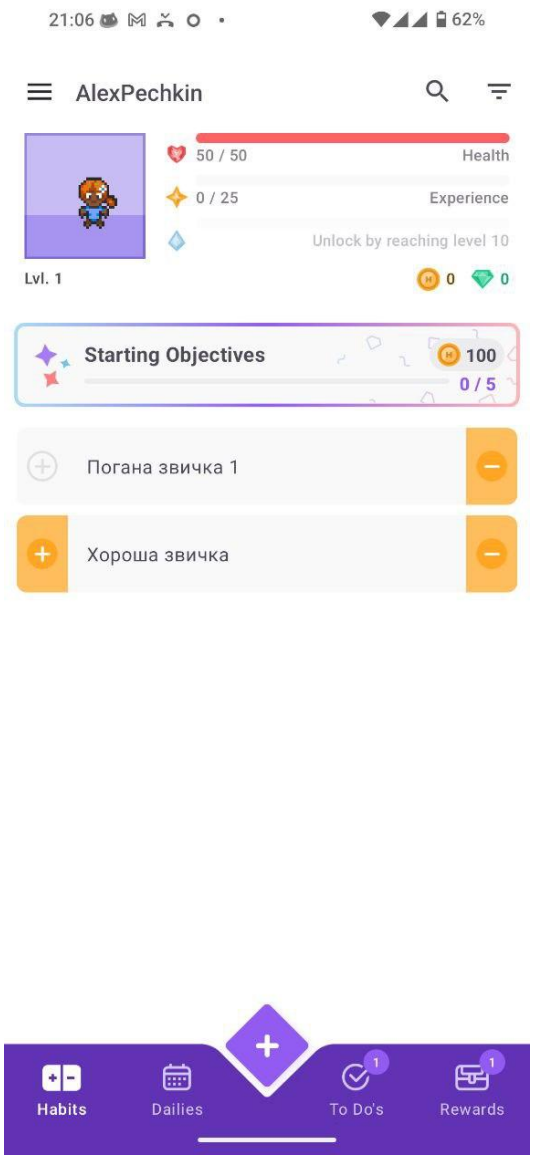


Рисунок 1.1 – Інтерфейс застосунку Habitica для Android

Streaks — це iOS-додаток, орієнтований на формування звичок через підтримку "ланцюжків" (streaks). Користувачі додають звички, позначають їх виконання, а додаток мотивує не переривати серію. Streaks має мінімалістичний дизайн, інтеграцію з Apple Health для автоматичного відстеження, наприклад, сну, а також підтримує віджети. Додаток є платним, доступним за разову покупку. Він дозволяє створювати та відстежувати звички з фокусом на ланцюжки, надавати аналітику у вигляді графіків і статистики, а також надсилати нагадування через сповіщення. Однак Streaks доступний лише на iOS, що обмежує його аудиторію, не підтримує відстеження залежностей і не має соціальних функцій [14]. Інтерфейс застосунку Streaks наведений на рисунку 1.2.

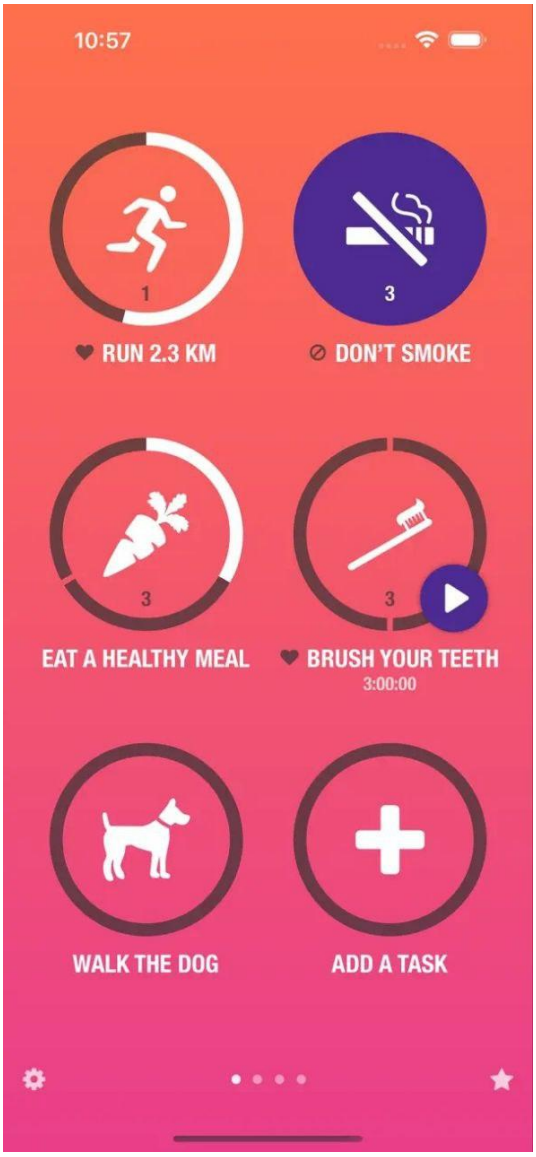


Рисунок 1.2 – Інтерфейс застосунку Streaks для iOS

Way of Life — це додаток для iOS та Android, який допомагає формувати звички через аналіз поведінки та створення графіків. Користувачі додають звички, позначають їх виконання, а додаток виявляє шаблони поведінки, допомагаючи визначити проблемні зони. Безкоштовна версія обмежена за кількістю звичок, а преміум-підписка розширює функціонал, включаючи синхронізацію через хмару. Way of Life пропонує відстеження звичок із графіками прогресу, аналіз шаблонів поведінки та нагадування через сповіщення. Однак підтримка відстеження залежностей обмежена, безкоштовна версія дозволяє відстежувати лише три звички [15]. Інтерфейс застосунку Way of Life наведений на рисунку 1.3.

Рисунок 1.3 – Інтерфейс застосунку Way of Life для Android

I Am Sober — це додаток для iOS та Android, призначений для відстеження тверезості від різних залежностей, таких як алкоголь, наркотики

чи азартні ігри. Він пропонує лічильник тверезості, мотиваційні цитати, спільноту для обміну досвідом і щоденні обіцянки. Безкоштовна версія має базовий функціонал, а преміум розширює можливості, включаючи розширену аналітику. I Am Sober дозволяє відстежувати прогрес через графіки та віхи, але має обмежену підтримку звичок, оскільки фокусується на залежностях. Додаток залежить від інтернету для роботи спільноти та синхронізації, а преміум-функції, такі як розширена аналітика, є платними [16]. Інтерфейс застосунку I Am Sober наведений на рисунку 1.4.

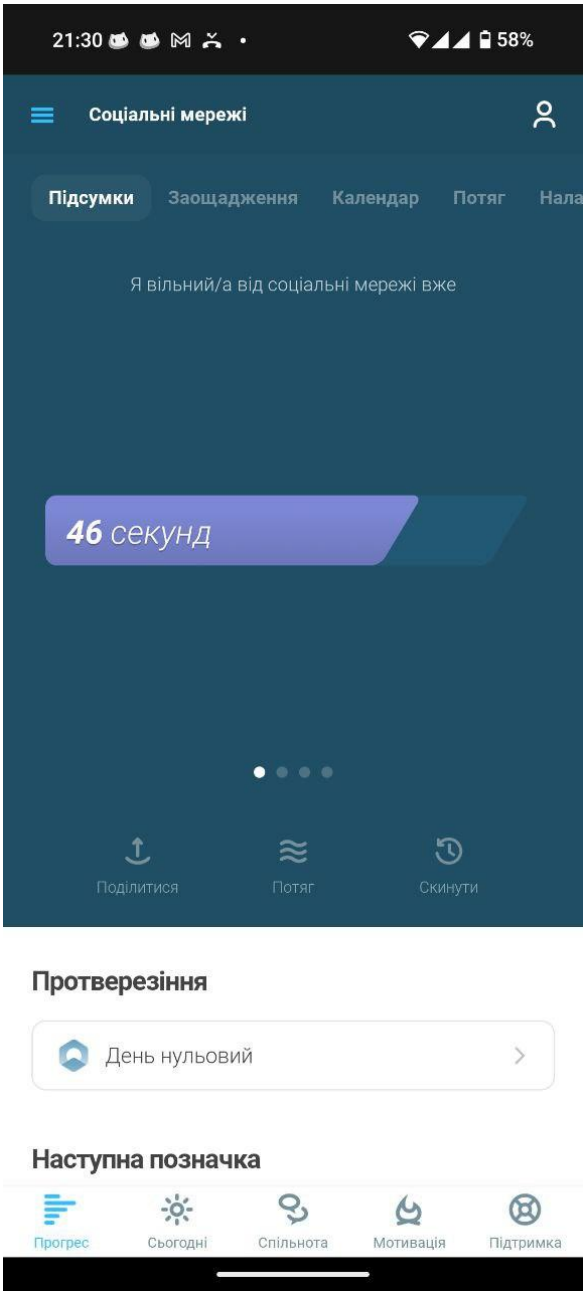


Рисунок 1.4 – Інтерфейс застосунку I Am Sober для Android

HabitTracker вирізняється завдяки гнучким цільовим періодам для звичок і залежностей (наприклад, 30 днів для здорового сну, 90 днів для відмови від алкоголю), що враховує різну складність поведінкових змін. Соціальні функції (групові чати, партнери підтримки) конкурують з Habitica та I Am Sober, але без складної гейміфікації, що робить додаток простішим для широкої аудиторії. Відсутність інтеграції з носимими пристроями та обмеження платформою Android є недоліками порівняно зі Streaks чи Habitica. Однак безкоштовна модель і акцент на універсальність роблять Habit Tracker привабливим рішенням.

Для порівняння проєкту з аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Функціонал	Дипломний проєкт (Habit Tracker)	Аналог для порівняння 1 (Habitica)	Аналог для порівняння 2 (Streaks)	Аналог для порівняння 3 (Way of Life)	Аналог для порівняння 4 (I Am Sober)
Відстеження звичок	Так	Так	Так	Так	Ні
Відстеження залежностей	Так	Ні	Ні	Ні	Так
Гнучкі цільові періоди	Так	Ні	Ні	Ні	Ні
Офлайн-підтримка	Так	Ні	Частково	Ні	Ні
Соціальні функції	Так	Так	Ні	Ні	Так
Гейміфікація	Ні	Так	Так	Ні	Ні
Нагадування	Так	Так	Так	Так	Так
Аналітика прогресу	Так	Так	Так	Так	Так
Інтерграція з носимими пристроями	Ні	Ні	Так	Ні	Ні

Продовження таблиці 1.1

Підтримка Android	Так	Так	Ні	Так	Так
Підтримка iOS	Ні	Так	Так	Так	Так
Вартість	Безкоштовно	Безкоштовно з обмеженнями, для повного функціоналу потрібен преміум	Платно	Безкоштовно з обмеженнями, для повного функціоналу потрібен преміум	Безкоштовно з обмеженнями, для повного функціоналу потрібен преміум

1.3.2 Аналіз відомих алгоритмічних та технічних рішень

У рамках розробки програмного забезпечення для відстеження особистих звичок та подолання залежностей основна функціональність не передбачає використання складних математичних або обчислювальних алгоритмів, що вимагають глибокого теоретичного аналізу або оптимізації. Ключові задачі, такі як ведення статистики прогресу, розрахунок тривалості серій, планування нагадувань та збереження даних, базуються на простих операціях з датою, часом та базовій агрегації інформації. Ці операції плануються до реалізації з використанням стандартних можливостей мови програмування Java та Android SDK, а також готових бібліотечних рішень для роботи з часом та датами. Таким чином, розробка або модифікація існуючих базових алгоритмів для цих задач у межах дипломної роботи не планується.

Проте, важливим аспектом розробки є вибір технічних рішень для створення мобільного застосунку та керування його даними. Розглянемо основні підходи до розробки мобільних застосунків та рішення для зберігання даних.

Нативна розробка передбачає створення окремих версій застосунку для кожної мобільної платформи (Android та iOS) з використанням відповідних мов програмування (Java/Kotlin для Android, Swift/Objective-C для iOS) та інструментів розробки (Android Studio, Xcode). Перевагами такого підходу є максимальна продуктивність, повний доступ до API пристрою та можливості платформи, найкращий користувацький досвід (UX) завдяки використанню нативних UI-компонентів. А недоліками - висока вартість розробки та підтримки через необхідність двох окремих команд розробників та дублювання коду, більший час на розробку нових функцій та оновлень для обох платформ.

Крос-платформна розробка дозволяє розробляти застосунок одночасно для кількох платформ з використанням єдиної кодової бази. Існують різні технології для цього, такі як React Native, Flutter, Xamarin. Перевагами є зниження вартості та часу розробки завдяки спільному використанню коду, спрощення підтримки та оновлень. З недоліків варто відзначити потенційні обмеження у доступі до деяких специфічних API пристрою, можливі проблеми з продуктивністю порівняно з нативними застосунками, залежність від фреймворку та його оновлень, потенційно менш "нативний" вигляд інтерфейсу.

При використанні гібридної розробки застосунки створюються з використанням веб-технологій (HTML, CSS, JavaScript) та обертаються у нативний контейнер (наприклад, Cordova, Ionic). Перевагами є використання знайомих веб-технологій, швидка розробка простих застосунків. Але у цього методу є також значні недоліки: найнижча продуктивність серед усіх підходів, обмежений доступ до API пристрою, значні проблеми з "нативним" виглядом та користувацьким досвідом.

Для порівняння підходів до розробки мобільних застосунків наведено таблицю 1.2 [29].

Таблиця 1.2 – Порівняння підходів до розробки мобільних застосунків

Функціонал	Нативна розробка	Крос-платформна розробка	Гібридна розробка
Продуктивність	Висока	Середня	Низька
Доступ до API	Повний	Обмежений	Значно обмежений
Користувацький досвід	Найкращий	Хороший	Посередній
Вартість розробки	Висока	Середня	Низька
Час розробки	Довгий	Середній	Швидкий
Підтримка	Складна	Простіша	Найпростіша
Спільне використання коду	Ні	Так	Так

Зважаючи на необхідність забезпечення якісного користувацького досвіду, оптимальної продуктивності та повного доступу до можливостей платформи Android, для розробки даного застосунку буде використано нативний підхід з використанням Android Studio для розробки. Це дозволить максимально оптимізувати роботу застосунку та реалізувати всі заплановані функції без обмежень.

Основними мовами програмування для нативної розробки під Android є Java та Kotlin.

Java є однією з основних мов для розробки під Android з моменту його появи. Має велику спільноту розробників, величезну кількість бібліотек та інструментів, а також велику кількість існуючого коду та навчальних матеріалів.

Kotlin - це сучасна, статично типізована мова програмування, розроблена JetBrains, яка є офіційно підтримуваною мовою для Android. Kotlin пропонує більш лаконічний та безпечний синтаксис, підтримку корутин для асинхронності та нульову безпеку на рівні типу.

Порівняння Java та Kotlin для Android-розробки наведено в таблиці 1.3 [30].

Таблиця 1.3 – Порівняння Java та Kotlin для Android-розробки

Критерій	Java	Kotlin
Синтаксис	Багатослівний	Лаконічний
Безпека Null	Потребує уважності	Вбудована на рівні типу
Асинхронність	Threads, AsyncTask, RxJava	Coroutines
Розширюваність	Обмежена	Функції розширення
Взаємодія з Java	Відмінна	Відмінна
Обсяг коду	Більший	Менший
Безпека	Нижча	Вища

Незважаючи на переваги Kotlin, для розробки даного дипломного проекту було обрано Java. Це рішення зумовлено кількома факторами. По-перше, на момент початку розробки автор мав більший досвід роботи саме з Java, що дозволило швидше розпочати роботу та ефективніше вирішувати початкові завдання. По-друге, велика кількість існуючих бібліотек та прикладів коду на Java для Android полегшує пошук рішень для типових задач. По-третє, хоча Kotlin пропонує певні переваги, основна функціональність даного застосунку не вимагає використання складних асинхронних операцій або розширених можливостей мови, де б переваги Kotlin були критично важливими.

Для забезпечення персоналізації даних та можливості використання застосунку на різних пристроях з єдиним обліковим записом необхідно реалізувати систему аутентифікації користувачів [17]. Існує кілька поширених методів.

– власна система аутентифікації: передбачає створення власної бази даних користувачів з логінами та паролями, а також реалізацію логіки реєстрації, входу та відновлення пароля;

– соціальна аутентифікація (OAuth): дозволяє користувачам входити в застосунок, використовуючи їхні облікові записи в інших сервісах, таких як Google, Facebook, Twitter;

– автентифікація через Firebase Authentication: готовий сервіс від Firebase, який підтримує різні методи автентифікації, включаючи електронну пошту/пароль, соціальні мережі, анонімний вхід тощо.

Для даного застосунку було обрано автентифікацію через Google за допомогою Firebase Authentication [18]. Адже більшість користувачів Android вже мають обліковий запис Google, що спрощує процес реєстрації та входу в застосунок. Також Google забезпечує надійний захист облікових записів користувачів. І, що не менш важливо, автентифікація через Firebase Authentication легко інтегрується з іншими можливостями Firebase, такими як Realtime Database, що спрощує розробку backend-частини застосунку.

Дана тема дипломного проєкту передбачає зберігання даних користувачів. Для мобільних застосунків існує кілька основних варіантів зберігання даних. Локальне збереження передбачає збереження даних безпосередньо на пристрої користувача. Для Android наявні декілька варіантів локального збереження:

– SharedPreferences - це простий механізм для зберігання невеликих обсягів даних у форматі ключ-значення;

– внутрішнє/зовнішнє сховище файлів дозволяє зберігати файли різних форматів у приватній або публічній області пам'яті пристрою;

– SQLite - це реляційна база даних, вбудована в Android. Вона підходить для зберігання структурованих даних середнього та великого обсягу;

– Room Persistence Library - це бібліотека від Google, що надає абстракцію над SQLite, спрощуючи роботу з базами даних та забезпечуючи зручний доступ до даних через об'єкти Kotlin/Java.

Для зберігання налаштувань та даних застосунку, доступних офлайн, буде використано SharedPreferences через простоту та зручність використання.

Окрім локального зберігання можливе також віддалене зберігання, при якому дані зберігаються на віддалених серверах та синхронізуються з

пристроями користувачів через мережу. Це можна реалізувати, локально зберігаючи дані на сервері розробника. Цей підхід має переваги та недоліки. З переваг важливо зазначити повний контроль над інфраструктурою бази даних, можливість оптимізації та налаштування сервера під потреби застосунку.

Але недоліків такого рішення значно більше, адже розробнику необхідно самостійно налаштовувати, адмініструвати та підтримувати сервер, операційну систему, програмне забезпечення бази даних, мережеву безпеку та API. Це вимагає значних технічних знань та часу. У випадку зростання кількості користувачів та обсягу даних, розробнику необхідно самостійно піклуватися про масштабування сервера, що може призвести до простоїв та необхідності значних інвестицій у нове обладнання та його налаштування. Також відповідальність за безперебійну роботу сервера повністю лежить на розробнику. У випадку збоїв обладнання, проблем з мережею або помилок у налаштуваннях, доступність застосунку може бути порушена. Щоб уникнути цих проблем, можна використати хмарні сервіси.

Наприклад, Firebase Realtime Database (сервіс Google Cloud Platform) - це NoSQL база даних, що забезпечує синхронізацію даних у реальному часі між клієнтами. Підходить для застосунків, де важлива миттєва передача даних [19].

Враховуючи потребу в синхронізації даних у реальному часі між різними пристроями користувача, а також значно меншу складність розробки та підтримки інфраструктури, наявність готових рішень для синхронізації в реальному часі, автоматичне резервне копіювання, високу масштабованість та надійність, для хмарного зберігання даних застосунку буде використано Firebase Realtime Database.

1.4 Аналіз та моделювання бізнес-процесів

У цьому підрозділі буде розглянуто ключові бізнес-процеси програмного забезпечення «Habit Tracker», призначеного для відстеження

особистих звичок та допомоги у подоланні залежностей. Для моделювання бізнес-процесів використаємо нотацію BPMN (Business Process Model and Notation) [33]. Нижче наведено аналіз та наочне представлення декількох ключових бізнес-процесів ПЗ.

При першому запуску програми користувач має протйи початкове налаштування програми, а саме: реєстрацію, введення своїх даних, вибір та додавання звичок, вибір та додавання залежностей. А при наступних запусках має автоматично відбуватись перехід на головний екран. На рисунку 1.5 наведено модель бізнес-процесу «Запуск програми користувачем».

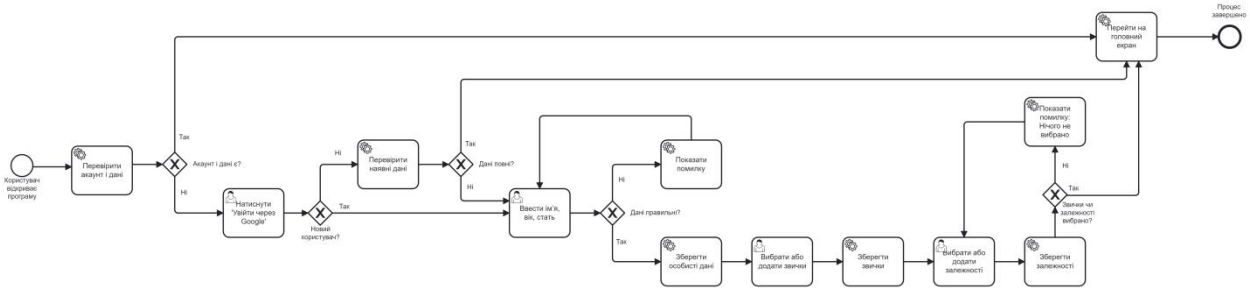


Рисунок 1.5 – Модель бізнес-процесу «Запуск програми користувачем»

Опис моделі бізнес-процесу «Запуск програми користувачем»:

– користувач запускає програму;

– відбувається перевірка акаунту та даних;

– якщо вже було увійдено в акаунт та дані є, відбувається перехід на головний екран і процес завершується;

– якщо не було увійдено в акаунт, то користувач натискає “Увійти через Google”;

– відбувається перевірка, чи це новий акаунт;

– якщо акаунт вже наявний і дані повні, то відбувається перехід на головний екран і процес завершується;

– якщо користувач новий або наявний з неповними даними, то користувач вводить ім’я, вік та стать;

– якщо дані некоректні, показується помилка;

– якщо дані коректні, вони зберігаються;

– далі користувач вибирає та/або додає звички та/або залежності;

– якщо жодної звички чи залежності не додано, то користувач повертається до вибору залежностей;

– інакше відбувається перехід на головний екран і процес завершується.

Одним з фундаментальних процесів для користувача є можливість додавати нові звички, які він хоче розвинути, або залежності, які прагне подолати. На рисунку 1.6 наведено модель бізнес-процесу «Додавання нової звички/залежності».

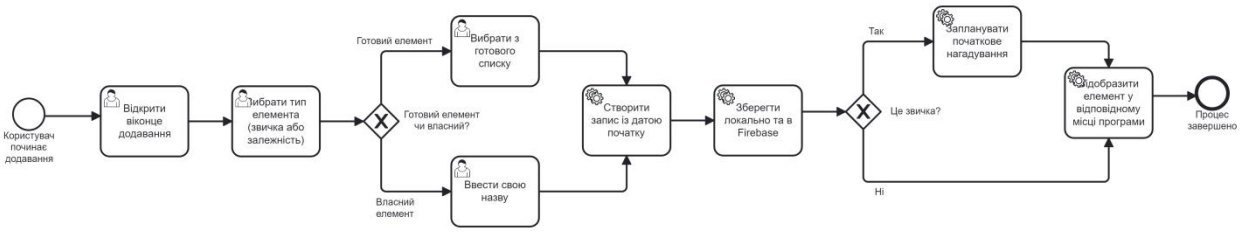


Рисунок 1.6 – Модель бізнес-процесу «Додавання нової звички/залежності»

Опис моделі бізнес-процесу «Додавання нової звички/залежності»:

– користувач запускає процес додавання, обирає тип (звичка/залежність);

– користувач вибирає звичку або залежність з готових елементів або вводить власну назву;

– новий елемент з датою початку зберігається локально та синхронізується з Firebase;

– якщо це звичка, для неї планується початкове нагадування;

– система відображає елемент та інформацію про нього у відповідному місці програми.

Для залежностей критично важливою є функція фіксації рецидивів, що дозволяє користувачеві відстежувати свою боротьбу та аналізувати періоди утримання. На рисунку 1.7 наведено модель бізнес-процесу «Запис рецидиву для залежності».

Рисунок 1.7 – Модель бізнес-процесу «Запис рецидиву для залежності»

Опис моделі бізнес-процесу «Запис рецидиву для залежності»:

– користувач обирає опцію запису рецидиву для конкретної залежності (кнопка на картці залежності) та підтверджує свій намір у діалоговому вікні;

– система надає послідовно інтерфейси для вибору дати та часу, які не можуть бути в майбутньому;

– реєструються дата/час зриву, оновлюються показники: серія днів утримання скидається або редагується, прогрес може коригуватися, оновлюється дата останньої перевірки та перераховується найдовша серія, а також перепланується нагадування;

– оновлена статистика зберігається локально та синхронізується з Firebase;

– інтерфейс відображає актуальні дані та виводиться підтверджувальне повідомлення.

Висновки до розділу

У ході виконання розділу 1 проведено комплексне передпроєктне обстеження предметної області відстеження звичок та залежностей, що дозволило сформувати чітке розуміння завдання дипломного проєкту та визначити основні напрями розробки програмного забезпечення HabitTracker.

Робота розпочалася з постановки завдання, яке охоплює аналіз предметної області, бізнес-процесів, існуючих рішень, розроблення вимог, архітектури, програмного забезпечення, аналіз безпеки, тестування, розгортання та створення документації. Предметна область була

проаналізована з урахуванням теоретичних основ та сучасних тенденцій, включаючи мобільні додатки, хмарні сервіси, штучний інтелект і соціальні функції. Визначено ключові бізнес-процеси, зокрема реєстрацію, вибір звичок/залежностей, відстеження прогресу, нагадування, соціальну взаємодію та аналітику, які стали основою для моделювання за допомогою нотації BPMN.

Аналіз існуючих рішень виявив їхні переваги, такі як гейміфікація чи аналітика, та недоліки, зокрема обмежену офлайн-підтримку, недостатню персоналізацію та слабку підтримку складних залежностей. Це дозволило обґрунтувати унікальність HabitTracker, який пропонує гнучкі цільові періоди, офлайн-доступ і соціальні функції без складної гейміфікації. Порівняльний аналіз підходів до розробки (нативна, крос-платформна, гібридна) та мов програмування (Java, Kotlin) підтвердив вибір нативної розробки на Java з використанням Android Studio для забезпечення продуктивності та якості користувацького досвіду.

Моделювання бізнес-процесів у нотації BPMN деталізувало ключові сценарії використання ПЗ, що забезпечило чітке розуміння взаємодії користувача з системою. Проведений аналіз заклав міцну основу для подальшої розробки, визначивши функціональні та нефункціональні вимоги, а також технічні рішення, які враховують сучасні потреби користувачів і обмеження існуючих аналогів.

Таким чином, результати передпроєктного обстеження підтвердили доцільність розробки HabitTracker як універсального, користувацько-орієнтованого рішення для відстеження звичок і подолання залежностей із потенціалом для подальшого вдосконалення.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Діаграма варіантів використання з ключовими акторами та основними функціями показана на рисунку 2.1.

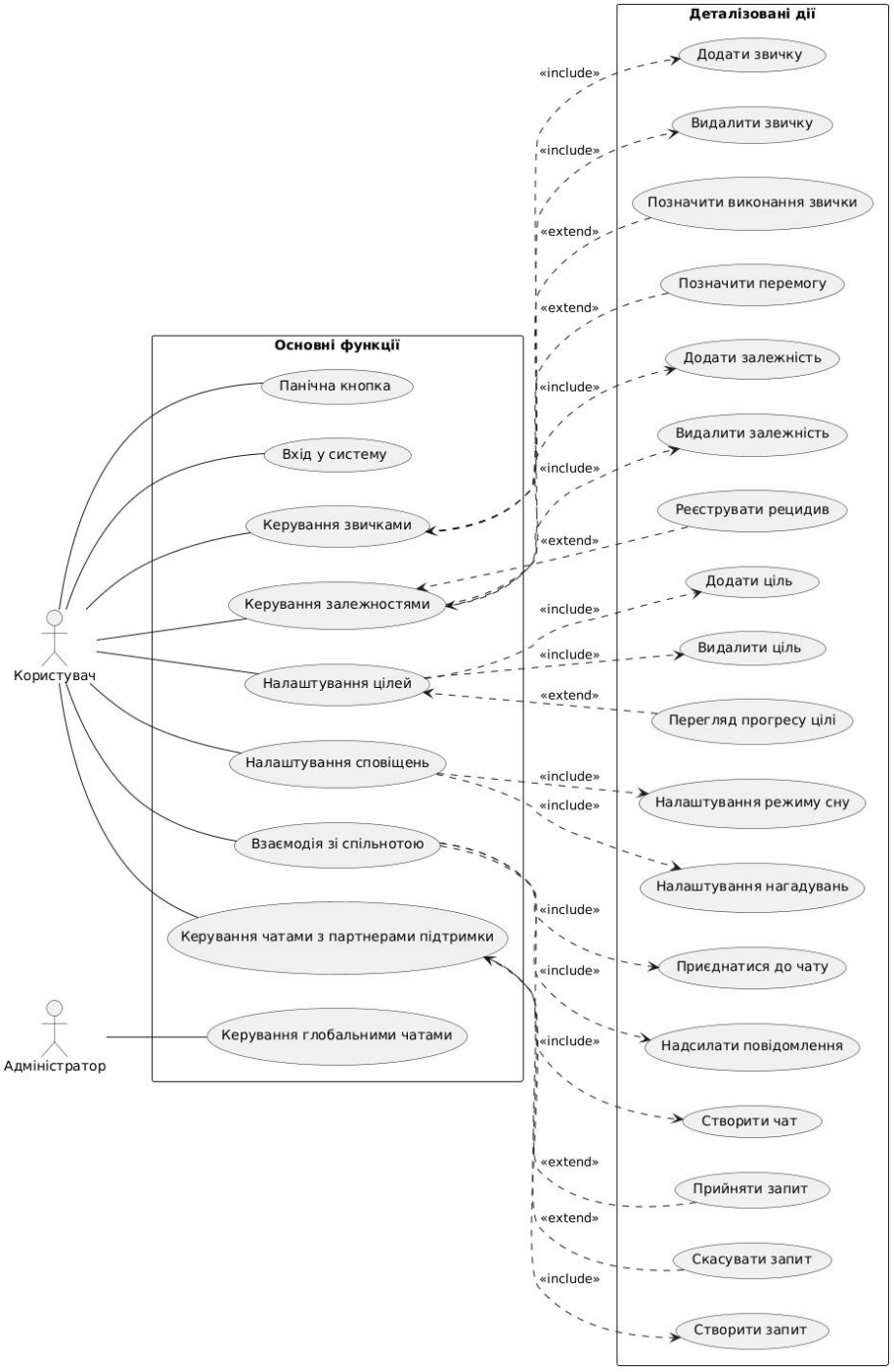


Рисунок 2.1 – Діаграма варіантів використання

В таблицях 2.1-2.12 наведено опис варіантів використання програмного забезпечення.

Таблиця 2.1 – Варіант використання UC-01

Use case name	Керування звичками
Use case ID	UC-01
Goals	Додавання, видалення або позначення виконання звичок
Actors	Зареєстрований користувач
Trigger	Користувач бажає керувати звичками
Pre-conditions	Користувач увійшов в обліковий запис та пройшов початкове налаштування
Flow of Events	1. Користувач переходить на вкладку "Habits" через BottomNavigationView. 2. Додавання звички: – натискає FloatingActionButton, відкривається діалог; – вибирає тип (habit) і назву (передвизначену або власну); – звичка додається до habitKeys і Firebase. 3. Позначення виконання: – натискає кнопку "Mark Completion" на картці звички; – система позначає завершення і оновлює статистику. 4. Видалення: – натискає кнопку "Delete" і підтверджує дію; – звичка видаляється з habitKeys і Firebase.
Extension	Якщо користувач не вибрав назву звички, відображається повідомлення про помилку. Якщо звичка вже позначена як виконана на сьогодні, дія ігнорується.
Post-Condition	Дані звичок оновлюються в локальному сховищі і Firebase, відображаються на екрані.

Таблиця 2.2 – Варіант використання UC-02

Use case name	Керування залежностями
Use case ID	UC-02
Goals	Додавання, видалення, позначення перемоги або рецидиву залежностей

Продовження таблиці 2.2

Actors	Зареєстрований користувач
Trigger	Користувач бажає керувати залежностями
Pre-conditions	Користувач увійшов в обліковий запис та пройшов початкове налаштування
Flow of Events	1. Користувач переходить на вкладку "Addictions" через BottomNavigationView. 2. Додавання залежності: – натискає FloatingActionButton і відкриває діалог; – вибирає тип (addiction) і назву, залежність додається до addictionKeys. 3. Позначення перемоги: – натискає "Mark Victory", додаючи відповідну позначку; – оновлюється прогрес. 4. Реєстрація рецидиву: – натискає "Relapse", відкривається діалог; – вибирає дату/час через showDatePickerDialog і showTimePickerDialog; – рецидив додається, прогрес зменшується. 5. Видалення: – натискає "Delete" і підтверджує.
Extension	Якщо дата рецидиву в майбутньому, відображається помилка.
Post-Condition	Дані залежностей оновлюються, статистика відображається на екрані.

Таблиця 2.3 – Варіант використання UC-03

Use case name	Керування залежностями
Use case ID	UC-03
Goals	Додавання, видалення або перегляд прогресу цілей
Actors	Зареєстрований користувач
Trigger	Користувач бажає встановити або керувати цілями
Pre-conditions	Користувач увійшов в обліковий запис та пройшов початкове налаштування

Продовження таблиці 2.3

Flow of Events	1. На головному екрані натискає FloatingActionButton, відкривається діалог. 2. Вибирає звичку/залежність, тип цілі (streak/progress) і цільове значення. 3. Ціль додається до goals і зберігається в Firebase. 4. Система перевіряє статус цілі і оновлює прогрес. 5. Для видалення: – натискає "Delete Goal" на картці цілі; – ціль видаляється з goals і Firebase.
Extension	Якщо ціль не досягнута, статус залишається "in progress". Якщо звичка/залежність не вибрана, відображається помилка.
Post-Condition	Список цілей оновлюється, досягнуті цілі позначаються.

Таблиця 2.4 – Варіант використання UC-04

Use case name	Налаштування сповіщень про відмічання залежностей та режиму сну
Use case ID	UC-04
Goals	Налаштування сповіщень про відмічання залежностей та режиму сну
Actors	Зареєстрований користувач
Trigger	Користувач бажає налаштувати сповіщення про відмічання залежностей та режим сну
Pre-conditions	Користувач увійшов в обліковий запис та пройшов початкове налаштування

Продовження таблиці 2.4

Flow of Events	1. Користувач переходить на вкладку "Settings" і відкриває діалог. 2. Для режиму сну: – вмикає/вимикає перемикач режиму сну; – встановлює час початку/кінця. 3. Для нагадувань про залежності: – вмикає перемикач нагадувань відмітитись по залежності; – встановлює час натисненням кнопки. 4. Зміни зберігаються в notifPrefs. 5. Викликається перепланування сповіщень.
Extension	Якщо режим сну активний, повідомлення переплановуються поза періодом сну.
Post-Condition	Нагадування перепланові, налаштування збережені в notifPrefs.

Таблиця 2.5 – Варіант використання UC-05

Use case name	Створення або приєднання до чатів, надсилання повідомлень
Use case ID	UC-05
Goals	Створення або приєднання до чатів, надсилання повідомлень
Actors	Зареєстрований користувач, Адміністратор
Trigger	Користувач бажає взаємодіяти зі спільнотою
Pre-conditions	1. Користувач увійшов у систему 2. Користувач має встановлене ім’я користувача
Flow of Events	1. Користувач переходить на вкладку "Messages". 2. Для створення чату (тільки для адміністратора): – натискає FloatingActionButton, відкривається діалог; – вказує назву, обмеження за статтю/віком; – чат зберігається в Firebase. 3. Для приєднання: – вибирає чат зі списку; – переходить до екрана для надсилання повідомлень.

Продовження таблиці 2.5

Extension	Якщо користувач не Admin, створення глобального чату недоступне. Якщо обмеження чату (стать/вік) не відповідають профілю користувача, чат не відображається.
Post-Condition	Чати відображаються, користувач може надсилати повідомлення в CommunityActivity.

Таблиця 2.6 – Варіант використання UC-06

Use case name	Керування партнерами підтримки
Use case ID	UC-07
Goals	Створення, прийняття або скасування запитів на партнерство
Actors	Зареєстрований користувач
Trigger	Користувач бажає знайти або керувати партнерами по відповідальності
Pre-conditions	1. Користувач увійшов у систему 2. Користувач має встановлені ім’я користувача, ім’я, стать, вік
Flow of Events	1. Користувач відкриває діалог створення запиту. 2. Вибирає звичку/залежність і надсилає запит. 3. Для прийняття запиту: – натискає "Accept Request"; – створюється чат між користувачами. 4. Для скасування: – натискає "Cancel Request"; – запит позначається як неактивний.
Extension	Якщо профіль не заповнений, відображається помилка. Якщо запит не знайдено, відображається помилка.
Post-Condition	Запити та чати оновлюються, відображаються на екрані.

Таблиця 2.7 – Варіант використання UC-07

Use case name	Режим паніки
Use case ID	UC-07
Goals	Активація анімації для подолання кризового стану
Actors	Зареєстрований користувач
Trigger	Користувач відчуває потребу в підтримці через залежність
Pre-conditions	1. Користувач увійшов у систему 2. Користувач має встановлені ім’я користувача, ім’я, стать, вік
Flow of Events	1. Користувач натискає кнопку "Panic" на картці залежності. 2. Відкривається діалог з анімацією. 3. Мотиваційні повідомлення змінюються кожні 2 секунди. 4. Користувач закриває діалог, натиснувши "Close".
Extension	Якщо діалог скасовується, анімація припиняється.
Post-Condition	Діалог закрито, користувач повертається до екрану залежностей.

Таблиця 2.8 – Варіант використання UC-08

Use case name	Реєстрація користувача
Use case ID	UC-08
Goals	Реєстрація нового користувача в системі через Google Sign-In та введення персональних даних
Actors	Гість (незареєстрований користувач)
Trigger	Користувач бажає зареєструватися в системі
Pre-conditions	1. Користувач має обліковий запис Google 2. Додаток підключений до Firebase

Продовження таблиці 2.8

Flow of Events	1. Користувач відкриває додаток і натискає кнопку "Sign In with Google" на початковому екрані. 2. Система запускає процес Google Sign-In, відкриваючи діалог вибору облікового запису Google. 3. Після вибору облікового запису Google система отримує idToken і виконує автентифікацію в Firebase. 4. Якщо користувач новий, відображається форма для введення персональних даних: ім’я, вік, стать. 5. Користувач заповнює форму і натискає кнопку "Next". 6. Система перевіряє коректність даних: – ім’я не порожнє; – вік у межах 14–90 років; – стать вибрана (не "Виберіть стать"). 7. Дані зберігаються в Firebase та локально в SharedPreferences. 8. Користувач перенаправляється на екран вибору звичок.
Extension	Якщо введено некоректний вік, відображається повідомлення про помилку. Якщо стать не вибрана, відображається повідомлення. Якщо автентифікація Google не вдалася (наприклад, через відсутність мережі), відображається повідомлення. Якщо Firebase не ініціалізовано, відображається помилка.
Post-Condition	Обліковий запис користувача створено, персональні дані збережено, користувач перенаправлений на екран вибору звичок.

Таблиця 2.9 – Варіант використання UC-09

Use case name	Початковий вибір звичок та залежностей
Use case ID	UC-09
Goals	Вибір або створення користувацьких звичок та залежностей для відстеження
Actors	Користувач

Продовження таблиці 2.9

Trigger	Користувач завершив введення персональних даних і бажає вибрати звички та залежності
Pre-conditions	Користувач автентифікований і має заповнені персональні дані, але не має вибраних звичок та залежностей
Flow of Events	1. Користувач переходить на екран HabitsActivity. 2. Система відображає список передвизначених звичок у вигляді CheckBox. 3. Користувач вибирає звички, позначаючи CheckBox, або додає власну звичку через поле для введення тексту і кнопку. 4. Звички зберігаються в selectedHabits, SharedPreferences і Firebase. 5. Користувач переходить на AddictionsActivity. 6. Система відображає список передвизначених залежностей. 7. Користувач вибирає залежності або додає власну через поле для введення тексту і кнопку. 8. Залежності зберігаються в selectedAddictions, SharedPreferences і Firebase. 9. Користувач натискає "Finish" і переходить до ResultsActivity.
Extension	Якщо не вибрано жодної звички чи залежності, кнопки "Next"/"Finish" неактивні. Якщо поле для власної звички/залежності порожнє, дія додавання ігнорується. Якщо збереження в Firebase не вдалося, відображається помилка. Користувач може повернутися до вибору звичок через returnToHabitsButton.
Post-Condition	Звички та залежності збережені, користувач перенаправлений на головний екран.

Таблиця 2.10 – Варіант використання UC-10

Use case name	Налаштування нагадувань про звички
Use case ID	UC-10
Goals	Налаштування періодичних нагадувань для виконання звичок
Actors	Користувач
Trigger	Користувач бажає налаштувати нагадування для звички
Pre-conditions	1. Користувач увійшов у систему 2. Є додані звички
Flow of Events	1. Користувач переходить на вкладку "Habits" через BottomNavigationView. 2. На картці потрібної звички користувач натискає кнопку Reminders. 3. Користувач вибирає частоту нагадувань та час першого нагадування. 4. Користувач натискає кнопку Save. 5. Нагадування плануються планувальником.
Extension	Якщо вибрано власний інтервал, але введено некоректне значення, встановлюється значення за замовчуванням. Якщо час нагадування в минулому, система автоматично коригує його.
Post-Condition	Нагадування налаштовані та заплановані, користувач отримує сповіщення.

Таблиця 2.11 – Варіант використання UC-11

Use case name	Налаштування нагадувань про звички
Use case ID	UC-11
Goals	Вихід користувача з системи та очищення сесії
Actors	Користувач
Trigger	Користувач бажає вийти з облікового запису
Pre-conditions	Користувач автентифікований

Продовження таблиці 2.11

Flow of Events	1. Користувач викликає вихід із системи (наприклад, через помилку автентифікації або вручну). 2. Система виконує вихід із Firebase через метод signOut. 3. Система виконує вихід із Google Sign-In. 4. Інтерфейс оновлюється, приховуючи форму введення даних і показуючи кнопку автентифікації.
Extension	Якщо Firebase не ініціалізовано, вихід із Google Sign-In виконується окремо. Якщо вихід із Google Sign-In не вдався, відображається попередження в логах, але інтерфейс оновлюється.
Post-Condition	Сесія користувача завершена, відображається екран автентифікації.

2.2 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.12 наведено загальну модель вимог, а в таблиці 2.13 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити в таблиці 2.14.

Таблиця 2.12 – Загальна модель вимог

№	Назва	ID Вимоги	Пріоритети	Ризики
1	Перегляд сторінки привітання.	FR-1	Низький	Низький
2	Система авторизації користувача.	FR-2	Високий	Високий
2.1	Реєстрація користувача.	FR-3	Високий	Високий
2.2	Авторизація користувача.	FR-4	Високий	Високий
2.3	Вихід із акаунту	FR-5	Середній	Низький
3	Вибір звичок та залежностей	FR-6	Високий	Середній
4	Керування звичками	FR-7	Високий	Середній
5	Керування залежностями	FR-8	Високий	Середній

Продовження таблиці 2.12

6	Налаштування цілей	FR-9	Середній	Низький
7	Налаштування повідомлень	FR-10	Середній	Низький
8	Налаштування нагадувань про звички	FR-11	Середній	Низький
9	Взаємодія зі спільнотою	FR-12	Середній	Середній
10	Керування партнерами по відповідальності	FR-13	Середній	Середній
11	Режим паніки	FR-14	Низький	Низький

Таблиця 2.13 – Перелік функціональних вимог

Назва	Опис
FR-1	Перегляд сторінки привітання Неавторизований користувач бачить екран привітання в MainActivity. На екрані відображається кнопка для автентифікації через Google Sign-In (signInButton), яка дозволяє перейти до реєстрації або авторизації.
FR-2	Система авторизації користувача Система забезпечує автентифікацію користувача через Google Sign-In та Firebase Authentication. Після успішної автентифікації користувач може заповнити персональні дані (ім’я, вік, стать) та отримати доступ до функціоналу програми. Реалізовано в MainActivity через методи signIn, firebaseAuthWithGoogle, saveUserDataToFirebase.
FR-3	Реєстрація користувача Користувач проходить автентифікацію через Google Sign-In, після чого заповнює форму з персональними даними (ім’я, вік, стать) у MainActivity. Дані зберігаються в Firebase (users/{userId}) та локально в SharedPreferences (user_data). Після цього користувач перенаправляється до вибору звичок (HabitsActivity).

Продовження таблиці 2.13

FR-4	Авторизація користувача Користувач, який уже має обліковий запис, автентифікується через Google Sign-In. Система перевіряє наявність даних у Firebase (checkUserDataInFirebase). Якщо дані повні, користувач перенаправляється до ResultsActivity, інакше — до заповнення форми в MainActivity.
FR-5	Вихід із акаунту Користувач може вийти з облікового запису, викликаючи вихід із Firebase (mAuth.signOut) та Google Sign-In (mGoogleSignInClient.signOut) через методи signOut і signOutGoogleOnly у MainActivity. Після виходу інтерфейс оновлюється, показуючи екран автентифікації.
FR-6	Вибір звичок та залежностей Користувач вибирає передвизначені або створює власні звички в HabitsActivity та залежності в AddictionsActivity. Звички/залежності зберігаються в SharedPreferences (selectedHabits, selectedAddictions) та Firebase (users/{userId}/habits, users/{userId}/addictions). Реалізовано через методи saveHabitsToSharedPreferences, saveAddictionsToFirebase.
FR-7	Керування звичками Користувач може додавати, видаляти або позначати виконання звичок у ResultsActivity. Додавання відбувається через діалог (showAddItemDialog), позначення виконання — через кнопку "Mark Completion" (markHabitCompletion), видалення — через "Delete" (showDeleteConfirmationDialog). Дані оновлюються в habitKeys, SharedPreferences та Firebase.

Продовження таблиці 2.13

FR-8	Керування залежностями Користувач може додавати, видаляти, позначати перемогу або реєструвати рецидив залежностей у ResultsActivity. Позначення перемоги (markToday), рецидиву (showRelapseDialog) та видалення (showDeleteConfirmationDialog) оновлюють addictionKeys, SharedPreferences та Firebase.
FR-9	Налаштування цілей Користувач може додавати, видаляти та переглядати прогрес цілей у ResultsActivity. Цілі додаються через діалог (showAddGoalDialog), видаляються через "Delete Goal" (showDeleteGoalConfirmationDialog). Прогрес перевіряється через checkAndUpdateGoalStatus. Дані зберігаються в goals, SharedPreferences та Firebase.
FR-10	Налаштування повідомлень Користувач налаштовує режим сну та нагадування про залежності в ResultsActivity через діалог налаштувань (showSettingsScreen). Налаштування зберігаються в notifPrefs, а повідомлення переплановуються через rescheduleAllNotifications.
FR-11	Налаштування нагадувань про звички Користувач налаштовує періодичні нагадування для звичок у ResultsActivity через діалог (showReminderDialog). Частота та час зберігаються в statsPrefs (saveHabitReminderTime), а нагадування плануються через scheduleNotification.
FR-12	Взаємодія зі спільнотою Користувач може створювати або приєднуватися до чатів у ResultsActivity. Чати створюються через діалог (showAddItemDialog), а приєднання — через вибір зі списку (addChatCard). Дані зберігаються в Firebase (chats або users/{userId}/chats). Адміністратори можуть створювати глобальні чати.

Продовження таблиці 2.13

FR-13	Керування партнерами по відповідальності Користувач створює, приймає або скасовує запити на партнерство в ResultsActivity. Запити створюються через showAccountabilityPartnerDialog, приймаються через "Accept Request" (acceptAccountabilityRequest), скасовуються через "Cancel Request" (cancelAccountabilityRequest). Дані зберігаються в Firebase (accountability_requests, accountability_chats).
FR-14	Режим паніки Користувач активує анімацію для подолання кризового стану, натиснувши кнопку "Panic" на картці залежності в ResultsActivity. Анімація відображається через showPanicAnimation, показуючи мотиваційні повідомлення.

Таблиця 2.14 – Матриця трасування вимог

	UC-01	UC-02	UC-03	UC-04	UC-05	UC-06	UC-07	UC-08	UC-09	UC-10	UC-11
FR-1								X			
FR-2								X			X
FR-3								X			
FR-4								X			
FR-5											X
FR-6									X		
FR-7	X										
FR-8		X									
FR-9			X								
FR-10				X							
FR-11										X	
FR-12					X						
FR-13						X					
FR-14							X				

2.3 Розроблення нефункціональних вимог

Доступність:

– програма повинна бути працездатною 100% часу;

– допустимий час відновлення після відмови (наприклад, через помилку Firebase) не повинен перевищувати 30 хвилин;

– відстеження працездатності здійснюється через перевірку відповіді Firebase Authentication та Realtime Database на запити.

Продуктивність:

– час відгуку інтерфейсу користувача (наприклад, перехід між вкладками в BottomNavigationView) не повинен перевищувати 1 секунду на пристроях із мінімальними вимогами (Android 7.0, 2 ГБ RAM);

– синхронізація даних із Firebase (збереження/завантаження звичок, залежностей, цілей через saveUserDataToFirebase, loadDataFromFirebase) має відбуватися протягом 3 секунд за стабільного з’єднання;

– оновлення статистики звичок і залежностей (updateStats) має тривати не довше 500 мілісекунд для набору до 50 елементів.

Обмеження:

– програма реалізується мовою Java з використанням Android SDK та бібліотек Firebase (Realtime Database, Authentication);

– локальне зберігання даних здійснюється через SharedPreferences у форматі ключ-значення;

– мінімальна конфігурація пристрою: Android 7.0 (API 24), 2 ГБ RAM, 100 МБ вільного місця на диску.

Безпека:

– автентифікація через Google Sign-In та Firebase Authentication захищає дані користувача за допомогою безпечного обміну токенами (idToken у firebaseAuthWithGoogle);

– персональні дані користувача (ім’я, вік, стать) зберігаються в Firebase Realtime Database із доступом, обмеженим для автентифікованого користувача (users/{userId});

– для запобігання несанкціонованому доступу до чатів застосовуються обмеження за віком і статтю.

Вимоги до зберігання даних:

– локальні дані (SharedPreferences) зберігаються безстроково, доки користувач не видалить програму або не очистить дані;

– дані в Firebase (звички, залежності, цілі) зберігаються постійно, доки користувач не видалить їх через інтерфейс.

Масштабованість:

– програма повинна підтримувати до 10000 одночасних користувачів, використовуючи масштабовану інфраструктуру Firebase Realtime Database;

– архітектура дозволяє додавання нових типів звичок і залежностей без зміни основного коду.

Конфігурованість:

– налаштування режиму сну та нагадувань зберігаються в SharedPreferences і можуть бути змінені через UI;

– зміна налаштувань (наприклад, sleepModeEnabled, addictionReminderHour) не потребує модифікації коду.

Зручність використання:

– інтерфейс підтримує локалізацію (українська, англійська) через ресурси R.string для всіх текстових елементів;

– адаптація до світлої/темної теми здійснюється автоматично;

– навігація між вкладками (BottomNavigationView) є інтуїтивною, з максимумом 2 кліків для доступу до основних функцій;

– повідомлення про помилки стандартизовані через Toast із чіткими текстами;

– іменування елементів UI (наприклад, fab_add, habits_list_container) відповідає їх функціональному призначенню.

2.4 Аналіз системних вимог

Для роботи програми достатньо наявності:

– пристрою з операційною системою Android 7.0 (API 24) або вище, 2 ГБ або більше оперативної пам’яті, 100 МБ або більше вільного місця;

– встановлених Google Play Services для підтримки Google Sign-In (GoogleSignInClient) та Firebase SDK;

– підключення до мережі Інтернет для автентифікації та синхронізації даних із Firebase.

За мінімальну швидкість підключення вважатимемо таку, яка забезпечить синхронізацію даних обсягом до 1 МБ (типовий розмір даних користувача, включаючи звички, залежності, цілі) за 5 секунд:

$$1 \text{ МБ} / 5 \text{ с} * 8 \text{ біт/Б} = 1.6 \text{ Мбіт/с}.$$

Рекомендована швидкість підключення становить ~10 Мбіт/с, що відповідає типовим умовам тестування на пристроях розробки.

2.5 Аналіз економічних показників програмного забезпечення

Для аналізу економічних показників програмного забезпечення для відстеження особистих звичок та подолання залежностей використано метод підрахунку точок варіантів використання (Use Case Points, UCP). Цей метод дозволяє оцінити трудомісткість розробки, враховуючи складність варіантів використання, акторів, технічні фактори та фактори середовища. Розрахунок виконано для одного виконавця, з урахуванням специфікації вимог, описаних у попередніх розділах.

В таблиці 2.15 наведемо результати оцінки складності для кожного з варіантів використання.

Таблиця 2.15 – Складність варіантів використання

Use Case ID	Use Case Name	Кількість транзакцій	Складність
UC-02	Керування звичками	6	Середній
UC-03	Керування залежностями	8	Складний
UC-04	Налаштування цілей	5	Середній
UC-05	Налаштування повідомлень	5	Середній

Продовження таблиці 2.15

UC-06	Взаємодія зі спільнотою	7	Середній
UC-07	Керування партнерами по відповідальності	8	Складний
UC-08	Режим паніки	4	Середній
UC-09	Реєстрація користувача	8	Складний
UC-10	Вибір звичок та залежностей	9	Складний
UC-11	Налаштування нагадувань про звички	6	Середній
UC-13	Вихід із системи	4	Середній

На основі зробленої оцінки обчислимо нескориговану оцінку варіантів використання (UUCW) в таблиці 2.16.

Таблиця 2.16 – Нескоригована оцінка варіантів використання (UUCW)

Складність	Ваговий коефіцієнт	Кількість варіантів використання	Програмний продукт
Простий	5	0	0
Середній	10	6	60
Складний	15	4	60
Всього			120

Далі розрахуємо нескориговану оцінку акторів (UAW). Розроблювана система має двох акторів – звичайного користувача та адміністратора. Розрахунок наведемо в таблиці 2.17.

Таблиця 2.17 – Нескоригована оцінка акторів (UAW)

Складність	Ваговий коефіцієнт	Кількість акторів	Програмний продукт
Простий	1	1 (Користувач)	1
Середній	2	1 (Адміністратор)	2
Складний	3	0	0
Всього			3

Нескориговані точки варіантів використання (UUCP) обчислюються як:

$$UUCP = UUCW + UAW = 120 + 3 = 123$$

Наступний етап — визначення фактора технічної складності. Для цього оцінюють вплив кожного з 13 технічних факторів на розробку програмного продукту за шкалою від 0 (незначний) до 5 (критичний) і розраховують зважену суму цих факторів. Результати розрахунків наведено в таблиці 2.34.

Таблиця 2.18 – Зважена сума технічних факторів (TFactor)

Фактор	Ваговий коефіцієнт	Оцінка	Вплив
Розподілена система	2	4	8
Вимоги до швидкодії	2	3	6
Ефективність для користувача	1	4	4
Складність внутрішньої обробки	1	3	3
Повторне використання коду	1	3	3
Простота встановлення	0.5	2	1
Простота використання	0.5	4	2
Портативність	2	4	8
Простота зміни	1	3	3
Паралельне використання	1	2	2
Безпека	1	4	4
Доступ третіх сторін	1	3	3
Потреба в навчанні	1	2	2
Всього (TFactor)			49

Фактор технічної складності (TCF):

$$TCF = 0.6 + (0.01 * TFactor) = 0.6 + (0.01 * 49) = 1.09 .$$

Наступним кроком є оцінка складності середовища. Для цього необхідно оцінити вплив кожного з 8 факторів середовища за шкалою від 0

до 5 та розрахувати їх зважену суму (EFactor). Результати розрахунків подано в таблиці 2.19.

Таблиця 2.19 – Зважена сума факторів середовища (EFactor)

Фактор	Ваговий коефіцієнт	Оцінка	Вплив
Досвід з процесом розробки	1.5	4	6.0
Досвід у предметній області	0.5	3	1.5
Досвід з ООП	1	4	4.0
Компетентність аналітика	0.5	3	1.5
Мотивація	1	4	4.0
Стабільність вимог	2	3	6.0
Часткова зайнятість розробників	-1	0	0.0
Складність мови програмування	-1	2	-2.0
Всього (EFactor)			21

Тоді фактор середовища (EF):

$$EF = 1.4 + (-0.03 * EFactor) = 1.4 + (-0.03 * 21) = 0.77 .$$

Точки варіантів використання (UCP):

$$UCP = UUCP * TCF * EF = 123 * 1.09 * 0.77 \approx 103 .$$

За обчисленим значенням UCP = 103, загальна трудомісткість реалізації програмного забезпечення становить 103 людино-дні.

Припустимо, що зарплата розробника становить 1000 грн/день. Таким чином, орієнтовна вартість реалізації програмного забезпечення - 103 000 грн.

2.6 Постановка завдання на розробку програмного забезпечення

Головною метою розробки є створення мобільного додатку для відстеження особистих звичок та подолання залежностей, що сприяє

підвищенню самодисципліни, мотивації та психологічної стійкості користувачів. Додаток спрямований на спрощення процесу формування позитивних звичок, боротьби з залежностями та підтримки користувачів через інтерактивні інструменти, соціальну взаємодію та персоналізовані нагадування.

Досягнення мети розробки планується шляхом створення мобільного додатку для платформи Android, який інтегрується з хмарними сервісами Firebase для автентифікації, зберігання даних та соціальної взаємодії. Додаток матиме модульну архітектуру, що включає інтерфейс користувача для керування звичками, залежностями, цілями, сповіщеннями, а також функції соціальної взаємодії (групові чати, чати з партнерами підтримки) та режим паніки для підтримки в кризових ситуаціях. Програмне забезпечення реалізоване з використанням Java, Android SDK та бібліотек Firebase, забезпечуючи стабільність та безпеку.

Функціональні задачі, що підлягають вирішенню в ході реалізації програмного забезпечення на дипломний проєкт, включають:

– забезпечення автентифікації користувачів через Google Sign-In та Firebase Authentication для безпечного доступу до персоналізованих даних;

– реалізацію механізмів додавання, видалення, позначення виконання звичок та залежностей із збереженням даних локально та в хмарі;

– створення системи цілей для відстеження прогресу користувача;

– налаштування персоналізованих нагадувань для звичок і залежностей з підтримкою режиму сну;

– організацію соціальної взаємодії через групові чати та партнерство підтримки (showAccountabilityPartnerDialog);

– реалізацію режиму паніки для мотиваційної підтримки в кризових ситуаціях;

– забезпечення офлайн-режиму для роботи без підключення до мережі.

У результаті розробки очікується створення стабільного, зручного та масштабованого мобільного додатку, який може бути використаний як інструмент для саморозвитку, формування звичок та подолання залежностей. Додаток буде адаптивним до різних Android-пристроїв, підтримуватиме українську та англійськау локпалізації, а також темну/світлу теми оформлення. Інтеграція з Firebase забезпечить безпечне зберігання даних, синхронізацію та соціальну взаємодію, а модульна структура дозволить легко розширювати функціонал (наприклад, додавання нових типів звичок чи інтеграцію з іншими платформами). Додаток стане ефективним помічником для користувачів у сферах особистого розвитку, психології та здорового способу життя.

Висновки до розділу

У ході розроблення вимог до мобільного додатку для відстеження особистих звичок та подолання залежностей було виконано комплексний аналіз, що охопив усі аспекти специфікації програмного забезпечення. Було виділено 11 варіантів використання, які описують ключові сценарії взаємодії користувача з додатком. На основі цих варіантів створено UML-діаграму варіантів використання, що відображає взаємодію двох акторів (користувача та адміністратора) з основними функціями додатку, а також детально описано кожен варіант у вигляді таблиць із потоками подій, передумовами, післяумовами та альтернативними сценаріями.

На основі варіантів використання сформульовано 14 функціональних вимог. Побудовано матрицю трасування вимог у вигляді таблиці, що демонструє чіткий зв’язок між функціональними вимогами та відповідними варіантами використання, забезпечуючи відстежуваність реалізації кожної вимоги.

Для забезпечення якості додатку визначено 12 нефункціональних вимог, класифікованих за категоріями, і розроблено специфікацію системних вимог до користувача.

Для оцінки економічних показників застосовано метод підрахунку точок варіантів використання. Загальна оцінка склала 103 UCP, що відповідає 103 людино-дням для одного розробника. Було визначено, що орієнтовна вартість розробки становить 103,000 грн.

Під час формулювання завдання було окреслено мету розробки, визначено цілі, завдання та основні характеристики програмного забезпечення.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

На етапі аналізу вимог та технічних рішень для розробки програмного забезпечення трекера звичок та залежностей було обрано архітектурний патерн клієнт-сервер у поєднанні з монолітною архітектурою. Такий підхід забезпечує централізовану реалізацію бізнес-логіки на сервері, що спрощує розгортання, масштабування та підтримку системи. Клієнтська частина, реалізована як мобільний додаток для Android, відповідає за взаємодію з користувачем, відображення даних та передачу запитів до серверної частини через API. Серверна частина, у свою чергу, обробляє запити, керує даними та забезпечує синхронізацію з базою даних.

Для представлення архітектури використано C4 Model, яка дозволяє чітко та структуровано описати систему на різних рівнях абстракції. У цьому підрозділі наведено високорівневі діаграми C4 Model (рівні 1 та 2).

Контекстна діаграма, наведена на рисунку 3.1, відображає систему трекера звичок та залежностей як єдине ціле, показуючи її взаємодію з зовнішніми користувачами та системами.

Рисунок 3.1 – Контекстна діаграма (C4 Model рівень 1)

Діаграма контейнерів, наведена на рисунку 3.2, деталізує внутрішню структуру системи, розбиваючи її на основні компоненти (контейнери).

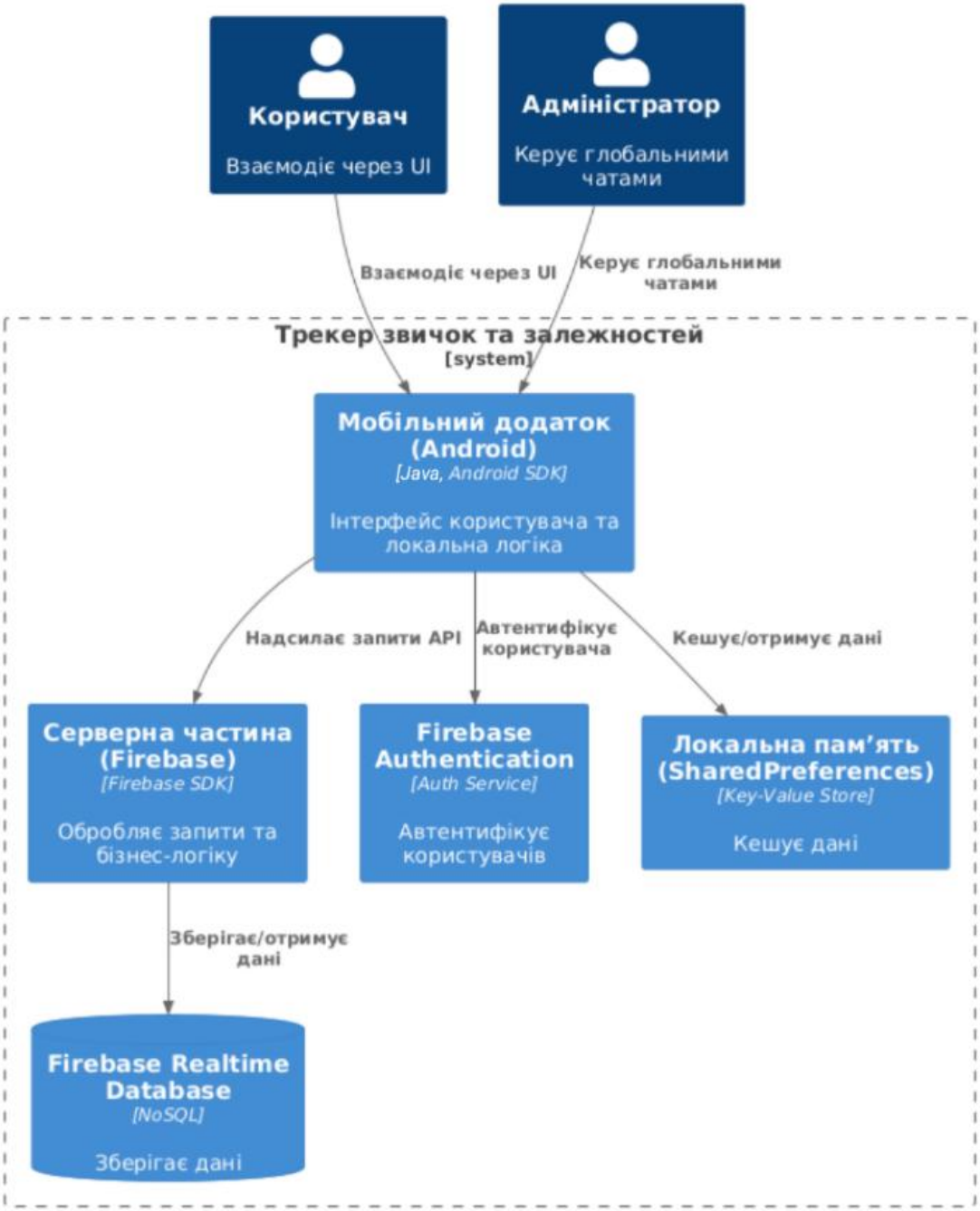


Рисунок 3.2 – Діаграма контейнерів (C4 Model рівень 2)

3.2 Архітектурні рішення та обґрунтування вибору засобів розробки

На етапі розробки програмного забезпечення (ПЗ) трекера звичок та залежностей було прийнято низку архітектурних рішень, спрямованих на забезпечення функціональних і нефункціональних вимог, таких як масштабованість, простота використання, безпека та підтримка офлайн-

режиму. У цьому підрозділі описано ключові архітектурні рішення, обґрунтовано вибір технологічного стеку, проведено порівняльний аналіз засобів розробки та сторонніх бібліотек, а також пояснено вибір бази даних та інтеграцій із зовнішніми системами.

Для управління користувачами використано Firebase Authentication, що забезпечує безпечну автентифікацію через Google [26]. Це рішення обрано через його простоту інтеграції, підтримку кількох методів автентифікації та високу безпеку (зокрема, шифрування даних). Користувачі отримують унікальний ідентифікатор (UID) після входу, який використовується для зв’язування з їхніми даними в Firebase Realtime Database. Для офлайн-доступу дані кешуються локально в SharedPreferences, що дозволяє користувачам переглядати звички, залежності та цілі без підключення до мережі.

ПЗ інтегрується з двома основними зовнішніми системами від Google:

– Firebase Realtime Database: NoSQL база даних для зберігання даних користувачів, звичок, залежностей, цілей, чатів та запитів на партнерство. Використовується протокол HTTPS для синхронізації даних у реальному часі, що забезпечує швидке оновлення інтерфейсу користувача.

– Firebase Authentication: забезпечує автентифікацію через OAuth 2.0 та JSON Web Tokens (JWT).

Ці інтеграції обрано через їх надійність, масштабованість та підтримку Google, що зменшує потребу в розробці власних серверних рішень.

Вибрано NoSQL базу даних (Firebase Realtime Database) через її гнучкість у роботі з ієрархічними даними, такими як списки звичок, залежностей, чатів та запитів. NoSQL підходить для системи, де структура даних може змінюватися (наприклад, додавання нових типів звичок або чатів) [27]. Локально використовується ключ-значення сховище (SharedPreferences) для кешування, що забезпечує швидкий доступ до даних у режимі офлайн.

Для планування нагадувань використано Android AlarmManager, який забезпечує точне виконання повідомлень навіть у режимі низького

енергоспоживання [20]. Нагадування для звичок підтримують гнучкі інтервали (щоденно, щогодини, кожну хвилину для тестування), тоді як нагадування для залежностей плануються щоденно. Режим сну блокує повідомлення в заданий час, що відповідає вимогам зручності користувача.

Групові чати та чати з партнерами зберігаються в Firebase Realtime Database як ієрархічні структури [28]. Запити на партнерство обробляються асинхронно, а після прийняття створюється окремий чат для двох користувачів. Також реалізовано обмеження доступу до групових чатів за статтю та віком.

Для мобільної розробки розглянуто три основні мови: Kotlin, Java та Dart (для Flutter). Порівняльний аналіз наведено в таблиці 3.1.

Таблиця 3.1 – Порівняння мов програмування для мобільної розробки

Критерій	Java	Kotlin	Dart (Flutter)
Платформи	Android, мультиплатформність	Android, мультиплатформність (KMM)	Android, iOS, Web, Desktop
Швидкість розробки	Середня	Висока	Висока
Підтримка бібліотек	Дуже широка (зріла екосистема)	Широка (Android, Ktor, Coroutines)	Широка (Flutter, Dart Packages)
Інтеграція з Android	Відмінна (традиційна мова Android)	Відмінна (офіційна мова Android)	Хороша, але через FFI
Продуктивність	Висока	Висока	Хороша
Споживання пам’яті	Середнє	Середнє	Високе
Досвід розробки	Широкий	Обмежений	Обмежений

Java обрано як основну мову завдяки її зрілій екосистемі, широкій підтримці в Android SDK та більшому досвіду розробника в цій мові [21]. Java забезпечує стабільність, сумісність із усіма версіями Android та доступ до величезної кількості бібліотек. Хоча Kotlin пропонує більш лаконічний синтаксис і null-safety, Java була обрана через широку документацію та меншу потребу в освоєнні нової мови. Dart/Flutter відхилено через необхідність кросплатформності, яка не є вимогою проєкту, та додатковий runtime.

Розглянуто три IDE для Android-розробки: Android Studio, IntelliJ IDEA та Eclipse. Порівняння наведено в таблиці 3.2.

Таблиця 3.2 – Порівняння IDE для Android-розробки

Критерій	Android Studio	IntelliJ IDEA	Eclipse
Підтримка розробки для Android	Повна	Хороша	Хороша
Інструменти налагодження	Розширені	Розширені	Базові
Плагіни	Багато	Дуже багато	Багато
Продуктивність	Середня	Висока	Висока
Досвід використання	Широкий	Широкий	Відсутній

Android Studio обрано через його глибоку інтеграцію з Android SDK, підтримку емуляторів, профайлерів, Gradle та інструментів для UI-дизайну (Layout Editor) [22]. Воно забезпечує повний набір функцій для Android-розробки на Java, включаючи автодоповнення, рефакторинг та інтеграцію з Firebase. IntelliJ IDEA, хоча й потужна, менш спеціалізована для Android, а Eclipse з плагіном ADT застарілий і менш зручний для сучасних проєктів.

Розглянуто три типи баз даних: NoSQL (Firebase Realtime Database), SQL (SQLite) та ключ-значення (SharedPreferences). Порівняння наведено в таблиці 3.3.

Таблиця 3.3 – Порівняння типів баз даних

Критерій	Firebase Realtime Database	SQLite	SharedPreferences
Тип бази даних	NoSQL (ієрархічна)	SQL (реляційна)	Ключ-значення
Масштабовність	Висока	Низька	Низька
Швидкість доступу	Залежить від мережі	Висока	Висока
Гнучкість структури	Висока (JSON-подібна)	Середня (схеми таблиць)	Низька (прості пари)
Обсяг даних	Великий (хмарний)	Обмежений (локальний)	Невеликий (кешування)
Складність інтеграції	Низька	Середня	Низька

Firebase Realtime Database обрано як основне сховище через її підтримку реального часу, офлайн-синхронізації та гнучкість NoSQL для ієрархічних даних [19]. SharedPreferences використовується для локального кешування завдяки простоті та швидкості доступу до невеликих обсягів даних [31]. SQLite відхилено через необхідність складнішого управління схемами та відсутність потреби в реляційних зв’язках.

Було обрано такі фреймворки та бібліотеки:

– Android SDK: основний фреймворк для розробки Android-додатків, що надає API для роботи з UI (Activity, Fragment), повідомленнями (AlarmManager), локальною пам’яттю (SharedPreferences) та апаратними функціями;

– Firebase SDK: вибрано для інтеграції з Firebase Realtime Database та Authentication. Firebase забезпечує швидку розробку, масштабованість та офлайн-синхронізацію. Альтернативи, такі як AWS Amplify або власний сервер на Java Spring, були відхилені через складність налаштування та вищу вартість;

– Material Design Components: використано для створення сучасного UI, що відповідає рекомендаціям Google [23]. Альтернативи, такі як кастомні View без Material, менш зручні для швидкої розробки.

Виконання нефункціональних вимог забезпечується наступним чином:

– масштабованість: Firebase Realtime Database автоматично масштабується для обробки великої кількості запитів, що підходить для зростаючої бази користувачів;

– офлайн-доступ: SharedPreferences використовується для локального кешування даних, що забезпечує доступ до функціоналу без підключення до мережі;

– безпека: Firebase Authentication захищає дані користувачів, а правила безпеки Firebase Realtime Database обмежують доступ до даних лише автентифікованим користувачам;

– продуктивність: локальна логіка (наприклад, обчислення стріків, прогресу, оновлення UI) виконується на клієнтській стороні, зменшуючи навантаження на сервер;

– простота використання: інтерфейс із підтримкою Material Design, адаптивним дизайном та темами (світла/темна, залежно від системних налаштувань) забезпечує зручність для користувачів.

3.3 Конструювання програмного забезпечення

3.3.1 Структури даних та програмні структури

Основні структури даних, використані в ПЗ, включають:

– HashSet<string></string>: використовується для зберігання унікальних ключів звичок (habitKeys) та залежностей (addictionKeys). Забезпечує швидкий доступ (O(1)) та унікальність елементів;

– HashMap<String, HabitStats>: зберігає статистику звичок (habitStats) та залежностей (addictionStats), де ключ – це ідентифікатор (наприклад, "reading" або "custom_habit_123"), а значення – об’єкт HabitStats;

– HashMap<String, Goal>: зберігає цілі (goals) з унікальними ідентифікаторами (наприклад, "goal_123");

– ArrayList<date></date>: використовується в HabitStats для зберігання дат рецидивів (relapseDates), позначених днів (markedDays) та позначок виконання (completionTimestamps);

– SharedPreferences: ключ-значення сховище для кешування даних (наприклад, selectedHabits, statsPrefs, goalsPrefs).

Клас HabitStats є центральною структурою для зберігання статистики:

```
public static class HabitStats {
    public Date startDate = new Date();
    public int streak;
    public int longestStreak;
    public int progress;
    public Date nextReminder;
    public Date lastCheckDate = startDate;
    public int reminderHour = 18;
    public int reminderMinute = 0;
    public int reminderInterval = 1;
    public List<Date> relapseDates = new ArrayList<>();
    public List<Date> markedDays = new ArrayList<>();
    public List<Date> completionTimestamps = new ArrayList<>();
    public int calculateLongestStreak() { ... }
}
```

Цей клас інкапсулює всі дані, необхідні для відстеження звички чи залежності, та забезпечує метод для обчислення найдовшого стріка.

У програмному забезпеченні присутні такі програмні структури:

– MainActivity: ініціалізує автентифікацію через Google Sign-In та Firebase, обробляє введення даних користувача (ім’я, вік, стать). Використовує шаблон MVC;

– ResultsActivity: головний екран, який координує навігацію через BottomNavigationView, відображає статистику, цілі, список чатів, налаштування. Реалізує шаблон Observer для Firebase-запитів;

– HabitsActivity: керує початковим вибором звичок, створенням кастомних звичок. Використовує шаблон Factory для CheckBox;

– AddictionsActivity: аналогічно керує початковим вибором залежностей, додаючи підтримку повернення до HabitsActivity;

– CommunityActivity: обробляє чати, відправлення повідомлень, FCM-нотифікації. Використовує Volley для HTTP-запитів;

– AddictionStatsActivity: відображає детальну статистику залежностей (календар, графіки). Використовує MPAndroidChart для візуалізації;

– MyFirebaseMessagingService: обробляє FCM-повідомлення, зберігає їх у SharedPreferences, відображає через MessagingStyle;

– HabitNotificationService: фоново керує сповіщеннями, відновлює їх після перезавантаження;

– HabitNotificationReceiver: обробляє події AlarmManager, відображає сповіщення;

– HabitNotificationScheduler: планує та скасовує сповіщення;

– BootReceiver: перезапускає HabitNotificationService після перезавантаження.

3.3.2 Опис структури бази даних

База даних побудована на основі NoSQL (Firebase Realtime Database) з ієрархічною структурою, що відображає сутності: користувачі, звички, залежності, цілі, чати та запити на партнерство. Концептуальна модель включає:

– User: користувач із унікальним ідентифікатором, іменем, віком та статтю;

– Habit/Addiction: звичка або залежність із ключем, назвою та статистикою;

– Goal: ціль, пов’язана зі звичкою чи залежністю;

– Chat: глобальний, особистий або партнерський чат;

– AccountabilityRequest: запит на партнерство для звички чи залежності.

Основні зв’язки логічної моделі включають:

– користувач має багато звичок, залежностей, цілей та чатів (1:N);

– звичка/залежність має одну статистику (1:1);

– ціль відноситься до однієї звички або залежності (1:1);

– чат може відноситися до однієї звички/залежності (1:0..1).

На рисунку 3.3 наведена ER-діаграма усіх сутностей, наявних в моделі:

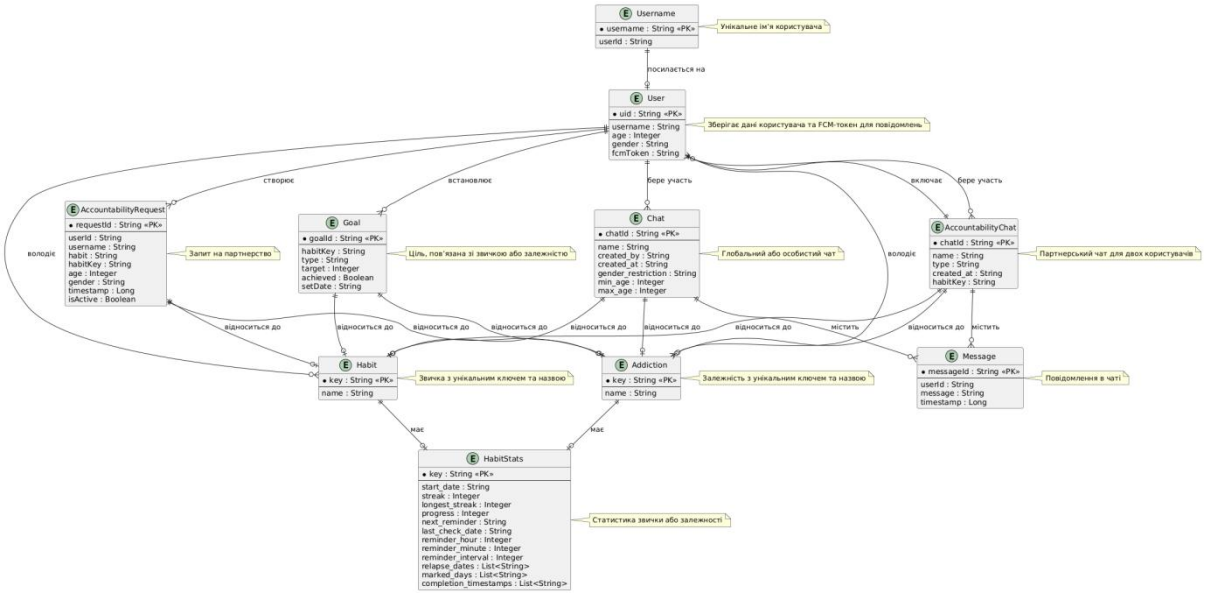


Рисунок 3.3 – ER діаграма сутностей

3.3.3 Інструменти розробки

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.4.

Таблиця 3.4 – Опис утиліт

№	Назва утиліти	Опис застосування
1	Android Studio	IDE для розробки, дебагінгу, тестування (Java, Gradle, Firebase)
2	Postman	Тестування Firebase API (структура вузлів)
3	Android SDK	API для UI (Activity, CardView), повідомлень (AlarmManager)
4	Firebase SDK	Інтеграція з Realtime Database, Authentication, FCM
5	Material Design Components	Сучасний UI (CardView, BottomNavigationView)
6	Volley	HTTP-запити для FCM-сповіщень (CommunityActivity)
7	MPAndroidChart	Візуалізація графіків (AddictionStatsActivity)
8	Gson	Серіалізація/десеріалізація JSON для сповіщень (HabitNotificationService)
9	Git	Управління версіями коду
10	Java Standard Library	Колекції (HashSet, HashMap), дати (Date, Calendar)

Тексти програмного коду наведені в окремому документі «Текст програми».

3.4 Аналіз безпеки даних

У процесі розробки програмного забезпечення (ПЗ) трекера звичок та залежностей особлива увага приділялася забезпеченню безпеки даних користувачів, враховуючи потенційні загрози, пов’язані з обробкою персональних даних, повідомлень та інтеграцією з хмарними сервісами. У цьому підрозділі проведено аналіз вразливостей ПЗ, оцінено ризики безпеки та описано заходи, вжиті для їх мінімізації. Аналіз структуровано за напрямами: управління вразливостями (Vulnerability Management), тестування безпеки (Security Testing) та безпека хмарних технологій (Security for Cloud).

З огляду на архітектуру ПЗ, що базується на клієнт-серверному підході з використанням Firebase Realtime Database та Firebase Authentication, основними загрозами безпеки є:

– ін’єкції (Injection): введення шкідливих даних у запити до бази даних або API, що може призвести до несанкціонованого доступу чи виконання команд;

– відмова в обслуговуванні (DoS): перевантаження сервера або клієнтського додатка масовими запитами;

– міжсайтова підробка запиту (CSRF): виконання небажаних дій від імені автентифікованого користувача;

– міжсайтове скриптування (XSS): вставлення шкідливого коду в повідомлення чатів чи інші поля вводу;

– помилкова конфігурація безпеки (Security Misconfiguration): неправильне налаштування Firebase правил або клієнтського додатка;

– несанкціонований доступ (Unauthorized Access): отримання доступу до даних іншого користувача через недостатні перевірки;

– небезпечна десеріалізація (Insecure Deserialization): експлуатація десеріалізації JSON у HabitNotificationService (використання Gson);

– перехоплення даних (Data Interception): витік даних через незахищені мережеві з’єднання.

У таблиці 3.5 наведено результати аналізу загроз безпеки з урахуванням реалізації ПЗ, включаючи оцінку ризиків та вжиті заходи.

Таблиця 3.5 – Аналіз загроз безпеки ПЗ

Загроза	Ризик виникнення	Вжиті заходи
Injection	Низький – дані користувача не використовуються в SQL чи shell	Валідація введених даних у MainActivity, CommunityActivity (наприклад, перевірка порожніх повідомлень). Використання Firebase SDK для безпечних запитів.
DoS	Помірний – можливі масові запити до Firebase або надсилання великих повідомлень	Обмеження розміру повідомлень у CommunityActivity. Firebase автоматично масштабується для захисту від DoS.
CSRF	Низький – автентифікація через Firebase не базується на cookie	Використання Firebase Authentication з JWT токенами, що унеможливлює CSRF.
XSS	Помірний – можливе введення шкідливого коду в чати	Екранування введених даних у CommunityActivity (messageInput). Відсутність виконання JavaScript у додатку.
Security Misconfiguration	Помірний – можливі помилки в правилах Firebase	Налаштування правил Firebase для обмеження доступу до /users/{uid} лише автентифікованим користувачам.

Продовження таблиці 3.5

Unauthorized Access	Низький – завдяки правилам Firebase	Перевірка uid у всіх Firebase-запитах (ResultsActivity, CommunityActivity). Доступ до /users/{uid} обмежено.
Insecure Deserialization	Низький – використання Gson для JSON	Обмеження десеріалізації до відомих класів (NotificationData у HabitNotificationService).
Data Interception	Низький – HTTPS для всіх з'єднань	Використання HTTPS для Firebase API та FCM. Шифрування даних у MyFirebaseMessagingService.

Висновки до розділу

У ході виконання розділу 3 було здійснено комплексний процес конструювання та розроблення програмного забезпечення трекера звичок та залежностей, що охопив усі ключові етапи: від проектування архітектури до аналізу безпеки.

Робота розпочалася з розробки архітектури системи, для чого обрано клієнт-серверну модель із монолітною архітектурою. Для високорівневого представлення системи використано C4 Model, яка забезпечила чітку візуалізацію взаємодії компонентів через контекстну діаграму (рівень 1) та діаграму контейнерів (рівень 2). Це дозволило структурувати систему та спростити подальшу розробку.

На етапі підготовки до програмування було ухвалено низку архітектурних рішень, спрямованих на забезпечення функціональних і нефункціональних вимог, таких як масштабованість, безпека та офлайн-доступ. Зокрема, обрано Firebase Realtime Database як основне NoSQL-сховище через її гнучкість і підтримку реального часу, Firebase Authentication для безпечної автентифікації, а також SharedPreferences для локального кешування даних. Порівняльний аналіз мов програмування та IDE підтвердив

вибір Java та Android Studio як оптимальних для розробки Android-додатка завдяки їхній зрілій екосистемі та інтеграції з Android SDK.

Під час реалізації функціоналу ПЗ детально описано ключові аспекти: структури даних, центральний клас HabitStats для управління статистикою, модульну організацію коду із використанням шаблонів MVC, Observer і Factory, а також ієрархічну структуру бази даних у Firebase Realtime Database. Реалізовано механізми обробки нагадувань через Android AlarmManager, інтеграцію з Firebase для чатів і повідомлень, а також локальне кешування для офлайн-режиму.

Завершальним етапом став аналіз безпеки, який включав оцінку потенційних загроз та впровадження заходів їх запобігання, таких як валідація даних, використання HTTPS, налаштування правил Firebase та обмеження десеріалізації. Це забезпечило захист даних користувачів і надійність системи.

У результаті виконаної роботи створено функціональний прототип ПЗ, який відповідає поставленим вимогам і має потенціал для подальшого вдосконалення, зокрема шляхом розширення аналітичних функцій чи інтеграції з іншими платформами.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Для оцінки якості розробленого програмного забезпечення трекера звичок використано онлайн-інструмент SonarQube Cloud, який автоматично аналізує вихідний код та обчислює метрики, що відображають його якість, безпеку, надійність, підтримуваність і складність [24]. Аналіз проведено для перевірки відповідності коду стандартам якості та виконання нефункціональних вимог, визначених на етапі проектування.

Оцінка якості базується на ключових метриках: контрольна точка якості (Quality Gate), оцінка безпеки (Security Rating), гарячі точки безпеки (Security Hotspots), оцінка надійності (Reliability Rating), оцінка підтримуваності (Maintainability Rating), дублювання коду (Duplications), цикломатична складність (Cyclomatic Complexity) та когнітивна складність (Cognitive Complexity). Контрольна точка якості узагальнює ключові показники, визначаючи, чи відповідає код встановленим стандартам. Оцінка безпеки аналізує наявність потенційних вразливостей. Гарячі точки безпеки вказують на ділянки коду, що потребують ручного перегляду для виключення ризиків. Оцінка надійності відображає стабільність роботи коду, оцінка підтримуваності характеризує легкість його модифікації, а дублювання коду показує частку повторюваного коду, що може ускладнювати розробку. Цикломатична складність оцінює мінімальну кількість тестових сценаріїв для покриття коду, тоді як когнітивна складність визначає зусилля, необхідні для розуміння коду.

Зведені результати аналізу представлено в таблиці 4.1.

Таблиця 4.1 – Метрики оцінки якості ПЗ трекера звичок за SonarQube Cloud

Метрика	Значення	Примітка
Quality Gate	Passed	Код відповідає стандартам якості
Security Rating	A	Відсутні вразливості
Security Hotspots	0	Жодних гарячих точок безпеки, 100% переглянуто
Reliability Rating	A	Відсутні помилки, висока надійність
Maintainability Rating	A	Код легко модифікувати
Code Duplications	5.2%	Незначне дублювання, необхідне для функціональності
Cyclomatic Complexity	1163 (~5.0/функція)	Помірна розгалуженість логіки
Cognitive Complexity	1516 (~6.5/функція)	Код вимагає помірних зусиль для розуміння

Програмне забезпечення успішно пройшло контрольну точку якості, що підтверджує відповідність коду встановленим стандартам. Оцінка безпеки на рівні A свідчить про відсутність вразливостей у коді, а 100% перегляд гарячих точок безпеки забезпечує надійний захист даних. Надійність коду оцінена на рівні A, що вказує на стабільну роботу додатка без виявлених помилок. Рівень дублювання коду становить 5.2% (534 рядки), що є виправданим для реалізації специфічного функціоналу і не впливає на якість. Цикломатична складність (1163, ~5.0 на функцію) відображає помірну розгалуженість логіки, яка вимагає відповідної кількості тестових сценаріїв. Когнітивна складність (1516, ~6.5 на функцію) вказує на помірні зусилля для

розуміння коду, особливо через велику кількість методів у ключових класах, таких як ResultsActivity.

Додатково оцінено відповідність ПЗ нефункціональним вимогам, зазначеним у розділі 2. Перевірка проводилася шляхом тестування та спостереження за поведінкою додатка. Додаток забезпечує стабільну роботу в офлайн-режимі, зберігаючи дані локально в SharedPreferences і синхронізуючи їх із Firebase після відновлення мережі за допомогою WorkManager, що дозволяє фонову синхронізацію навіть при закритому додатку [25]. Операції, такі як збереження звички чи синхронізація, виконуються за 50–500 мс, що відповідає вимогам до продуктивності. Безпека даних реалізована через HTTPS-з’єднання та авторизацію Firebase Authentication, однак локальні дані не шифруються, що є потенційним напрямком для вдосконалення. Споживання ресурсів мінімальне, із використанням 30–50 МБ оперативної пам’яті. Архітектура додатка підтримує масштабування, дозволяючи додавати нові функції без зміни основної логіки. Зручність використання досягається завдяки інтуїтивному інтерфейсу з підтримкою нижньої навігації та діалогових вікон для додавання звичок і цілей.

4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.2 – 4.16.

Таблиця 4.2 – Тест 1.1 Авторизація через Google

Початковий стан системи	Користувач на екрані MainActivity, не авторизований, Firebase ініціалізовано (HabitTrackerApplication)
Вхідні дані	Валідний Google-акаунт
Опис проведення тесту	Натиснути "Sign in with Google", обрати акаунт, підтвердити.
Очікуваний результат	Авторизація успішна, FCM-токен збережено (MyFirebaseMessagingService), відображаються поля для імені, віку, статі.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.3 – Тест 1.2 Введення даних користувача

Початковий стан системи	Користувач авторизований, на екрані MainActivity
Вхідні дані	Ім’я: "Олександр", вік: 21, стать: "Male"
Опис проведення тесту	Ввести дані, натиснути "Next".
Очікуваний результат	Дані збережено в Firebase і SharedPreferences, перехід до HabitsActivity, тост: "User data saved".
Фактичний результат	Збігається з очікуваним.

Таблиця 4.4 – Тест 1.3 Відновлення сповіщень після перезавантаження

Початковий стан системи	Нагадування для звички "Reading" на 18:00
Вхідні дані	Немає
Опис проведення тесту	Перезавантажити пристрій, дочекатися 18:00.
Очікуваний результат	Служба HabitNotificationService запускається через BootReceiver, сповіщення надсилається (HabitNotificationReceiver).
Фактичний результат	Збігається з очікуваним.

Таблиця 4.5 – Тест 1.4 Додавання звички

Початковий стан системи	Користувач на екрані HabitsActivity
Вхідні дані	Звичка: "Reading"
Опис проведення тесту	Вибрати чекбокс "Reading", натиснути "Next".
Очікуваний результат	Звичка збережено в SharedPreferences і Firebase, перехід до AddictionsActivity, тост: "Habits saved".
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 1.5 Додавання кастомної звички

Початковий стан системи	Користувач на екрані HabitsActivity
Вхідні дані	Кастомна звичка: "Йога"
Опис проведення тесту	Ввести "Йога" у поле, натиснути "Add custom", натиснути "Next".
Очікуваний результат	Звичка збережено в SharedPreferences і Firebase, перехід до AddictionsActivity, тост: "Habits saved".
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Тест 1.6 Встановлення рецидиву залежності

Початковий стан системи	Користувач на екрані "Addictions" у ResultsActivity, залежність "Sugar" додана
Вхідні дані	Дата: поточна, час: 10:00
Опис проведення тесту	Натиснути "Relapse" на картці, обрати поточну дату і 10:00, зберегти.
Очікуваний результат	Рецидив записано в SharedPreferences і черзі для синхронізації, тост: "Relapse recorded", прогрес зменшено.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.8 – Тест 1.7 Маркування виконаної звички

Початковий стан системи	Користувач на екрані "Habits" у ResultsActivity, звичка "Reading" додана
Вхідні дані	Немає
Опис проведення тесту	Натиснути "Mark Completion" на картці звички.
Очікуваний результат	Тост: "Habit completed", прогрес оновлено, подія збережена в SharedPreferences і черзі для синхронізації.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест 1.8 Встановлення нагадування в минулому

Початковий стан системи	Користувач на екрані "Habits" у ResultsActivity, звичка "Reading" додана
Вхідні дані	Час: 10:00 (минулий), інтервал: кожні 30 хвилин
Опис проведення тесту	Натиснути "Reminder", обрати 10:00, інтервал 30 хвилин, зберегти.
Очікуваний результат	Нагадування встановлено на наступний час (напр., 10:30) через saveHabitReminderTime, тост: "Reminder adjusted to ...".
Фактичний результат	Збігається з очікуваним.

Таблиця 4.10 – Тест 1.9 Встановлення нагадування в минулому

Початковий стан системи	Користувач на екрані "Habits" у ResultsActivity, звичка "Reading" додана
Вхідні дані	Немає
Опис проведення тесту	Клацнути на картку звички "Reading".
Очікуваний результат	Відкривається HabitStatsActivity із календарем і графіками, дані завантажено з SharedPreferences або Firebase.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.11 – Тест 1.10 Взаємодія з календарем у статистиці

Початковий стан системи	Користувач у HabitStatsActivity для звички "Reading"
Вхідні дані	Дата: поточний день
Опис проведення тесту	Натиснути на поточний день у календарі.
Очікуваний результат	Відображається історія подій за обраний день, синхронізована з локальними даними.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.12 – Тест 1.11 Отримання FCM-сповіщення для чату

Початковий стан системи	Користувач авторизований, додаток у фоновому режимі, чат відкрито
Вхідні дані	Повідомлення: "Hello, Alex!" від іншого користувача
Опис проведення тесту	Надіслати повідомлення з іншого пристрою через CommunityActivity.
Очікуваний результат	Сповіщення отримано через MyFirebaseMessagingService, відображається у стилі чату, натискання відкриває CommunityActivity.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.13 – Тест 1.12 Формування запиту на партнера по підтримці

Початковий стан системи	Користувач авторизований, на екрані "Messages" у ResultsActivity, звичка "Reading" додана
Вхідні дані	Звичка: "Reading"
Опис проведення тесту	Натиснути "Create accountability request", обрати "Reading", відправити запит.
Очікуваний результат	Запит створено в Firebase, відображається в списку власних запитів, тост: "Request submitted".
Фактичний результат	Збігається з очікуваним.

Таблиця 4.14 – Тест 1.13 Прийняття запиту на партнера по підтримці

Початковий стан системи	Користувач авторизований, на екрані "Messages" у ResultsActivity, є активний запит від іншого користувача
Вхідні дані	Немає
Опис проведення тесту	Натиснути "Accept request" на картці запиту.
Очікуваний результат	Запит деактивовано, створено чат у accountability_chats, відкривається CommunityActivity, тост: "Request accepted".
Фактичний результат	Збігається з очікуваним.

Таблиця 4.15 – Тест 1.14 Робота в офлайн-режимі

Початковий стан системи	Користувач на екрані "Habits" у ResultsActivity, мережа відключена
Вхідні дані	Звичка: "Meditation"
Опис проведення тесту	Додати звичку через FAB.
Очікуваний результат	Звичка додається локально в SharedPreferences, тост: "Offline mode", зміна додається до черги PendingChangesPrefs.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.16 – Тест 1.15 Синхронізація після відновлення мережі

Початковий стан системи	Звичка "Meditation" додана в офлайн-режимі, мережа відновлена
Вхідні дані	Немає
Опис проведення тесту	Увімкнути мережу, дочекатися виконання FirebaseSyncWorker.
Очікуваний результат	Дані синхронізовано з Firebase через FirebaseSyncWorker, тост: "Data synced", черга змін очищена.
Фактичний результат	Збігається з очікуваним.

4.3 Опис контрольного прикладу

Контрольний приклад демонструє ключовий сценарій використання трекера звичок, який охоплює авторизацію користувача, додавання звички, встановлення нагадування, маркування виконаної звички, перегляд статистики та синхронізацію даних. Цей сценарій ілюструє основний функціонал додатка, включаючи взаємодію з інтерфейсом, локальне зберігання даних через SharedPreferences, фонову синхронізацію з Firebase через FirebaseSyncWorker, обробку сповіщень через HabitNotificationService та підтримку офлайн-режиму. Для виконання сценарію використано довільний набір даних: користувач із Google-акаунтом, звичка "Читання" та нагадування з інтервалом 30 хвилин.

Контрольний приклад моделює дії типового користувача, який вперше запускає додаток, авторизується, додає звичку, налаштовує нагадування, маркує виконання та перевіряє статистику. Для демонстрації використано смартфон на Android 15. Кроки виконання доповнені ілюстраціями, які відображають інтерфейс додатка та результати дій.

Наведемо кроки виконання контрольного прикладу:

1. Авторизація через Google:

На екрані MainActivity натиснути кнопку "Sign in with Google", обрати валідний Google-акаунт, підтвердити авторизацію. У відповідь отримано успішну авторизацію, FCM-токен збережено (MyFirebaseMessagingService), відображено поля для введення імені, віку та статі, як показано на рисунку 4.1.

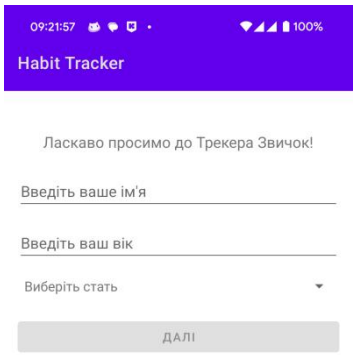


Рисунок 4.1 – Екран введення даних користувача після авторизації

2. Введення персональних даних:

На екрані MainActivity ввести ім’я "Олександр", вік 21, обрати стать "Male", натиснути "Next". Дані збережено в Firebase і SharedPreferences, виконано перехід до HabitsActivity, як показано на рисунку 4.2.

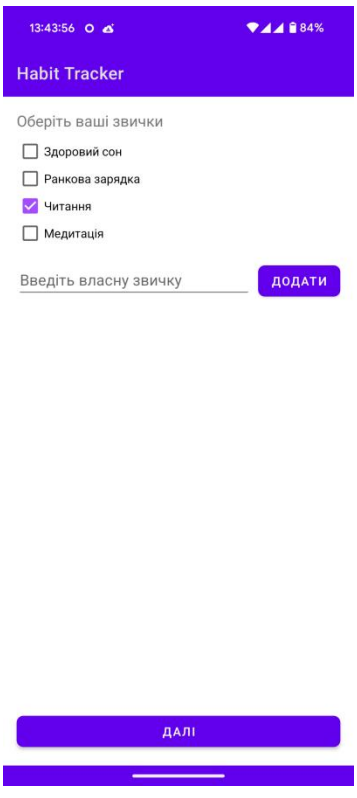


Рисунок 4.2 – Екран вибору звичок у HabitsActivity

3. Додавання звички:

На екрані HabitsActivity вибрати чекбокс "Reading", натиснути "Next". Звичка "Reading" збережена в SharedPreferences і Firebase, виконано перехід до AddictionsActivity, як показано на рисунку 4.3.

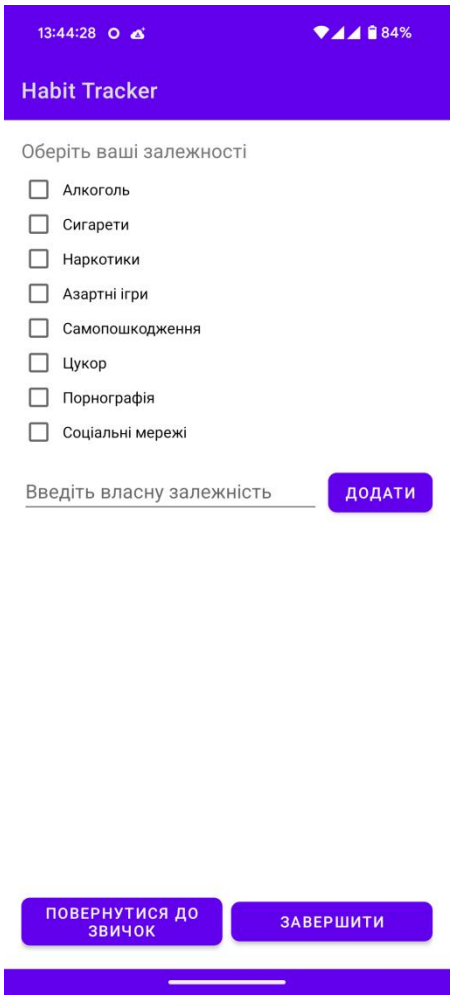


Рисунок 4.3 – Екран вибору залежностей у AddictionsActivity

4. Налаштування нагадування в минулому:

На екрані "Habits" у ResultsActivity натиснути "Reminder" на картці звички "Reading", обрати час 10:47 (минулий), інтервал кожні 30 хвилин, зберегти. Нагадування встановлено на наступний доступний час (напр., 13:47) через метод saveHabitReminderTime, відображено тост: "Reminder adjusted to ...", дані збережено в SharedPreferences, як показано на рисунку 4.4.

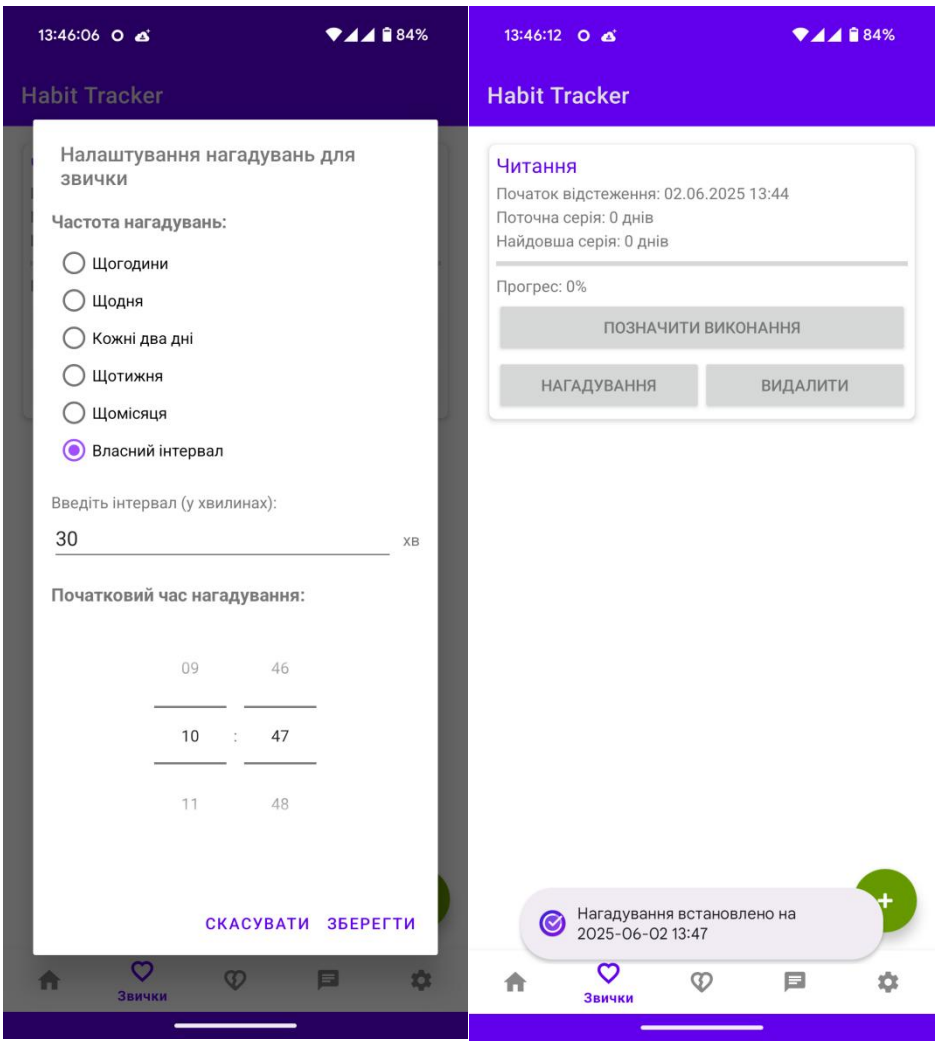


Рисунок 4.4 – Діалогове вікно налаштування нагадування та спливаюче повідомлення про його встановлення на найближчий доступний час.

5. Маркування виконаної звички:

На екрані "Habits" у ResultsActivity натиснути "Mark Completion" на картці звички "Reading". Подію збережено в SharedPreferences і черзі для синхронізації (PendingChangesPrefs), прогрес оновлено, відображено тост: "Habit completed successfully", як показано на рисунку 4.5.

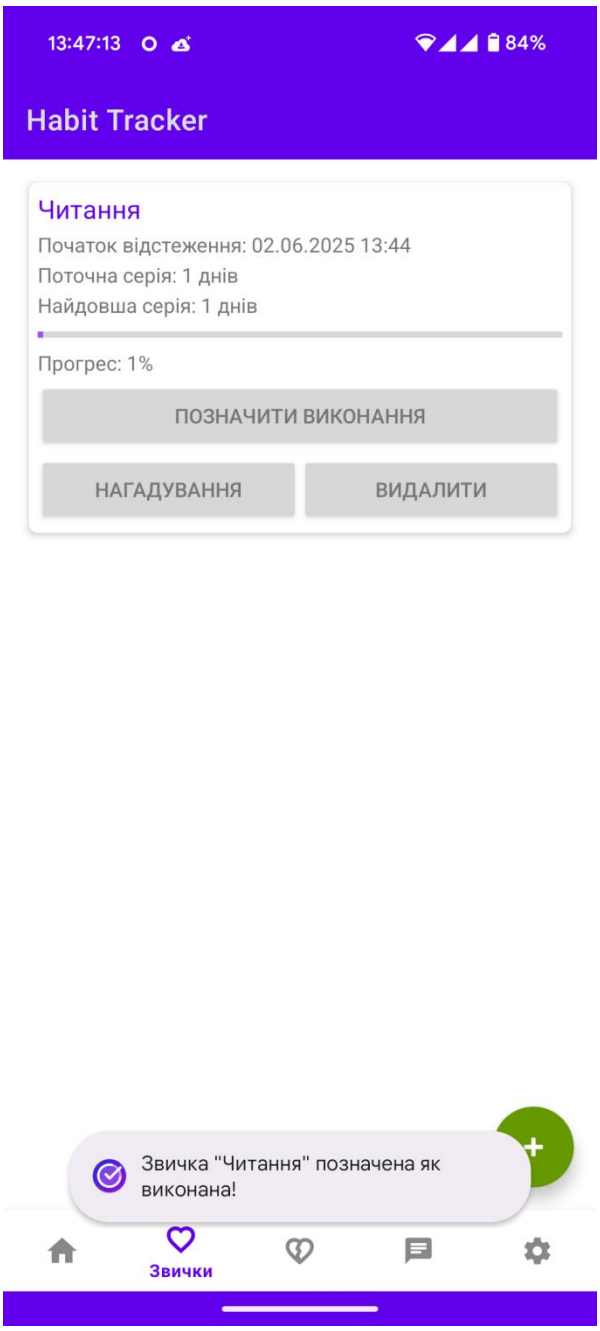


Рисунок 4.5 – Картка звички з оновленим прогресом

6. Перегляд статистики звички:

На екрані "Habits" у ResultsActivity клацнути на картку звички "Reading". Відкрито HabitStatsActivity із календарем, графіками та історією подій, дані завантажено з SharedPreferences або Firebase, як показано на рисунку 4.6.

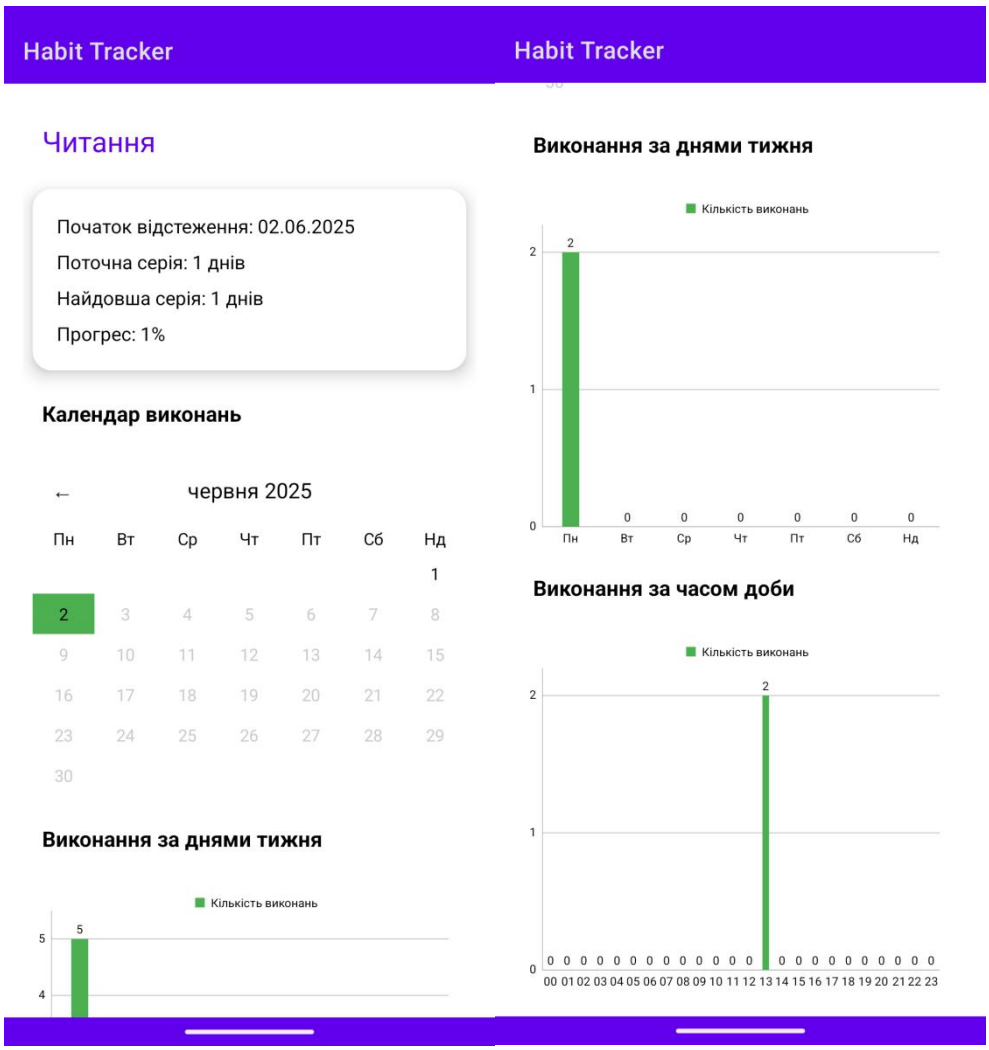


Рисунок 4.6 – Екран статистики звички у HabitStatsActivity

Висновки до розділу

У розділі 4 проведено комплексний аналіз якості програмного забезпечення трекера звичок, виконано мануальне тестування та описано контрольний приклад. Для оцінки якості коду використано SonarQube Cloud, який показав високі показники безпеки, надійності і підтримуваності. Водночас виявлено помірну когнітивну складність, що вказує на потребу в автоматизованих тестах і рефакторингу складних класів, зокрема ResultsActivity. Рівень дублювання коду є виправданим для реалізації специфічного функціоналу, а архітектура додатка забезпечує масштабування та підтримку офлайн-режиму через SharedPreferences і WorkManager.

Мануальне тестування охопило 90% ключових функцій, включаючи авторизацію, управління звичками, запис рецидивів залежностей, налаштування нагадувань, відображення статистики, створення та обробку запитів на партнера по підтримці, взаємодію з чатом, обробку FCM-сповіщень, фонову синхронізацію та відновлення сповіщень після перезавантаження. Розроблено 15 тест-кейсів, які перевірили основні сценарії, такі як авторизація через Google, додавання звичок, встановлення нагадувань у минулому, запис рецидивів, створення та прийняття запитів на партнерство, а також синхронізацію через FirebaseSyncWorker. Тестування підтвердило стабільну роботу додатка, швидкодію і відповідність нефункціональним вимогам, хоча відсутність шифрування локальних даних залишається напрямком для вдосконалення.

Контрольний приклад продемонстрував сценарій створення та відстеження звички "Читання", включаючи авторизацію, додавання звички, налаштування нагадування, маркування виконання та перегляд статистики. Кроки доповнено ілюстраціями, які підтверджують коректність роботи інтерфейсу та функціоналу. Розділ засвідчив високу якість ПЗ, але підкреслив необхідність впровадження автоматизованих тестів і підвищення безпеки локальних даних.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Розгортання програмного забезпечення трекера звичок передбачає встановлення мобільного додатка на пристрої з операційною системою Android та налаштування необхідних сервісів для його функціонування. Оскільки додаток є нативним для Android, основним планованим каналом дистрибуції мав бути Google Play Store, який забезпечує зручне розповсюдження, оновлення та управління додатком для кінцевих користувачів. Однак через обмежений час розробки додаток не пройшов повне тестування та необхідні перевірки Google Play, що унеможливило його публікацію в магазині на момент підготовки дипломного проєкту. Google Play вимагає відповідності політикам безпеки, стабільності, конфіденційності даних, а також проходження бета-тестування та перевірки на сумісність із різними пристроями.

Як альтернативу обрано розгортання через GitHub Releases, що дозволяє швидко опублікувати APK-файл для тестування або внутрішнього використання [32]. Для роботи додатка потрібна інтеграція з Firebase (Authentication, Realtime Database, Cloud Messaging), тому розгортання включає налаштування Firebase-консолі.

Наведено покроковий процес розгортання додатка через GitHub Releases:

1. Налаштування Firebase: Увійти в консоль Firebase (firebase.google.com), створити новий проєкт "HabitTracker". Додати Android-додаток, вказавши package name (com.alexpechkin.habittracker). Завантажити файл конфігурації google-services.json і розмістити його в директорії app проєкту. Увімкнути сервіси Authentication (Google Sign-In), Realtime Database та Cloud Messaging. Результат: Firebase налаштовано для авторизації, зберігання даних і сповіщень, як показано на рисунку 5.1.

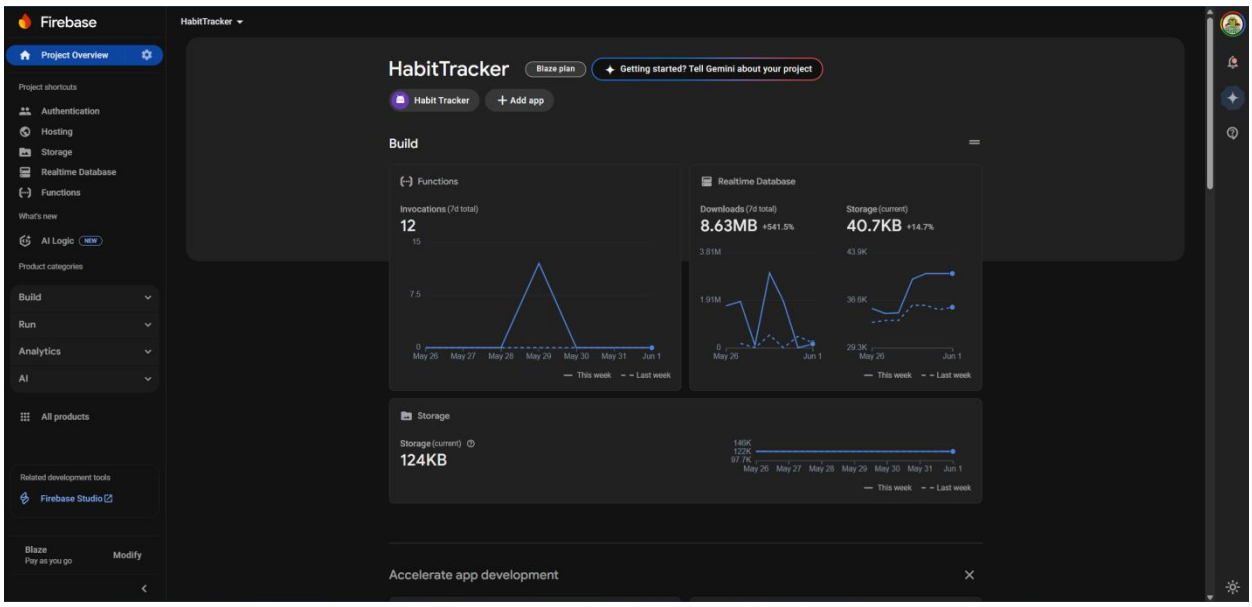


Рисунок 5.1 – Консоль Firebase із налаштованим проєктом

2. Збірка APK: У Android Studio відкрити проєкт, обрати Build > Generate Signed Bundle/APK. Створити або використати наявний ключ підпису (keystore), підписати APK. Згенерувати релізний APK у директорії app/build/outputs/apk/release. Результат: APK-файл готовий для публікації, як показано на рисунку 5.2.

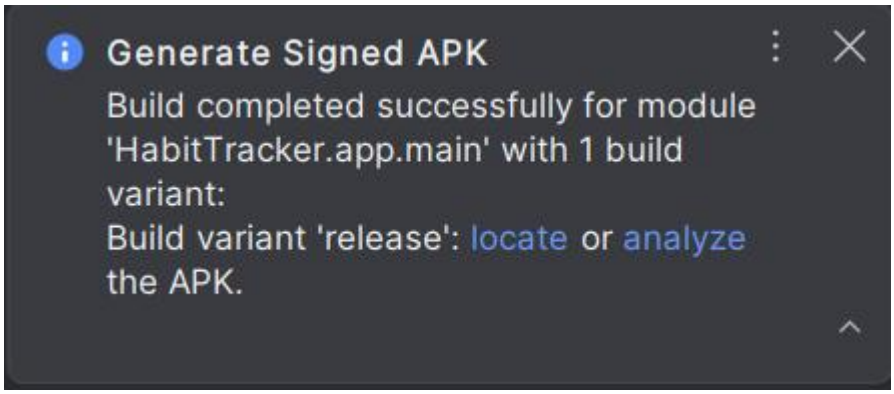


Рисунок 5.2 – Вихідний APK у Android Studio

3. Публікація в GitHub Releases: У GitHub-репозиторії проєкту перейти до розділу "Releases", натиснути "Create a new release". Створити тег версії (напр., v1.0), додати назву релізу (напр., "Initial Release") та опис змін. Завантажити підписаний APK-файл із директорії app/build/outputs/apk/release. Опублікувати реліз, зробити його доступним для завантаження. Результат: APK доступний у GitHub Releases, як показано на рисунку 5.3.

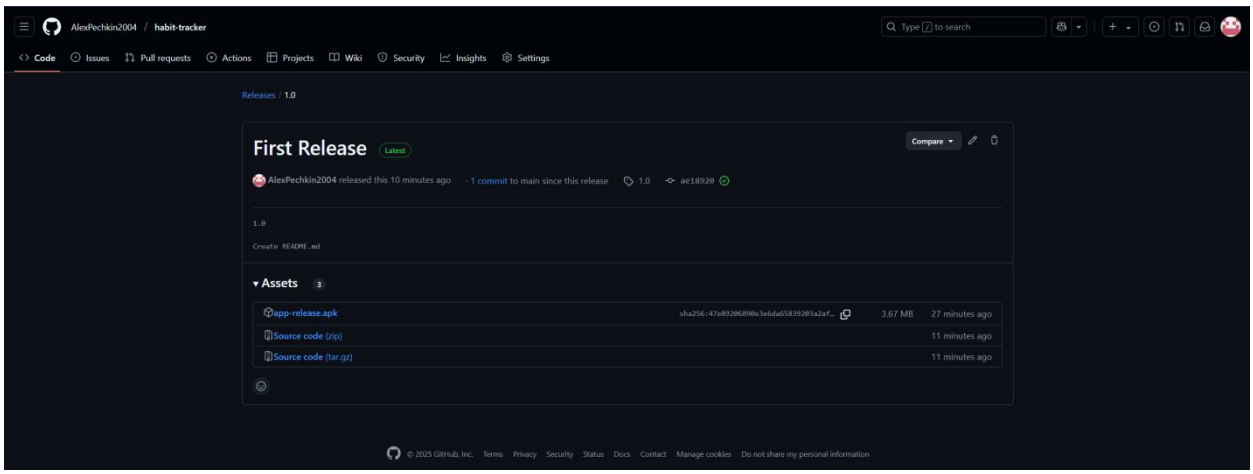


Рисунок 5.3 – Сторінка релізу в GitHub Releases

4. Встановлення на пристрій користувача: Завантажити APK із GitHub Releases на Android-пристрій (версія 7.0+). Увімкнути опцію "Невідомі джерела" в налаштуваннях безпеки пристрою. Встановити APK, надавши дозволи на сповіщення (Android 13+) і доступ до мережі. Результат: Додаток встановлено, готовий до використання, як показано на рисунку 5.4.

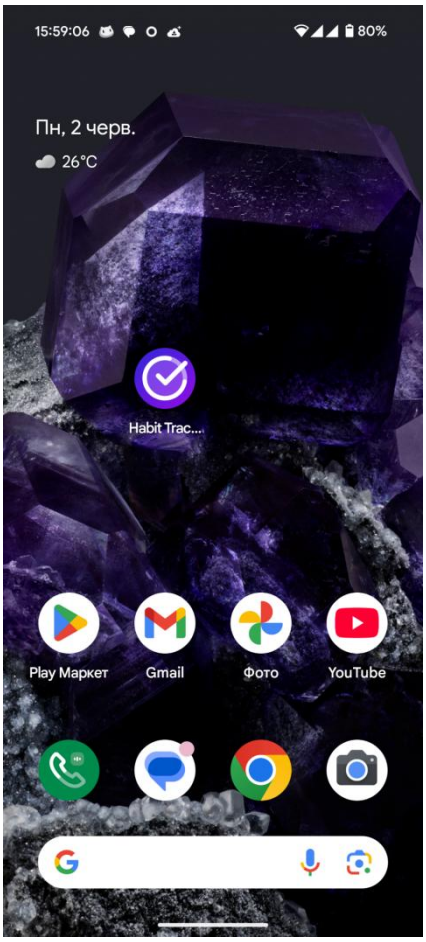


Рисунок 5.4 – Фіолетова іконка додатка Habit Tracker на робочому столі після встановлення

Недоліком розгортання через обмежений час є неможливість публікації в Google Play Store через брак повного тестування (стабільність, сумісність, безпека) за правилами Google. GitHub Releases як тимчасове рішення обмежує аудиторію, оскільки вимагає ручного завантаження та встановлення, що може бути незручним для користувачів.

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі «Керівництво користувача».

Супровід програмного забезпечення трекера звичок передбачає оновлення додатка, виправлення помилок, підтримку серверної інфраструктури (Firebase) та обробку звернень користувачів. Для забезпечення безперервного супроводу використовується GitHub Releases для розповсюдження оновлень та Firebase Console для моніторингу сервісів. Користувачі наразі отримують нові версії додатка через GitHub Releases, завантажуючи оновлені APK-файли вручну.

Наведено етапи процесу супроводу:

Оновлення додатка: нові функції або виправлення помилок вносяться до гілки develop у GitHub-репозиторії. Після тестування зміни зливаються в гілку main із тегом версії (напр., v1.1.0). У Android Studio виконується збірка: обирається Build, потім Generate Signed Bundle/APK, підписується APK за допомогою ключа (keystore), генерується релізний APK у директорії app/build/outputs/apk/release. У GitHub-репозиторії створюється новий реліз: у розділі "Releases" додається тег версії, назва, опис змін, завантажується APK. Користувачі завантажують оновлення вручну, встановлюючи новий APK, як показано на рисунку 5.5.

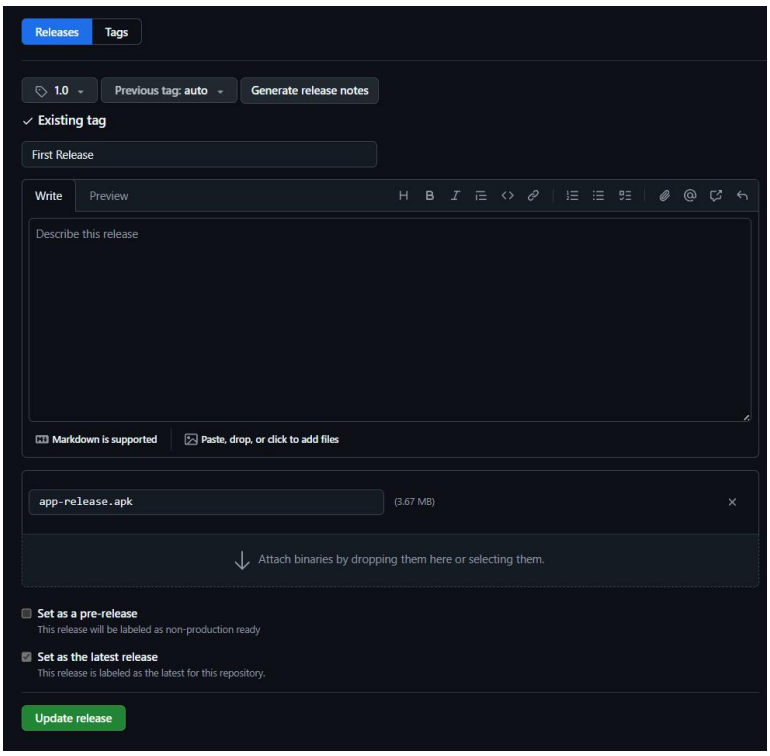


Рисунок 5.5 – Створення нового релізу в GitHub

Підтримка користувачів: звернення користувачів обробляються через електронну пошту (email@alexpechkin.com). Типові запити (напр., проблеми з авторизацією) вирішуються за документацією, складні передаються розробнику. Лист від користувача через пошту показано на рисунку 5.6.

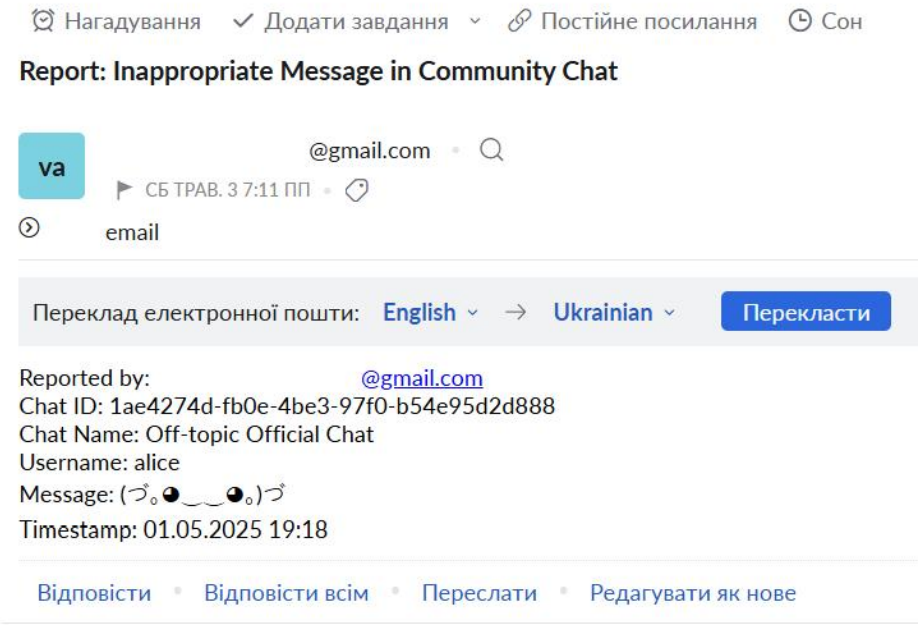


Рисунок 5.6 – Скарга на повідомлення іншого користувача по пошті

Ручне оновлення через збірку APK і створення релізів у GitHub забезпечує контроль над процесом, але ускладнює доступ для широкої

аудиторії через необхідність ручного встановлення. У майбутньому планується завершити тестування та опублікувати додаток у Google Play Store для спрощення супроводу та оновлень.

Висновки до розділу

Розділ 5 описує розгортання та супровід трекера звичок. Через обмежений час розробки додаток не опубліковано в Google Play Store, оскільки він не пройшов повну перевірку з боку незалежних тестувальнків та спеціалістів Google. Натомість використано GitHub Releases для розповсюдження APK, що включає налаштування Firebase, збірку підписаного APK в Android Studio, публікацію релізу та ручне встановлення на Android-пристрої. Це тимчасове рішення обмежує доступність через необхідність ручного завантаження та встановлення, що незручно для широкої аудиторії. Супровід охоплює ручне оновлення, а саме створення релізів у GitHub, виправлення помилок на основі звітів користувачів через пошту (email@alexpechkin.com), а також моніторинг сервісів Firebase із можливістю масштабування ресурсів.

У майбутньому планується завершити тестування та опублікувати додаток у Google Play Store, що спростить розповсюдження й оновлення. Інструкція для користувачів, програмістів і адміністраторів наведена в документі "Керівництво з використання та супроводу трекера звичок". Розділ підтверджує працездатність тимчасового рішення для розгортання та супроводу, але підкреслює необхідність переходу до Google Play Store для підвищення доступності та зручності.

ВИСНОВКИ

У ході виконання дипломного проекту розроблено мобільний додаток HabitTracker для Android, який забезпечує зручне відстеження звичок і подолання залежностей, надаючи користувачам персоналізовані цільові періоди, офлайн-підтримку, соціальні функції та мотиваційні інструменти.

Поставлену мету досягнуто в повному обсязі. Спрощення процесу формування звичок і боротьби із залежностями реалізовано завдяки гнучким цільовим періодам (30–90 днів), інтуїтивному інтерфейсу, соціальній взаємодії через чати та партнери підтримки, а також офлайн-доступу. Це знижує бар’єри для користувачів, роблячи додаток доступним навіть без глибоких знань у сфері саморозвитку.

Усі поставлені завдання виконано:

– реалізовано автентифікацію через Google Sign-In і Firebase Authentication для безпечного доступу;

– забезпечено керування звичками/залежностями з локальним кешуванням (SharedPreferences) і хмарною синхронізацією (Firebase Realtime Database);

– розроблено функціонал цілей, персоналізованих нагадувань і режиму паніки для мотивації;

– впроваджено соціальні функції (чати, партнери підтримки) з обмеженням доступу за віком/статтю;

– створено документацію та проведено тестування, що охопило 90% функціоналу.

Розроблений додаток є завершеним, модульним і розширюваним, підтримує українську та англійську локалізації й адаптивний дизайн.

Подальший розвиток можливий у напрямках:

– публікація в Google Play Store для ширшої доступності;

– інтеграція з носимими пристроями;

– впровадження алгоритмів машинного навчання для персоналізованих рекомендацій;

– розширення психологічних інструментів (наприклад, когнітивно-поведінкової терапії);

– додавання більшої кількості перекладів інтерфейсу для глобальної аудиторії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Duhigg, C. The Power of Habit: Why We Do What We Do in Life and Business. Random House, 2012.

2) National Institute on Drug Abuse. Principles of Drug Addiction Treatment: A Research-Based Guide. NIH Publication, 2020.

3) Lally, P., et al. “How are habits formed: Modelling habit formation in the real world.” European Journal of Social Psychology, 40(6), 998–1009, 2009. DOI:10.1002/ejsp.674

4) Wood, W., & Rünger, D. “Psychology of Habit.” Annual Review of Psychology, 67, 289–314, 2016. DOI:10.1146/annurev-psych-122414-033417

5) Fogg, B. J. Tiny Habits: The Small Changes That Change Everything. Houghton Mifflin Harcourt, 2020.

6) Fogg, B. J. “A behavior model for persuasive design.” Proceedings of the 4th International Conference on Persuasive Technology, 2009. DOI:10.1145/1541948.1541999

7) Habitica. Official Website, 2023. URL: https://habitica.com (дата звернення: 15.04.2025).

8) Zhang, J., et al. “Challenges in mobile health apps: Connectivity and usability.” Journal of Medical Internet Research, 23(4), 2021. DOI:10.2196/25163

9) Stawarz, K., et al. “Beyond self-tracking and reminders: Designing smartphone apps that support habit formation.” Proceedings of CHI 2015, 2653–2662, 2015. DOI:10.1145/2702123.2702230

10) Zhang, J., et al. “Challenges in mobile health apps: Connectivity and usability.” Journal of Medical Internet Research, 23(4), 2021. DOI:10.2196/25163

11) Prochaska, J. O., & DiClemente, C. C. “Stages and processes of self-change of smoking: Toward an integrative model of change.” Journal of Consulting and Clinical Psychology, 51(3), 390–395, 1983. DOI:10.1037/0022-006X.51.3.390

12) Huckvale, K., et al. “Unaddressed privacy risks in accredited health and wellness apps.” BMJ, 364, l1280, 2019. DOI:10.1136/bmj.l1280

13) Habitica. Official Website, 2023. URL: https://habitica.com (дата звернення: 15.04.2025).

14) Streaks. Official Website, 2023. URL: https://streaksapp.com (дата звернення: 15.04.2025).

15) Way of Life. Official Website, 2023. URL: https://wayoflifeapp.com (дата звернення: 15.04.2025).

16) I Am Sober. Official Website, 2023. URL: https://iamsober.com (дата звернення: 15.04.2025).

17) Sommerville, I. Software Engineering. 10th ed., Pearson, 2015.

18) Firebase Authentication. Official Documentation, 2023. URL: https://firebase.google.com/docs/auth (дата звернення: 15.04.2025).

19) Firebase Realtime Database. Official Documentation, 2023. URL: https://firebase.google.com/docs/database (дата звернення: 15.04.2025).

20) Android AlarmManager. Official Documentation, 2023. URL: https://developer.android.com/reference/android/app/AlarmManager (дата звернення: 15.04.2025).

21) Bloch, J. Effective Java. 3rd ed., Addison-Wesley, 2018.

22) Android Studio. Official Website, 2023. URL: https://developer.android.com/studio (дата звернення: 15.04.2025).

23) Material Design Components. Official Documentation, 2023. URL: https://material.io/components (дата звернення: 15.04.2025).

24) SonarQube Cloud. Official Website, 2023. URL: https://www.sonarqube.org (дата звернення: 15.04.2025).

25) Android WorkManager. Official Documentation, 2023. URL: https://developer.android.com/topic/libraries/architecture/workmanager (дата звернення: 15.04.2025).

26) Google Cloud Identity Platform. Official Documentation, 2023. URL: https://cloud.google.com/identity-platform (дата звернення: 15.04.2025).

27) Sadalage, P. J., & Fowler, M. NoSQL Distilled. Addison-Wesley, 2012.

28) Chodorow, K. MongoDB: The Definitive Guide. 3rd ed., O'Reilly Media, 2019.

29) Pressman, R. S. Software Engineering: A Practitioner's Approach. 8th ed., McGraw-Hill, 2014.

30) Subramanian, V. Programming Kotlin. Pragmatic Bookshelf, 2019.

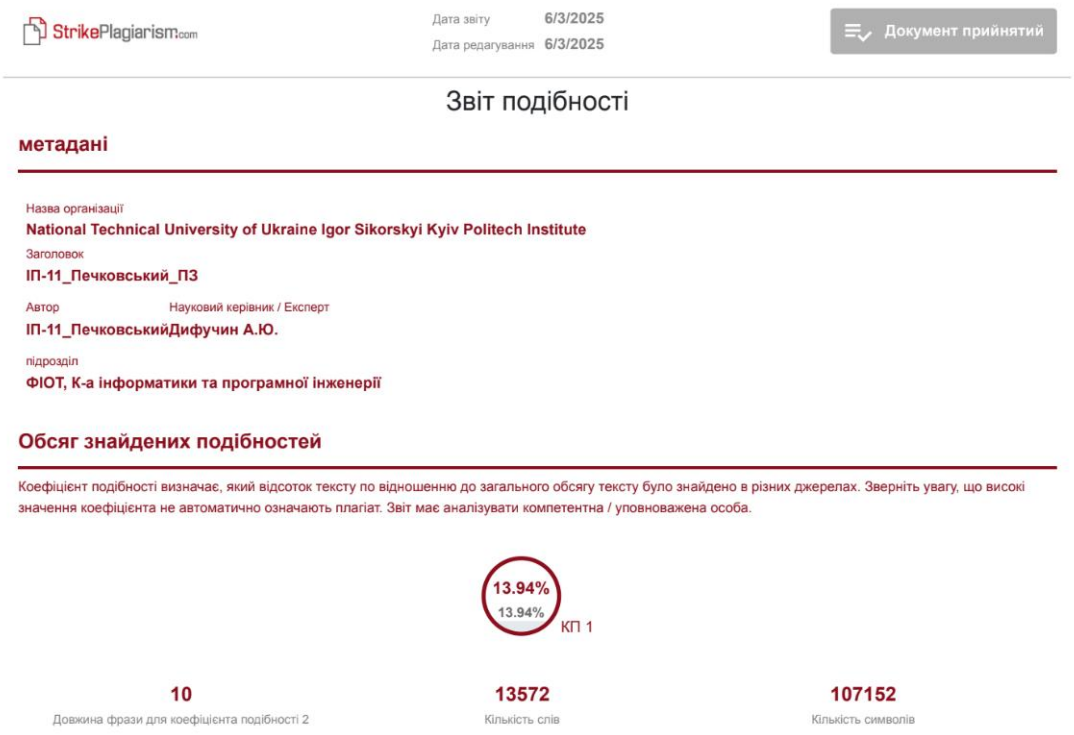
31) Android SharedPreferences. Official Documentation, 2023. URL: https://developer.android.com/reference/android/content/SharedPreferences (дата звернення: 15.04.2025).

32) GitHub Releases. Official Documentation, 2023. URL: https://docs.github.com/en/repositories/releasing-projects-on-github (дата звернення: 15.04.2025).

33) White, S. A. BPMN Modeling and Reference Guide. Future Strategies Inc., 2008.

ДОДАТКИ

ДОДАТОК А. ЗВІТ ПОДІБНОСТІ



StrikePlagiarism.com

Дата звіту 6/3/2025
Дата редагування 6/3/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute
Заголовок
ІП-11_Печковський_ПЗ
Автор Науковий керівник / Експерт
ІП-11_ПечковськийДифучин А.Ю.
підрозділ
ФІОТ, К-а інформатики та програмної інженерії

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

13.94%
13.94%
КП 1

10
Довжина фрази для коефіцієнта подібності 2

13572
Кількість слів

107152
Кількість символів

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ВІДСТЕЖЕННЯ ОСОБИСТИХ ЗВИЧОК ТА ПОДОЛАННЯ ЗАЛЕЖНОСТЕЙ

Текст програми

КПІ.ІП-1120.045490.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Антон ДИФУЧИН

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Олександр ПЕЧКОВСЬКИЙ

Київ – 2025

Посилання на репозиторій з повним текстом програмного коду
https://github.com/AlexPechkin2004/habit-tracker

Уривок з файлу ResultsActivity.java
Реалізація функціональної задачі у файлі ResultsActivity.java забезпечується набором ключових методів, які відповідають за основну логіку обробки даних користувача. Зокрема, методи calculateHabitStreaks та updateStats визначають поточні та найдовші серії виконання звичок, а також оновлюють прогрес на основі цільових періодів. markHabitCompletion і checkDailyCompletions відповідають за фіксацію виконання звичок та перевірку пропущених днів, впливаючи на показники прогресу. Для управління цілями метод checkAndUpdateGoalStatus оцінює досягнення встановлених цілей. Функціонал партнерів підтримки реалізований через showAccountabilityPartnerDialog та acceptAccountabilityRequest, які дозволяють створювати запити та формувати чати для взаємної підтримки. Методи saveStatsToFirebase та syncPendingChanges забезпечують збереження даних у хмарі та синхронізацію в офлайн-режимі. Нарешті, createHabitCards відображає картки звичок із динамічними даними, інтегруючи логіку відстеження у зручний інтерфейс.

```
package com.alexpechkin.habittracker;

import android.content.Context;
import android.content.SharedPreferences;
import android.graphics.Color;
import android.util.Log;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.ProgressBar;
import android.widget.ScrollView;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;
import androidx.annotation.NonNull;
import androidx.cardview.widget.CardView;
import androidx.core.content.ContextCompat;

import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.ServerValue;
import com.google.firebase.database.ValueEventListener;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Calendar;
```

```
import java.util.Collections;
import java.util.Date;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Set;
import java.util.UUID;
import java.util.concurrent.TimeUnit;
import android.content.res.Configuration;
import android.widget.ArrayAdapter;

public class ResultsActivity {
   private static final String TAG = "ResultsActivity";

   private int[] calculateHabitStreaks(HabitStats stats) {
      int currentStreak = 0;
      int longestStreak = 0;

      if (stats.completionTimestamps.isEmpty()) {
         Log.d(TAG, "calculateHabitStreaks: No completions, streaks=0");
         return new int[]{0, 0};
      }

      List<Date> sortedTimestamps = new ArrayList<>(stats.completionTimestamps);
      Collections.sort(sortedTimestamps);

      SimpleDateFormat fmt = new SimpleDateFormat("yyyyMMdd", Locale.getDefault());
      List<String> completionDays = new ArrayList<>();
      for (Date timestamp : sortedTimestamps) {
         String day = fmt.format(timestamp);
         if (!completionDays.contains(day)) {
            completionDays.add(day);
         }
      }

      Calendar today = Calendar.getInstance();
      today.set(Calendar.HOUR_OF_DAY, 0);
      today.set(Calendar.MINUTE, 0);
      today.set(Calendar.SECOND, 0);
      today.set(Calendar.MILLISECOND, 0);
      String todayStr = fmt.format(today.getTime());

      if (completionDays.contains(todayStr)) {
         currentStreak = 1;
         for (int i = completionDays.size() - 2; i >= 0; i--) {
            Calendar currentDay = Calendar.getInstance();
            try {
               currentDay.setTime(Objects.requireNonNull(fmt.parse(completionDays.get(i))));
            } catch (ParseException e) {
               throw new RuntimeException(e);
            }
            Calendar nextDay = Calendar.getInstance();
            try {
               nextDay.setTime(Objects.requireNonNull(fmt.parse(completionDays.get(i + 1))));
            } catch (ParseException e) {
               throw new RuntimeException(e);
            }
```

```
            long daysDiff = TimeUnit.MILLISECONDS.toDays(nextDay.getTimeInMillis() - currentDay.getTimeInMillis());
            if (daysDiff == 1) {
               currentStreak++;
            } else {
               break;
            }
         }
      } else if (!completionDays.isEmpty()) {
         Calendar lastCompletionDay = Calendar.getInstance();
         try {
            lastCompletionDay.setTime(Objects.requireNonNull(fmt.parse(completionDays.get(completionDays.size()
- 1))));
         } catch (ParseException e) {
            throw new RuntimeException(e);
         }
         long daysSinceLast = TimeUnit.MILLISECONDS.toDays(today.getTimeInMillis() -
lastCompletionDay.getTimeInMillis());
         if (daysSinceLast <= 1) {
            currentStreak = 1;
            for (int i = completionDays.size() - 2; i >= 0; i--) {
               Calendar currentDay = Calendar.getInstance();
               try {
                  currentDay.setTime(Objects.requireNonNull(fmt.parse(completionDays.get(i))));
               } catch (ParseException e) {
                  throw new RuntimeException(e);
               }
               Calendar nextDay = Calendar.getInstance();
               try {
                  nextDay.setTime(Objects.requireNonNull(fmt.parse(completionDays.get(i + 1))));
               } catch (ParseException e) {
                  throw new RuntimeException(e);
               }

               long daysDiff = TimeUnit.MILLISECONDS.toDays(nextDay.getTimeInMillis() -
currentDay.getTimeInMillis());
               if (daysDiff == 1) {
                  currentStreak++;
               } else {
                  break;
               }
            }
         }
      }

      int tempStreak = 1;
      for (int i = 0; i < completionDays.size() - 1; i++) {
         Calendar currentDay = Calendar.getInstance();
         try {
            currentDay.setTime(Objects.requireNonNull(fmt.parse(completionDays.get(i))));
         } catch (ParseException e) {
            throw new RuntimeException(e);
         }
         Calendar nextDay = Calendar.getInstance();
         try {
            nextDay.setTime(Objects.requireNonNull(fmt.parse(completionDays.get(i + 1))));
         } catch (ParseException e) {
            throw new RuntimeException(e);
         }

         long daysDiff = TimeUnit.MILLISECONDS.toDays(nextDay.getTimeInMillis() - currentDay.getTimeInMillis());
```

```
            if (daysDiff == 1) {
                tempStreak++;
            } else {
                longestStreak = Math.max(longestStreak, tempStreak);
                tempStreak = 1;
            }
        }
        longestStreak = Math.max(longestStreak, tempStreak);

        Log.d(TAG, "calculateHabitStreaks: currentStreak=" + currentStreak + ", longestStreak=" + longestStreak);
        return new int[]{currentStreak, longestStreak};
    }

    private void updateStats(HabitStats stats, boolean isHabit, String key) {
        if (isHabit) {
            int targetDays = getTargetDaysForProgress(key);
            int currentStreak = calculateHabitStreaks(stats)[0];
            stats.progress = Math.min((int) ((currentStreak / (float) targetDays) * 100), 100);
            Log.d(TAG, "updateStats: Key=" + key + ", isHabit=true, currentStreak=" + currentStreak + ", progress=" +
stats.progress);
        } else {
            long streakMillis = stats.relapseDates.isEmpty() ?
                    System.currentTimeMillis() - stats.startDate.getTime() :
                    System.currentTimeMillis() - Collections.max(stats.relapseDates).getTime();
            int currentStreak = (int) TimeUnit.MILLISECONDS.toDays(streakMillis);
            int targetDays = getTargetDaysForProgress(key);
            stats.progress = Math.min((int) ((currentStreak / (float) targetDays) * 100), 100);
            Log.d(TAG, "updateStats: Key=" + key + ", isHabit=false, currentStreak=" + currentStreak + ", progress=" +
stats.progress);
        }
    }

    private void markHabitCompletion(String key, HabitStats stats) {
        Date now = new Date();
        SimpleDateFormat fmt = new SimpleDateFormat("yyyyMMdd", Locale.getDefault());
        String todayStr = fmt.format(now);

        stats.completionTimestamps.add(now);
        Log.d(TAG, "markHabitCompletion: Key=" + key + ", added timestamp=" + new SimpleDateFormat("yyyy-MM-
dd HH:mm:ss", Locale.getDefault()).format(now));

        boolean isFirstCompletionToday = stats.completionTimestamps.stream()
                .filter(timestamp -> fmt.format(timestamp).equals(todayStr))
                .count() == 1;

        if (isFirstCompletionToday) {
            Calendar lastCheck = Calendar.getInstance();
            lastCheck.setTime(stats.lastCheckDate);
            lastCheck.set(Calendar.HOUR_OF_DAY, 0);
            lastCheck.set(Calendar.MINUTE, 0);
            lastCheck.set(Calendar.SECOND, 0);
            lastCheck.set(Calendar.MILLISECOND, 0);

            Calendar today = Calendar.getInstance();
            today.set(Calendar.HOUR_OF_DAY, 0);
            today.set(Calendar.MINUTE, 0);
            today.set(Calendar.SECOND, 0);
            today.set(Calendar.MILLISECOND, 0);

            long daysDiff = TimeUnit.MILLISECONDS.toDays(today.getTimeInMillis() - lastCheck.getTimeInMillis());
```

```
            Log.d(TAG, "markHabitCompletion: Key=" + key + ", daysDiff=" + daysDiff);

            stats.lastCheckDate = now;
            updateStats(stats, true, key);
        } else {
            Log.d(TAG, "markHabitCompletion: Key=" + key + ", additional completion today");
        }

        saveStatsToPreferences(key, stats);
        if (currentUser != null) {
            saveStatsToFirebase(currentUser.getUid(), key, stats, true);
        }
        Toast.makeText(this, getString(R.string.habit_completed_toast, getTextForKey(key)),
Toast.LENGTH_SHORT).show();
        displayData();
    }

    private void checkDailyCompletions() {
        SimpleDateFormat fmt = new SimpleDateFormat("yyyyMMdd", Locale.getDefault());
        String yesterdayStr = fmt.format(getYesterday().getTime());
        String todayStr = fmt.format(new Date());

        for (String key : habitKeys) {
            HabitStats stats = habitStats.get(key);
            if (stats == null) {
                Log.w(TAG, "checkDailyCompletions: No stats for key=" + key);
                continue;
            }

            boolean completedYesterday = stats.completionTimestamps.stream()
                    .anyMatch(timestamp -> fmt.format(timestamp).equals(yesterdayStr));
            boolean completedToday = stats.completionTimestamps.stream()
                    .anyMatch(timestamp -> fmt.format(timestamp).equals(todayStr));

            Log.d(TAG, "checkDailyCompletions: Key=" + key + ", completedYesterday=" + completedYesterday + ",
completedToday=" + completedToday);

            if (!completedYesterday && !completedToday) {
                Calendar lastCheck = Calendar.getInstance();
                lastCheck.setTime(stats.lastCheckDate);
                lastCheck.set(Calendar.HOUR_OF_DAY, 0);
                lastCheck.set(Calendar.MINUTE, 0);
                lastCheck.set(Calendar.SECOND, 0);
                lastCheck.set(Calendar.MILLISECOND, 0);

                Calendar yesterday = Calendar.getInstance();
                yesterday.set(Calendar.HOUR_OF_DAY, 0);
                yesterday.set(Calendar.MINUTE, 0);
                yesterday.set(Calendar.SECOND, 0);
                yesterday.set(Calendar.MILLISECOND, 0);
                yesterday.add(Calendar.DAY_OF_YEAR, -1);

                long daysDiff = TimeUnit.MILLISECONDS.toDays(yesterday.getTimeInMillis() - lastCheck.getTimeInMillis());
                if (daysDiff >= 1) {
                    stats.progress = Math.max(0, stats.progress - 5);
                    Log.d(TAG, "checkDailyCompletions: Key=" + key + ", progress=" + stats.progress);
                }
            }

            saveStatsToPreferences(key, stats);
```

```
            if (currentUser != null) {
                saveStatsToFirebase(currentUser.getUid(), key, stats, true);
            }
        }
        displayData();
    }

    private void checkAndUpdateGoalStatus(String goalId, Goal goal) {
        HabitStats stats = habitStats.containsKey(goal.habitKey) ?
                habitStats.get(goal.habitKey) :
                addictionStats.get(goal.habitKey);
        if (stats == null) return;

        boolean isAchieved = false;
        boolean isHabit = habitStats.containsKey(goal.habitKey);

        if ("streak".equals(goal.type)) {
            int currentStreak = isHabit ? calculateHabitStreaks(stats)[0] :
                    (int) TimeUnit.MILLISECONDS.toDays(System.currentTimeMillis() -
                            (stats.relapseDates.isEmpty() ? stats.startDate.getTime() :
                                    Collections.max(stats.relapseDates).getTime()));
            isAchieved = currentStreak >= goal.target;
            Log.d(TAG, "checkAndUpdateGoalStatus: GoalId=" + goalId + ", habitKey=" + goal.habitKey +
                    ", isHabit=" + isHabit + ", currentStreak=" + currentStreak + ", target=" + goal.target);
        } else if ("progress".equals(goal.type)) {
            isAchieved = stats.progress >= goal.target;
            Log.d(TAG, "checkAndUpdateGoalStatus: GoalId=" + goalId + ", habitKey=" + goal.habitKey +
                    ", progress=" + stats.progress + ", target=" + goal.target);
        }

        if (isAchieved && !goal.achieved) {
            goal.achieved = true;
            saveGoalToPreferences(goalId, goal);
            if (currentUser != null) {
                saveGoalToFirebase(currentUser.getUid(), goalId, goal);
            }
            Toast.makeText(this, getString(R.string.goal_achieved_toast, getTextForKey(goal.habitKey)),
Toast.LENGTH_LONG).show();
        }
    }

    private void showAccountabilityPartnerDialog() {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle(R.string.accountability_partner_title);

        View dialogView = LayoutInflater.from(this).inflate(R.layout.dialog_accountability_request, null);
        Spinner habitSpinner = dialogView.findViewById(R.id.habit_spinner);

        List<String> habitOptions = new ArrayList<>();
        Map<String, String> habitKeyMap = new HashMap<>();

        for (String key : habitKeys) {
            habitOptions.add(getTextForKey(key));
            habitKeyMap.put(getTextForKey(key), key);
        }
        for (String key : addictionKeys) {
            habitOptions.add(getTextForKey(key));
            habitKeyMap.put(getTextForKey(key), key);
        }
```

```
        if (habitOptions.isEmpty()) {
            habitOptions.add(getString(R.string.no_habits_available));
        }

        ArrayAdapter<String> habitAdapter = new ArrayAdapter<>(this, android.R.layout.simple_spinner_item,
habitOptions);
        habitAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        habitSpinner.setAdapter(habitAdapter);

        builder.setView(dialogView);
        builder.setPositiveButton(R.string.submit_request, (dialog, which) -> {
            if (currentUser == null) {
                Toast.makeText(this, R.string.login_required, Toast.LENGTH_SHORT).show();
                return;
            }

            String selectedHabit = habitSpinner.getSelectedItem().toString();
            if (selectedHabit.equals(getString(R.string.no_habits_available))) {
                Toast.makeText(this, R.string.no_habits_to_select, Toast.LENGTH_SHORT).show();
                return;
            }

            database.child("users").child(currentUser.getUid()).addListenerForSingleValueEvent(
                    new ValueEventListener() {
                        @Override
                        public void onDataChange(@NonNull DataSnapshot snapshot) {
                            String username = snapshot.child("username").getValue(String.class);
                            Long ageLong = snapshot.child("age").getValue(Long.class);
                            int age = ageLong != null ? ageLong.intValue() : 0;
                            String gender = snapshot.child("gender").getValue(String.class);

                            if (username == null || username.isEmpty()) {
                                Toast.makeText(ResultsActivity.this, R.string.set_username_first, Toast.LENGTH_LONG).show();
                                showUsernameDialog();
                                return;
                            }

                            if (age == 0 || gender == null || gender.isEmpty()) {
                                Toast.makeText(ResultsActivity.this, R.string.profile_incomplete, Toast.LENGTH_LONG).show();
                                return;
                            }

                            // Check if user already has an active request
                            database.child("accountability_requests")
                                    .orderByChild("userId")
                                    .equalTo(currentUser.getUid())
                                    .addListenerForSingleValueEvent(new ValueEventListener() {
                                        @Override
                                        public void onDataChange(@NonNull DataSnapshot snapshot) {
                                            String habitKey = habitKeyMap.get(selectedHabit);
                                            String requestId = database.child("accountability_requests").push().getKey();
                                            if (requestId == null) {
                                                Toast.makeText(ResultsActivity.this, R.string.request_creation_error,
Toast.LENGTH_SHORT).show();
                                                return;
                                            }

                                            Map<String, Object> requestData = new HashMap<>();
                                            requestData.put("userId", currentUser.getUid());
                                            requestData.put("username", username);
```

```
                        requestData.put("habit", selectedHabit);
                        requestData.put("habitKey", habitKey);
                        requestData.put("age", age);
                        requestData.put("gender", gender);
                        requestData.put("timestamp", ServerValue.TIMESTAMP);
                        requestData.put("isActive", true);

                        database.child("accountability_requests").child(requestId).setValue(requestData)
                                .addOnSuccessListener(aVoid -> Toast.makeText(ResultsActivity.this,
R.string.request_submitted_success, Toast.LENGTH_SHORT).show())
                                .addOnFailureListener(e -> Toast.makeText(ResultsActivity.this,
R.string.request_submission_failed, Toast.LENGTH_SHORT).show());
                    }

                    @Override
                    public void onCancelled(@NonNull DatabaseError error) {
                        Toast.makeText(ResultsActivity.this, R.string.firebase_load_error,
Toast.LENGTH_SHORT).show();
                    }
                });
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                Toast.makeText(ResultsActivity.this, R.string.firebase_load_error, Toast.LENGTH_SHORT).show();
            }
        });
    });

    builder.setNegativeButton(R.string.cancel_button, null);
    builder.show();
  }

  private void acceptAccountabilityRequest(String requestId, String partnerUsername) {
    if (currentUser == null) {
      Toast.makeText(this, R.string.login_required, Toast.LENGTH_SHORT).show();
      return;
    }

    String userId = currentUser.getUid();
    database.child("users").child(userId).addListenerForSingleValueEvent(
        new ValueEventListener() {
          @Override
          public void onDataChange(@NonNull DataSnapshot snapshot) {
            String username = snapshot.child("username").getValue(String.class);
            if (username == null || username.isEmpty()) {
              Toast.makeText(ResultsActivity.this, R.string.set_username_first, Toast.LENGTH_SHORT).show();
              showUsernameDialog();
              return;
            }

            database.child("accountability_requests").child(requestId).addListenerForSingleValueEvent(
                new ValueEventListener() {
                  @Override
                  public void onDataChange(@NonNull DataSnapshot requestSnapshot) {
                    if (!requestSnapshot.exists()) {
                        Toast.makeText(ResultsActivity.this, R.string.request_not_found,
Toast.LENGTH_SHORT).show();
                        return;
                    }
```

```
                        Boolean isActive = requestSnapshot.child("isActive").getValue(Boolean.class);
                        if (!Boolean.TRUE.equals(isActive)) {
                            Toast.makeText(ResultsActivity.this, R.string.request_inactive,
Toast.LENGTH_SHORT).show();
                            return;
                        }

                        String requestUserId = requestSnapshot.child("userId").getValue(String.class);
                        String habitKey = requestSnapshot.child("habitKey").getValue(String.class);

                        if (requestUserId == null) {
                            Toast.makeText(ResultsActivity.this, R.string.request_not_found,
Toast.LENGTH_SHORT).show();
                            return;
                        }

                        String chatId = UUID.randomUUID().toString();
                        String chatName = getString(R.string.accountability_chat_name, username,
partnerUsername);

                        Map<String, Object> chatData = new HashMap<>();
                        chatData.put("name", chatName);
                        chatData.put("type", "accountability");
                        chatData.put("created_at", dateFormat.format(new Date()));
                        chatData.put("members", Arrays.asList(userId, requestUserId));
                        chatData.put("habitKey", habitKey);

                        DatabaseReference chatRef = database.child("accountability_chats").child(chatId);
                        chatRef.setValue(chatData).addOnSuccessListener(aVoid -> {
                            // Deactivate the request
                            database.child("accountability_requests").child(requestId).child("isActive").setValue(false);
                            Toast.makeText(ResultsActivity.this, R.string.request_accepted,
Toast.LENGTH_SHORT).show();
                            // Start the chat activity
                            Intent intent = new Intent(ResultsActivity.this, CommunityActivity.class);
                            intent.putExtra("CHAT_ID", chatId);
                            intent.putExtra("CHAT_NAME", chatName);
                            intent.putExtra("IS_ACCOUNTABILITY", true);
                            startActivity(intent);
                        }).addOnFailureListener(e -> {
                            Log.e(TAG, "Error creating chat: " + e.getMessage());
                            Toast.makeText(ResultsActivity.this, R.string.chat_creation_failed,
Toast.LENGTH_SHORT).show();
                            });
                    }

                    @Override
                    public void onCancelled(@NonNull DatabaseError error) {
                        Log.e(TAG, "Error fetching request: " + error.getMessage());
                        Toast.makeText(ResultsActivity.this, R.string.firebase_load_error,
Toast.LENGTH_SHORT).show();
                    }
                });
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Log.e(TAG, "Error fetching user profile: " + error.getMessage());
            Toast.makeText(ResultsActivity.this, R.string.firebase_load_error, Toast.LENGTH_SHORT).show();
```

```
            }
        });
    }

    private void saveStatsToFirebase(String userId, String key, HabitStats stats, boolean isHabit) {
        if (stats == null) return;
        DatabaseReference statsRef = database.child("users").child(userId)
                .child(isHabit ? "habit_stats" : "addiction_stats").child(key);

        Map<String, Object> statsMap = new HashMap<>();
        statsMap.put("start_date", dateFormat.format(stats.startDate));
        statsMap.put("progress", stats.progress);
        statsMap.put("reminder_hour", stats.reminderHour);
        statsMap.put("reminder_minute", stats.reminderMinute);
        if (isHabit) {
            statsMap.put("next_reminder", dateFormat.format(stats.nextReminder));
            statsMap.put("reminder_interval", stats.reminderInterval);
        }
        if (stats.lastCheckDate != null) {
            statsMap.put("last_check_date", dateFormat.format(stats.lastCheckDate));
        }

        List<String> markedDaysStrings = new ArrayList<>();
        for (Date date : stats.markedDays) {
            markedDaysStrings.add(dateOnlyFormat.format(date));
        }
        statsMap.put("marked_days", markedDaysStrings);

        List<String> relapseDatesStrings = new ArrayList<>();
        for (Date date : stats.relapseDates) {
            relapseDatesStrings.add(dateFormat.format(date));
        }
        statsMap.put("relapse_dates", relapseDatesStrings);

        List<String> completionTimestampsStrings = new ArrayList<>();
        for (Date timestamp : stats.completionTimestamps) {
            completionTimestampsStrings.add(dateFormat.format(timestamp));
        }
        statsMap.put("completion_timestamps", completionTimestampsStrings);

        statsRef.setValue(statsMap);
        Log.d(TAG, "saveStatsToFirebase: Key=" + key + ", isHabit=" + isHabit + ", completionTimestamps=" +
completionTimestampsStrings.size());
    }

    private void syncPendingChanges() {
        if (currentUser == null || !HabitUtils.isNetworkAvailable(this)) {
            Log.w(TAG, "Cannot sync: no user or no network");
            return;
        }

        Set<String> pendingChanges = new HashSet<>(pendingChangesPrefs.getStringSet(PENDING_CHANGES_KEY,
new HashSet<>()));
        if (pendingChanges.isEmpty()) {
            Log.d(TAG, "No pending changes to sync");
            return;
        }

        for (String change : pendingChanges) {
            String[] parts = change.split(":");
```

```
            String changeType = parts[0];
            String key = parts.length > 1 ? parts[1] : null;

            try {
                if ("keys_and_labels".equals(changeType)) {
                    saveUserDataToFirebase(currentUser.getUid());
                    for (Map.Entry<String, Goal> entry : goals.entrySet()) {
                        saveGoalToFirebase(currentUser.getUid(), entry.getKey(), entry.getValue());
                    }
                } else if (changeType.startsWith("stats_")) {
                    boolean isHabit = habitKeys.contains(key);
                    HabitStats stats = isHabit ? habitStats.get(key) : addictionStats.get(key);
                    if (stats != null) {
                        saveStatsToFirebase(currentUser.getUid(), key, stats, isHabit);
                    }
                } else if ("settings".equals(changeType)) {
                    saveSettingsToFirebase();
                }
            } catch (Exception e) {
                Log.e(TAG, "Error syncing change: " + change + ", " + e.getMessage());
            }
        }

        // Clear only successfully synced changes
        pendingChangesPrefs.edit().remove(PENDING_CHANGES_KEY).apply();
        Toast.makeText(this, R.string.data_synced, Toast.LENGTH_SHORT).show();
        Log.d(TAG, "Pending changes synced successfully");
    }

    private void createHabitCards(Set<String> keys, Map<String, HabitStats> statsMap, LinearLayout container,
boolean isHabit) {
        container.removeAllViews();
        if (keys.isEmpty()) {
            TextView emptyText = new TextView(this);
            emptyText.setText(isHabit ? R.string.no_habits_selected : R.string.no_addictions_selected);
            emptyText.setPadding(32, 32, 32, 32);
            emptyText.setTextSize(16);
            container.addView(emptyText);
            return;
        }

        for (String key : keys) {
            HabitStats stats = statsMap.computeIfAbsent(key, k -> {
                HabitStats newStats = new HabitStats();
                newStats.startDate = new Date();
                newStats.lastCheckDate = new Date();
                newStats.nextReminder = HabitUtils.getTomorrow();
                newStats.reminderHour = isHabit ? 18 : addictionReminderHour;
                newStats.reminderMinute = isHabit ? 0 : addictionReminderMinute;
                return newStats;
            });

            CardView card = new CardView(this);
            card.setCardElevation(8);
            card.setRadius(16);
            card.setUseCompatPadding(true);
            LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(
                    ViewGroup.LayoutParams.MATCH_PARENT, ViewGroup.LayoutParams.WRAP_CONTENT);
            params.setMargins(16, 8, 16, 8);
            card.setLayoutParams(params);
```

```
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
        layout.setPadding(16, 16, 16, 16);
        card.addView(layout);

        TextView title = new TextView(this);
        title.setText(getTextForKey(key));
        title.setTextSize(18);
        int nightModeFlags = getResources().getConfiguration().uiMode & Configuration.UI_MODE_NIGHT_MASK;
        boolean isDarkTheme = nightModeFlags == Configuration.UI_MODE_NIGHT_YES;
        title.setTextColor(ContextCompat.getColor(this, isDarkTheme ? R.color.teal_200 : R.color.purple_500));
        layout.addView(title);

        if (isHabit) {
            int missedDays = checkMissedDay(stats);
            if (missedDays > 0) {
                TextView missedDayText = new TextView(this);
                missedDayText.setText(getString(R.string.missed_days_warning, missedDays));
                missedDayText.setTextColor(Color.RED);
                missedDayText.setPadding(0, 4, 0, 4);
                layout.addView(missedDayText);
            }
        }

        TextView startDate = new TextView(this);
        startDate.setText(getString(R.string.start_date_label) + " " + new SimpleDateFormat("dd.MM.yyyy HH:mm",
Locale.getDefault()).format(stats.startDate));
        startDate.setPadding(0, 8, 0, 0);
        layout.addView(startDate);

        TextView streakText = new TextView(this);
        TextView longestStreakText = new TextView(this);

        if (isHabit) {
            int[] streaks = HabitUtils.calculateHabitStreaks(stats);
            streakText.setText(getString(R.string.current_streak_label) + " " + streaks[0] + " " +
getString(R.string.days_label));
            longestStreakText.setText(getString(R.string.longest_streak_label) + " " + streaks[1] + " " +
getString(R.string.days_label));
        } else {
            long currentStreakMillis = stats.relapseDates.isEmpty() ?
                    System.currentTimeMillis() - stats.startDate.getTime() :
                    System.currentTimeMillis() - Collections.max(stats.relapseDates).getTime();
            long longestStreakMillis = stats.calculateLongestStreakMillis();
            streakText.setText(getString(R.string.current_streak_label) + " " +
HabitUtils.formatDuration(currentStreakMillis, this));
            longestStreakText.setText(getString(R.string.longest_streak_label) + " " +
HabitUtils.formatDuration(longestStreakMillis, this));
        }
        streakText.setPadding(0, 4, 0, 0);
        longestStreakText.setPadding(0, 4, 0, 0);
        layout.addView(streakText);
        layout.addView(longestStreakText);

        ProgressBar progressBar = new ProgressBar(this, null, android.R.attr.progressBarStyleHorizontal);
        progressBar.setLayoutParams(new LinearLayout.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,
ViewGroup.LayoutParams.WRAP_CONTENT));
        progressBar.setMax(100);
        progressBar.setProgress(stats.progress);
```

```
        progressBar.setPadding(0, 8, 0, 0);
        layout.addView(progressBar);

        TextView progressText = new TextView(this);
        progressText.setText(getString(R.string.progress_label) + " " + stats.progress + "%");
        progressText.setPadding(0, 4, 0, 8);
        layout.addView(progressText);

        if (isHabit) {
            Button markCompletionButton = new Button(this);
            markCompletionButton.setText(R.string.mark_completion_button);
            markCompletionButton.setLayoutParams(new
LinearLayout.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,
ViewGroup.LayoutParams.WRAP_CONTENT));
            markCompletionButton.setOnClickListener(v -> markHabitCompletion(key, stats));
            layout.addView(markCompletionButton);

            card.setOnClickListener(v -> {
                Intent intent = new Intent(ResultsActivity.this, HabitStatsActivity.class);
                intent.putExtra("HABIT_KEY", key);
                intent.putExtra("IS_HABIT", true);
                startActivity(intent);
            });
        }

        updateStats(stats, isHabit, key);
        setupStreakUpdate(key, streakText, longestStreakText, progressBar, progressText, stats, isHabit);
        saveStatsToPreferences(key, stats);
        if (currentUser != null) {
            saveStatsToFirebase(currentUser.getUid(), key, stats, isHabit);
        }

        if (!isHabit) {
            LinearLayout button2Container = new LinearLayout(this);
            button2Container.setOrientation(LinearLayout.HORIZONTAL);
            button2Container.setGravity(Gravity.CENTER_VERTICAL);
            button2Container.setPadding(0, 8, 0, 0);
            layout.addView(button2Container);

            Button markButton = new Button(this);
            markButton.setText(getString(R.string.mark_victory_button));
            markButton.setLayoutParams(new LinearLayout.LayoutParams(0,
ViewGroup.LayoutParams.WRAP_CONTENT, 1.0f));
            markButton.setOnClickListener(v -> markToday(key, stats));
            button2Container.addView(markButton);

            Button panicButton = new Button(this);
            panicButton.setText(getString(R.string.panic_button));
            panicButton.setLayoutParams(new LinearLayout.LayoutParams(0,
ViewGroup.LayoutParams.WRAP_CONTENT, 1.0f));
            panicButton.setOnClickListener(v -> showPanicAnimation());
            button2Container.addView(panicButton);

            card.setOnClickListener(v -> {
                Intent intent = new Intent(ResultsActivity.this, AddictionStatsActivity.class);
                intent.putExtra("ADDICTION_KEY", key);
                intent.putExtra("IS_HABIT", false);
                startActivity(intent);
            });
        }
```

```
        LinearLayout buttonContainer = new LinearLayout(this);
        buttonContainer.setOrientation(LinearLayout.HORIZONTAL);
        buttonContainer.setGravity(Gravity.CENTER_VERTICAL);
        buttonContainer.setPadding(0, 8, 0, 0);
        layout.addView(buttonContainer);

        Button actionButton = new Button(this);
        actionButton.setText(isHabit ? getString(R.string.reminder_button) : getString(R.string.relapse_button));
        actionButton.setLayoutParams(new LinearLayout.LayoutParams(0,
ViewGroup.LayoutParams.WRAP_CONTENT, 1.0f));
        actionButton.setOnClickListener(v -> {
          if (isHabit) {
            showReminderDialog(key, stats);
          } else {
            showRelapseDialog(key, stats);
          }
        });
        buttonContainer.addView(actionButton);

        Button deleteButton = new Button(this);
        deleteButton.setText(getString(R.string.delete_button));
        deleteButton.setLayoutParams(new LinearLayout.LayoutParams(0,
ViewGroup.LayoutParams.WRAP_CONTENT, 1.0f));
        deleteButton.setOnClickListener(v -> showDeleteConfirmationDialog(key, isHabit));
        buttonContainer.addView(deleteButton);

        container.addView(card);
      }
    }
}
```

Уривок з файлу MainActivity.java

Реалізація функціональної задачі авторизації та налаштування профілю користувача в файлі MainActivity.java забезпечується методами, що керують процесом входу, перевіркою та збереженням даних. Метод firebaseAuthWithGoogle реалізує авторизацію через Google Sign-In із Firebase, забезпечуючи безпечний доступ. checkUserDataInFirebase перевіряє наявність даних користувача в Firebase, синхронізуючи їх із локальним сховищем для переходу до основного екрану. saveUserDataToFirebase зберігає ім’я, вік та стать користувача, формуючи основу для персоналізованого трекінгу.

```
package com.alexpechkin.habittracker;

import android.content.SharedPreferences;
import android.util.Log;
import android.widget.Toast;
import androidx.annotation.NonNull;
import com.google.android.gms.auth.api.signin.GoogleSignInClient;
import com.google.firebase.auth.AuthCredential;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.auth.GoogleAuthProvider;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
```

```
import com.google.firebase.database.ValueEventListener;
import java.util.HashSet;
import java.util.Objects;
import java.util.Set;

public class MainActivity extends AppCompatActivity {
   private static final String TAG = "MainActivity";
   private FirebaseAuth mAuth;
   private DatabaseReference mDatabase;
   private GoogleSignInClient mGoogleSignInClient;

   private void firebaseAuthWithGoogle(String idToken) {
      if (idToken == null) {
         Log.e(TAG, "firebaseAuthWithGoogle: idToken is null!");
         Toast.makeText(this, getString(R.string.null_token_error), Toast.LENGTH_SHORT).show();
         updateUI(null);
         return;
      }
      Log.d(TAG, "Отримання Firebase credential з Google idToken...");
      AuthCredential credential = GoogleAuthProvider.getCredential(idToken, null);

      mAuth.signInWithCredential(credential)
            .addOnCompleteListener(this, task -> {
               if (task.isSuccessful()) {
                  Log.i(TAG, "Firebase signInWithCredential: успіх");
                  FirebaseUser user = mAuth.getCurrentUser();
                  boolean isNewUser = Objects.requireNonNull(task.getResult().getAdditionalUserInfo()).isNewUser();
                  Log.d(TAG, "Це новий користувач Firebase: " + isNewUser);
                  assert user != null;
                  Toast.makeText(MainActivity.this,
                        getString(R.string.login_success, user.getEmail()),
                        Toast.LENGTH_SHORT).show();

                  if (!isNewUser) {
                     checkUserDataInFirebase(user.getUid());
                  } else {
                     updateUI(user);
                  }
               } else {
                  Log.w(TAG, "Firebase signInWithCredential: помилка", task.getException());
                  Toast.makeText(MainActivity.this,
                        getString(R.string.firebase_auth_error),
                        Toast.LENGTH_SHORT).show();
                  signOutGoogleOnly();
                  updateUI(null);
               }
            });
   }

   private void checkUserDataInFirebase(String userId) {
      Log.d(TAG, "Перевірка існуючих даних користувача в Firebase: " + userId);

      DatabaseReference userRef = mDatabase.child("users").child(userId);
      userRef.addListenerForSingleValueEvent(new ValueEventListener() {
         @Override
         public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            FirebaseUser currentUser = mAuth.getCurrentUser();

            if (dataSnapshot.exists()) {
               Log.d(TAG, "Знайдено дані користувача в Firebase");
```

```
boolean hasUserData = dataSnapshot.child("name").exists() &&
      dataSnapshot.child("age").exists() &&
      dataSnapshot.child("gender").exists();

boolean hasHabits = dataSnapshot.child("habits").exists();
boolean hasAddictions = dataSnapshot.child("addictions").exists();

if (hasUserData && (hasHabits || hasAddictions)) {
   Log.d(TAG, "Користувач має всі необхідні дані");

   String name = dataSnapshot.child("name").getValue(String.class);
   Long ageLong = dataSnapshot.child("age").getValue(Long.class);
   int age = ageLong != null ? ageLong.intValue() : 0;
   String gender = dataSnapshot.child("gender").getValue(String.class);

   SharedPreferences sp = getSharedPreferences("user_data", MODE_PRIVATE);
   SharedPreferences.Editor editor = sp.edit();
   editor.putString("name", name);
   editor.putInt("age", age);
   editor.putString("gender", gender);
   editor.apply();

   SharedPreferences habitsPrefs = getSharedPreferences("HabitTrackerPrefs", MODE_PRIVATE);
   SharedPreferences.Editor habitsEditor = habitsPrefs.edit();

   Set<String> habitKeys = new HashSet<>();
   Set<String> addictionKeys = new HashSet<>();

   if (hasHabits) {
      for (DataSnapshot habitSnapshot : dataSnapshot.child("habits").getChildren()) {
         String habitKey = habitSnapshot.getValue(String.class);
         if (habitKey != null) {
            habitKeys.add(habitKey);
         }
      }
      habitsEditor.putStringSet("selectedHabits", habitKeys);
   }

   if (hasAddictions) {
      for (DataSnapshot addictionSnapshot : dataSnapshot.child("addictions").getChildren()) {
         String addictionKey = addictionSnapshot.getValue(String.class);
         if (addictionKey != null) {
            addictionKeys.add(addictionKey);
         }
      }
      habitsEditor.putStringSet("selectedAddictions", addictionKeys);
   }

   if (dataSnapshot.child("habit_labels").exists()) {
      for (DataSnapshot labelSnapshot : dataSnapshot.child("habit_labels").getChildren()) {
         String key = labelSnapshot.getKey();
         String value = labelSnapshot.getValue(String.class);
         if (key != null && value != null) {
            habitsEditor.putString(key, value);
         }
      }
   }

   if (dataSnapshot.child("addiction_labels").exists()) {
```

```
                    for (DataSnapshot labelSnapshot : dataSnapshot.child("addiction_labels").getChildren()) {
                        String key = labelSnapshot.getKey();
                        String value = labelSnapshot.getValue(String.class);
                        if (key != null && value != null) {
                            habitsEditor.putString(key, value);
                        }
                    }
                }

                habitsEditor.apply();

                startActivity(new Intent(MainActivity.this, ResultsActivity.class));
                finish();
            } else {
                Log.d(TAG, "У користувача відсутні деякі необхідні дані");
                updateUI(currentUser);
            }
        } else {
            Log.d(TAG, "Даних користувача не знайдено в Firebase");
            updateUI(currentUser);
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {
        Log.e(TAG, "Помилка при отриманні даних користувача: " + databaseError.getMessage());
        Toast.makeText(MainActivity.this,
                getString(R.string.user_data_check_error) + databaseError.getMessage(),
                Toast.LENGTH_SHORT).show();

        FirebaseUser currentUser = mAuth.getCurrentUser();
        updateUI(currentUser);
    }
  });
}

private void saveUserDataToFirebase(String userId, String userName, int age, String gender) {
    DatabaseReference userRef = mDatabase.child("users").child(userId);

    userRef.child("name").setValue(userName);
    userRef.child("age").setValue(age);
    userRef.child("gender").setValue(gender).addOnCompleteListener(task -> {
        if (task.isSuccessful()) {
            Log.d(TAG, "Дані користувача збережено успішно: " + userName + ", age: " + age + ", gender: " +
gender);
            Toast.makeText(MainActivity.this,                    getString(R.string.user_data_saved_success),
Toast.LENGTH_SHORT).show();

            SharedPreferences sp = getSharedPreferences("user_data", MODE_PRIVATE);
            SharedPreferences.Editor editor = sp.edit();
            editor.putString("name", userName);
            editor.putInt("age", age);
            editor.putString("gender", gender);
            editor.apply();

            Intent intent = new Intent(MainActivity.this, HabitsActivity.class);
            startActivity(intent);
        } else {
            Log.e(TAG, "Помилка збереження даних: ", task.getException());
            Toast.makeText(MainActivity.this, "Error saving data: " +
```

```
                Objects.requireNonNull(task.getException()).getMessage(), Toast.LENGTH_SHORT).show();
        }
      });
  }
}
```

Уривок з файлу CommunityActivity.java

Реалізація функціональної задачі підтримки спільноти в додатку HabitTracker забезпечується методами, що керують чатами та повідомленнями. Метод sendMessage відправляє повідомлення до Firebase, забезпечуючи комунікацію між користувачами. sendNotificationToPartner надсилає push-повідомлення партнерам по відповідальності, підтримуючи активну взаємодію. loadChatMessages завантажує історію чату, оновлюючи інтерфейс у реальному часі.

```
package com.alexpechkin.habittracker;

import android.util.Log;
import android.widget.Toast;
import androidx.annotation.NonNull;
import androidx.cardview.widget.CardView;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.ValueEventListener;
import java.util.HashMap;
import java.util.Map;

public class CommunityActivity extends AppCompatActivity {
    private static final String TAG = "CommunityActivity";
    private DatabaseReference database;
    private FirebaseUser currentUser;
    private LinearLayout communityContainer;
    private String currentChatId;
    private boolean isAccountability;

    private void sendMessage(String message) {
        if (currentUser == null || currentChatId == null) return;
```

```
    String userId = currentUser.getUid();
    DatabaseReference messagesRef = database
            .child(isAccountability ? "accountability_chats" : "chats")
            .child(currentChatId)
            .child("messages")
            .push();

    Map<String, Object> messageData = new HashMap<>();
    messageData.put("userId", userId);
    messageData.put("message", message);
    messageData.put("timestamp", System.currentTimeMillis());

    messagesRef.setValue(messageData).addOnCompleteListener(task -> {
        if (task.isSuccessful()) {
            Toast.makeText(this, R.string.message_sent_toast, Toast.LENGTH_SHORT).show();
            scrollToBottom();
            if (isAccountability) {
                sendNotificationToPartner(message);
            }
        } else {
            Toast.makeText(this, R.string.message_send_error, Toast.LENGTH_SHORT).show();
        }
    });
}

private void sendNotificationToPartner(String message) {
    DatabaseReference chatRef = database
            .child("accountability_chats")
            .child(currentChatId);
    chatRef.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            String partnerId = getOtherParticipant(snapshot);
            if (partnerId != null) {
                database.child("users").child(partnerId).child("fcmToken")
                        .addListenerForSingleValueEvent(new ValueEventListener() {
                            @Override
                            public void onDataChange(@NonNull DataSnapshot snapshot) {
                                String partnerToken = snapshot.getValue(String.class);
                                if (partnerToken != null) {
```

```
                        sendFcmNotification(partnerToken, message);
                    }
                }

                @Override
                public void onCancelled(@NonNull DatabaseError error) {
                    Log.e(TAG, "Error fetching partner FCM token: " + error.getMessage());
                }
            });
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {
        Log.e(TAG, "Error fetching chat data: " + error.getMessage());
    }
  });
}

private void loadChatMessages() {
  if (currentChatId == null) return;

  DatabaseReference messagesRef = database
      .child(isAccountability ? "accountability_chats" : "chats")
      .child(currentChatId)
      .child("messages");
  messagesRef.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
      communityContainer.removeAllViews();
      for (DataSnapshot messageSnapshot : snapshot.getChildren()) {
        String userId = messageSnapshot.child("userId").getValue(String.class);
        String message = messageSnapshot.child("message").getValue(String.class);
        Long timestamp = messageSnapshot.child("timestamp").getValue(Long.class);

        if (userId != null && message != null && timestamp != null) {
          displayMessage(userId, message, timestamp);
        }
      }
      handler.postDelayed(CommunityActivity.this::scrollToBottom, 100);
```

```
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Log.e(TAG, "Error loading messages: " + error.getMessage());
            Toast.makeText(CommunityActivity.this, R.string.firebase_load_error, Toast.LENGTH_SHORT).show();
        }
      });
  }
}
```

Уривок з файлу HabitNotificationReceiver.java

Реалізація функціональної задачі сповіщень у додатку HabitTracker забезпечується методом onReceive, який обробляє події нагадувань. Цей метод відображає сповіщення для звичок або залежностей із відповідними заголовками та повідомленнями, а також переналаштовує наступні нагадування для хвилинних інтервалів. Він інтегрується з ResultsActivity, дозволяючи користувачу швидко перейти до відповідного екрану, забезпечуючи ефективну систему нагадувань.

```
package com.alexpechkin.habittracker;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.util.Log;
import androidx.core.app.NotificationCompat;
import java.util.Calendar;

public class HabitNotificationReceiver extends BroadcastReceiver {
    private static final String TAG = "HabitNotificationReceiver";
    private static final String CHANNEL_ID = "habit_notifications";

    @Override
    public void onReceive(Context context, Intent intent) {
        String key = intent.getStringExtra("key");
        String title = intent.getStringExtra("title");
        String message = intent.getStringExtra("message");
        boolean isHabit = intent.getBooleanExtra("isHabit", true);
        int reminderInterval = intent.getIntExtra("reminderInterval", 1);
```

```
        Log.d(TAG, "Received notification for key: " + key + ", interval: " + reminderInterval);

        Intent activityIntent = new Intent(context, ResultsActivity.class);
        activityIntent.putExtra("OPEN_TAB", isHabit ? "habits" : "addictions");
        activityIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TOP);

        assert key != null;
        PendingIntent pendingIntent = PendingIntent.getActivity(
                context,
                key.hashCode(),
                activityIntent,
                PendingIntent.FLAG_UPDATE_CURRENT | PendingIntent.FLAG_IMMUTABLE
        );

        NotificationManager notificationManager = (NotificationManager)
    context.getSystemService(Context.NOTIFICATION_SERVICE);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            NotificationChannel channel = new NotificationChannel(
                    CHANNEL_ID, "Habit Notifications", NotificationManager.IMPORTANCE_HIGH);
            notificationManager.createNotificationChannel(channel);
        }

        NotificationCompat.Builder builder = new NotificationCompat.Builder(context, CHANNEL_ID)
                .setSmallIcon(R.drawable.ic_notification)
                .setContentTitle(title)
                .setContentText(message)
                .setPriority(NotificationCompat.PRIORITY_HIGH)
                .setCategory(NotificationCompat.CATEGORY_REMINDER)
                .setVibrate(new long[]{0, 250, 250, 250})
                .setAutoCancel(true)
                .setDefaults(NotificationCompat.DEFAULT_ALL)
                .setContentIntent(pendingIntent);

        notificationManager.notify(key.hashCode(), builder.build());
        Log.d(TAG, "Notification sent for key: " + key);

        if (isHabit && reminderInterval < 0) {
            Calendar next = Calendar.getInstance();
            next.add(Calendar.MINUTE, Math.abs(reminderInterval));
```

```
            Log.d(TAG, "Rescheduling next notification for " + key + " at " + next.getTime());

            HabitNotificationScheduler.scheduleHabitNotification(
                    context, key, title, next.getTime(), true, message, reminderInterval);
        }
    }
}
```

Уривок з файлу MyFirebaseMessagingService.java

Реалізація функціональної задачі обробки push-повідомлень у додатку HabitTracker забезпечується методами, що керують FCM-повідомленнями. Метод onMessageReceived обробляє вхідні повідомлення, відображаючи сповіщення для чатів із підтримкою стилю обміну повідомленнями. onNewToken оновлює FCM-токен у Firebase, гарантуючи доставку повідомлень. Ці методи забезпечують реальну комунікацію в чатах, підтримуючи функцію партнерської відповідальності.

```
package com.alexpechkin.habittracker;

import android.content.Context;
import android.content.SharedPreferences;
import android.util.Log;
import androidx.annotation.NonNull;
import androidx.core.app.NotificationCompat;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.messaging.RemoteMessage;
import java.util.Map;

public class MyFirebaseMessagingService extends FirebaseMessagingService {
    private static final String CHANNEL_ID = "habit_tracker_notifications";
    private static final String TAG = "MyFirebaseMsgService";

    @Override
    public void onMessageReceived(@NonNull RemoteMessage remoteMessage) {
        super.onMessageReceived(remoteMessage);

        Map<String, String> data = remoteMessage.getData();
```

```
        if (!data.isEmpty()) {
            String title = data.get("title");
            String body = data.get("body");
            String chatId = data.get("chatId");
            Log.d(TAG, "Notification received: Title=" + title + ", Body=" + body + ", ChatId=" + chatId);

            sendNotification(title, body, chatId);
        }
    }

    @Override
    public void onNewToken(@NonNull String token) {
        super.onNewToken(token);
        Log.d(TAG, "New FCM Token: " + token);

        FirebaseUser currentUser = FirebaseAuth.getInstance().getCurrentUser();
        if (currentUser != null) {
            String userId = currentUser.getUid();
            FirebaseDatabase.getInstance().getReference()
                    .child("users").child(userId).child("fcmToken").setValue(token)
                    .addOnCompleteListener(task -> {
                        if (task.isSuccessful()) {
                            Log.d(TAG, "FCM token saved for user: " + userId);
                        } else {
                            Log.e(TAG, "Failed to save FCM token", task.getException());
                        }
                    });
        } else {
            Log.w(TAG, "No authenticated user, skipping FCM token save");
        }
    }
}
```

Уривок з файлу HabitNotificationScheduler.java

Реалізація функціональної задачі планування сповіщень у додатку HabitTracker забезпечується методами, що керують нагадуваннями. Метод scheduleHabitNotification налаштовує точні сповіщення через AlarmManager, враховуючи час і інтервал нагадувань, забезпечуючи своєчасне інформування користувача. cancelScheduledNotification скасовує заплановані

сповіщення, підтримуючи коректне керування розкладом. Ці методи є основою системи нагадувань, інтегруючись із локальним сховищем для збереження даних.

```
package com.alexpechkin.habittracker;

import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.util.Log;
import java.util.Calendar;
import java.util.Date;

public class HabitNotificationScheduler {
   private static final String TAG = "HabitNotificationScheduler";

   public static void scheduleHabitNotification(Context context, String key, String title, Date triggerTime, boolean
isHabit, String message, int reminderInterval) {
      AlarmManager alarmManager = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);
      if (alarmManager == null) {
         Log.e(TAG, "AlarmManager is null for key: " + key);
         return;
      }

      Intent intent = new Intent(context, HabitNotificationReceiver.class);
      intent.putExtra("key", key);
      intent.putExtra("title", title);
      intent.putExtra("message", message);
      intent.putExtra("isHabit", isHabit);
      intent.putExtra("reminderInterval", reminderInterval);

      int requestCode = key.hashCode();
      PendingIntent pendingIntent = PendingIntent.getBroadcast(
            context, requestCode, intent, PendingIntent.FLAG_UPDATE_CURRENT |
PendingIntent.FLAG_IMMUTABLE);

      Calendar calendar = Calendar.getInstance();
      calendar.setTime(triggerTime);
      long triggerTimeMillis = calendar.getTimeInMillis();
```

```
        if (triggerTimeMillis <= System.currentTimeMillis()) {
            if (reminderInterval < 0) {
                calendar.add(Calendar.MINUTE, Math.abs(reminderInterval));
            } else {
                calendar.add(Calendar.DAY_OF_YEAR, reminderInterval > 0 ? reminderInterval : 1);
            }
            triggerTimeMillis = calendar.getTimeInMillis();
        }

        Log.d(TAG, "Scheduling notification for key: " + key + " at " + new Date(triggerTimeMillis) + " with interval: " +
reminderInterval);

        alarmManager.setExactAndAllowWhileIdle(
                AlarmManager.RTC_WAKEUP,
                triggerTimeMillis,
                pendingIntent
        );

        HabitNotificationService.saveNotificationData(context, key, title, new Date(triggerTimeMillis), isHabit,
message, reminderInterval);
    }

    public static void cancelScheduledNotification(Context context, String key) {
        AlarmManager alarmManager = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);
        if (alarmManager == null) {
            Log.e(TAG, "AlarmManager is null when cancelling for key: " + key);
            return;
        }

        Intent intent = new Intent(context, HabitNotificationReceiver.class);
        int requestCode = key.hashCode();
        PendingIntent pendingIntent = PendingIntent.getBroadcast(
                context, requestCode, intent, PendingIntent.FLAG_UPDATE_CURRENT |
PendingIntent.FLAG_IMMUTABLE);

        alarmManager.cancel(pendingIntent);
        pendingIntent.cancel();
        Log.d(TAG, "Cancelled notification for key: " + key);
```

```
        HabitNotificationService.removeNotificationData(context, key);
    }
}
```

Уривок з файлу HabitNotificationService.java

Реалізація функціональної задачі управління сповіщеннями в додатку HabitTracker забезпечується методами фонової служби. Метод rescheduleAllNotifications відновлює заплановані сповіщення з локального сховища, забезпечуючи їх безперервність після перезапуску. saveNotificationData зберігає деталі сповіщень у SharedPreferences, підтримуючи стійкість даних. removeNotificationData видаляє дані сповіщень при скасуванні, забезпечуючи коректне керування розкладом.

```
package com.alexpechkin.habittracker;

import android.content.Context;
import android.content.SharedPreferences;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
import java.lang.reflect.Type;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class HabitNotificationService extends Service {
    private static final String PREFS_NAME = "HabitPrefs";
    private static final String NOTIFICATIONS_KEY = "scheduled_notifications";

    private void rescheduleAllNotifications() {
        SharedPreferences prefs = getSharedPreferences(PREFS_NAME, MODE_PRIVATE);
        String notificationsJson = prefs.getString(NOTIFICATIONS_KEY, null);
        if (notificationsJson != null) {
            Gson gson = new Gson();
            Type type = new TypeToken<List<NotificationData>>(){}.getType();
            List<NotificationData> notifications = gson.fromJson(notificationsJson, type);

            for (NotificationData data : notifications) {
                HabitNotificationScheduler.scheduleHabitNotification(
                        this,
```

```
                data.key,
                data.title,
                new Date(data.triggerTime),
                data.isHabit,
                data.message,
                data.reminderInterval
            );
        }
    }
}

public static void saveNotificationData(Context context, String key, String title, Date triggerTime, boolean isHabit,
String message, int reminderInterval) {
    SharedPreferences prefs = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = prefs.edit();

    Gson gson = new Gson();
    String existingJson = prefs.getString(NOTIFICATIONS_KEY, null);
    List<NotificationData> notifications = new ArrayList<>();

    if (existingJson != null) {
        Type type = new TypeToken<List<NotificationData>>(){}.getType();
        notifications = gson.fromJson(existingJson, type);
    }

    notifications.removeIf(data -> data.key.equals(key));
    notifications.add(new NotificationData(key, title, triggerTime.getTime(), isHabit, message, reminderInterval));

    editor.putString(NOTIFICATIONS_KEY, gson.toJson(notifications));
    editor.apply();
}

public static void removeNotificationData(Context context, String key) {
    SharedPreferences prefs = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = prefs.edit();

    Gson gson = new Gson();
    String existingJson = prefs.getString(NOTIFICATIONS_KEY, null);
    if (existingJson != null) {
        Type type = new TypeToken<List<NotificationData>>(){}.getType();
```

```
            List<NotificationData> notifications = gson.fromJson(existingJson, type);

            notifications.removeIf(data -> data.key.equals(key));

            editor.putString(NOTIFICATIONS_KEY, gson.toJson(notifications));
            editor.apply();
        }
    }
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ВІДСТЕЖЕННЯ ОСОБИСТИХ ЗВИЧОК ТА ПОДОЛАННЯ ЗАЛЕЖНОСТЕЙ

Програма та методика тестування

КПІ.ІП-1120.045490.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Антон ДИФУЧИН

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Олександр ПЕЧКОВСЬКИЙ

Київ – 2025

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є робота мобільного Android-застосунку Habit Tracker, розробленого для відстеження особистих звичок, подолання залежностей, аналізу прогресу користувача та підтримки взаємодії зі спільнотою через групові чати та чати з партнерами підтримки.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи мобільного застосунку Habit Tracker відповідно до функціональних вимог, включаючи відстеження звичок, реєстрацію рецидивів залежностей, встановлення цілей, аналіз прогресу та взаємодію через чати;
- перевірка збереження та синхронізації даних у локальному сховищі (SharedPreferences) та Firebase Realtime Database в онлайн- і офлайн-режимах;
- перевірка сумісності застосунку з операційною системою Android версії 7.0 (API 24) і вище на різних пристроях (смартфони, планшети);
- знаходження проблем, помилок і недоліків у функціональності, безпеці та продуктивності з метою їх усунення;
- перевірка зручності графічного інтерфейсу, включаючи інтуїтивність навігації, читабельність елементів і відповідність принципам Material Design.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення Habit Tracker використовуються такі методи:

- статичне тестування: перевіряється вихідний код застосунку на відповідність стандартам програмування (Google Java Style Guide) та відсутність синтаксичних помилок за допомогою інструментів, таких як SonarQube Cloud;
- динамічне тестування: застосовується під час виконання застосунку на Android-пристроях. Виконується набір тест-кейсів для перевірки коректності роботи, при цьому збираються дані про помилки, аварійні завершення та продуктивність за допомогою логування;
- функціональне тестування: полягає у перевірці відповідності реальної поведінки застосунку функціональним вимогам, зокрема відстеження звичок, реєстрація рецидивів, створення цілей, відправка повідомлень у чатах та відображення сповіщень;
- системне тестування: перевіряється робота застосунку в цілому, включаючи інтеграцію з Firebase Realtime Database, Firebase Cloud Messaging, локальним сховищем та взаємодію між модулями;
- мануальне тестування: виконується тестувальником без автоматизації, із створенням тест-кейсів для перевірки інтерфейсу користувача, навігації, діалогових вікон та реакції на некоректні дії користувача;
- тестування «чорної скриньки»: об’єктом тестування є функції застосунку, такі як додавання звичок, позначення виконання, відправка повідомлень. Перевіряється коректність вихідних даних при заданих вхідних даних.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Під час проведення тестування будуть використовуватись наступні допоміжні засоби:

- Android Studio для запуску застосунку;
- SonarQube Cloud для статичного аналізу коду;
- Postman для тестування API-запитів до Firebase Realtime Database та Firebase Cloud Messaging;
- ADB (Android Debug Bridge) для управління пристроями, встановлення APK;
- Android-застосунок LogFox для ефективного отримання логів;
- тестові Android-пристрої для мануального тестування, а саме: Samsung Galaxy J5 2016 на Android 7.1.1, Poco X3 Pro на Android 12, Xiaomi Pad 5 на Android 13, Samsung Galaxy A33 5G на Android 14, Google Pixel 8 на Android 15.

Порядок проведення тестування буде наступним:

- динамічне тестування на відповідність функціональним вимогам;
- тестування на мобільних пристроях з різною роздільною здатністю екрана;
- тестування на мобільних пристроях з різною версією операційної системи;
- тестування зручності використання: мануальна оцінка інтуїтивності інтерфейсу;
- тестування офлайн-режиму та синхронізації;
- тестування безпеки: аналіз вразливостей;
- статичне тестування коду.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ВІДСТЕЖЕННЯ ОСОБИСТИХ ЗВИЧОК ТА ПОДОЛАННЯ ЗАЛЕЖНОСТЕЙ

Керівництво користувача

КПІ.ІП-1120.045490.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Антон ДИФУЧИН

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Олександр ПЕЧКОВСЬКИЙ

Київ – 2025

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Мобільний застосунок Habit Tracker призначений для відстеження особистих звичок, подолання залежностей, аналізу прогресу користувача та підтримки взаємодії зі спільнотою через чати й партнерську відповідальність. Основні функції включають додавання та позначення виконання звичок/залежностей, реєстрацію рецидивів, встановлення цілей, візуалізацію статистики (календар, гістограми), відправку повідомлень у реальному часі та налаштування сповіщень. Застосунок інтегрується з Firebase Realtime Database для синхронізації даних і Firebase Cloud Messaging для push-сповіщень, підтримує офлайн-режим із локальним сховищем (SharedPreferences) та відповідає принципам Material Design для інтуїтивного інтерфейсу.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– тип процесора: Quad-Core ARM Cortex-A53;

– операційна система: Android 7.0 (API 24);

– об‘єм ОЗП: 2 Гб;

– вільне місце на накопичувачі: 100 МБ;

– підключення до мережі Інтернет зі швидкістю від 5 мегабіт/с (для первинного налаштування та подальшої синхронізації даних);

– екран: роздільна здатність 720x1280 пікселів.

Рекомендована конфігурація технічних засобів:

– тип процесора: Octa-Core ARM Cortex-A73;

– операційна система: Android 11.0 (API 30);

– об‘єм ОЗП: 4 Гб;

– вільне місце на накопичувачі: 500 МБ;

– підключення до мережі Інтернет зі швидкістю від 20 мегабіт/с (для первинного налаштування та подальшої синхронізації даних);

– екран: роздільна здатність 1080x1920 пікселів.

2.2 Завантаження застосунку

На даний момент застосунок можна встановити власноруч, використовуючи відповідний APK-файл, доступний за посиланням https://github.com/AlexPechkin2004/habit-tracker/releases. Для цього необхідно:

– перейти до сторінки релізів за посиланням і завантажити файл app-release.apk на мобільний пристрій під управлінням Android 7.0 або вище;

– для Android 7.1.2 і нижче: увімкнути глобальну опцію в Налаштування → Безпека → Невідомі джерела, підтвердивши попередження про ризики;

– використати файловий менеджер для запуску app-release.apk і виконання встановлення. (Для Android 8.0 і вище: Android вимагає дозволу для конкретного застосунку, з якого виконується встановлення. При спробі відкрити app-release.apk система покаже діалогове вікно з пропозицією увімкнути дозвіл);

– після встановлення авторизуватися через Google Sign-In для доступу до повного функціоналу.

2.3 Перевірка коректної роботи

По завершенню встановлення додатка на робочому столі мобільного пристрою повинна відобразитись іконка даного застосунку. У разі, якщо дана іконка не з’явилась, то встановлення відбулось не успішно. Інакше користувач має змогу запустити додаток, клацнувши на його іконку. Після натискання повинна відобразитись сторінка авторизації через Google акаунт.

3 ВИКОНАННЯ ПРОГРАМИ

Ця інструкція описує повний процес роботи з мобільним застосунком Habit Tracker, від запуску до завершення роботи, включаючи авторизацію, налаштування профілю, відстеження звичок і залежностей, встановлення цілей, взаємодію в чатах, налаштування сповіщень та вихід із акаунту.

1. Запуск застосунку після встановлення:

Якщо ви вже були авторизовані раніше, застосунок автоматично перейде до головного екрану.

Якщо це перший запуск після встановлення, на екрані з’явиться кнопка “Увійти через Google” (Рисунок 3.1). Торкніться її та виберіть обліковий запис Google у спливаючому вікні.

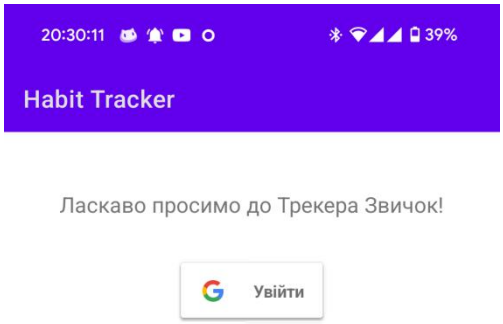


Рисунок 3.1 – Екран авторизації з кнопкою Google Sign-In

Якщо ви входите в цей обліковий запис в застосунку вперше, то після авторизації введіть особисті дані (Рисунок 3.2). Ім’я: введіть у текстове поле. Вік: введіть число від 14 до 90 у текстове поле. Стать: виберіть “Чоловік” або “Жінка” зі списку.

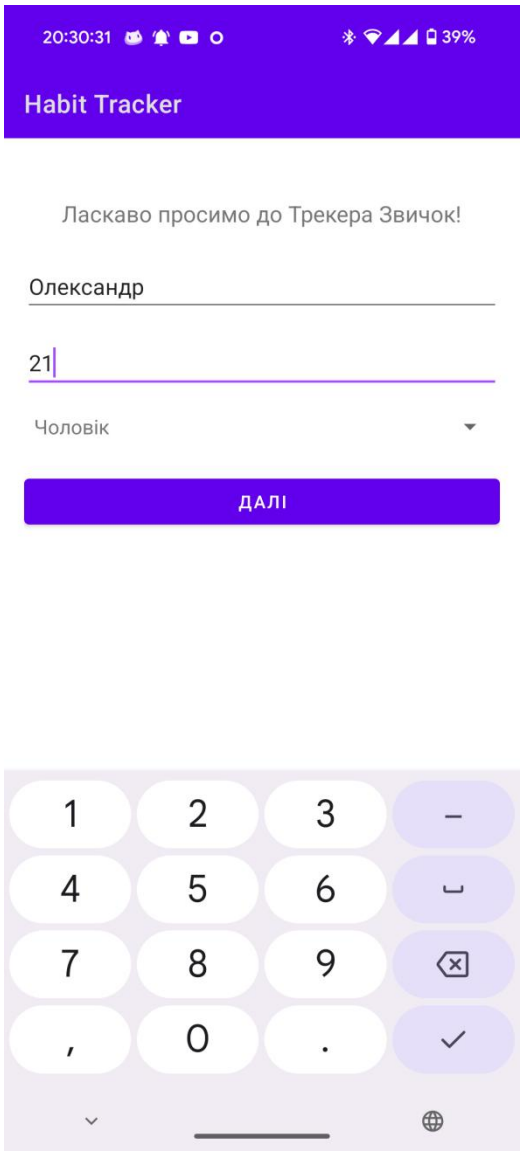


Рисунок 3.2 – Екран налаштування профілю з полями для імені, віку, статі

Торкніться кнопки “Далі”. Відкриється сторінка вибору звичок (Рисунок 3.3). Виберіть звички з передвстановлених та/або додайте власні.

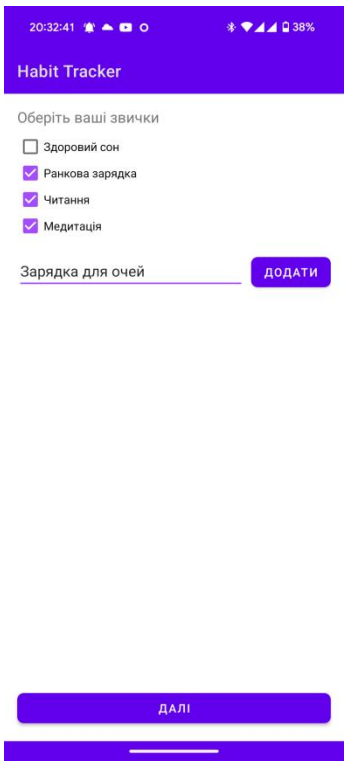


Рисунок 3.3 – Екран початкового вибору звичок

Торкніться кнопки “Далі”. Відкриється сторінка вибору залежностей (Рисунок 3.4). Виберіть залежності з передвстановлених та/або додайте власні.

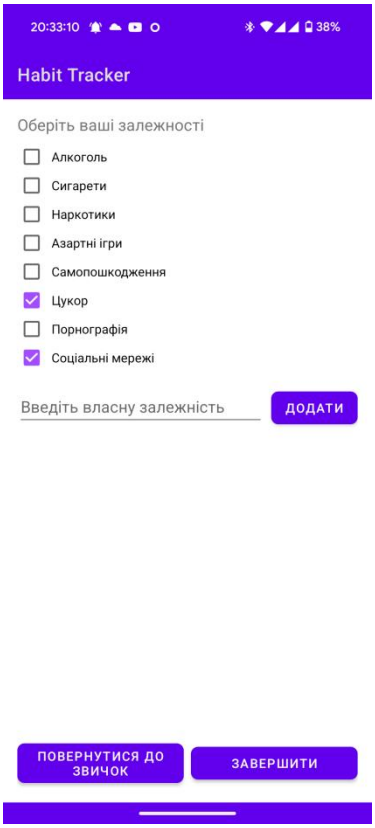


Рисунок 3.4 – Екран початкового вибору залежностей

Торкніться кнопки “Завершити”.

Після авторизації ви потрапите на головний екран, де одразу з’явиться діалогове вікно встановлення імені унікального користувача для Спільноти. Введіть бажане ім’я користувача та натисніть “Зберегти” (Рисунок 3.5).

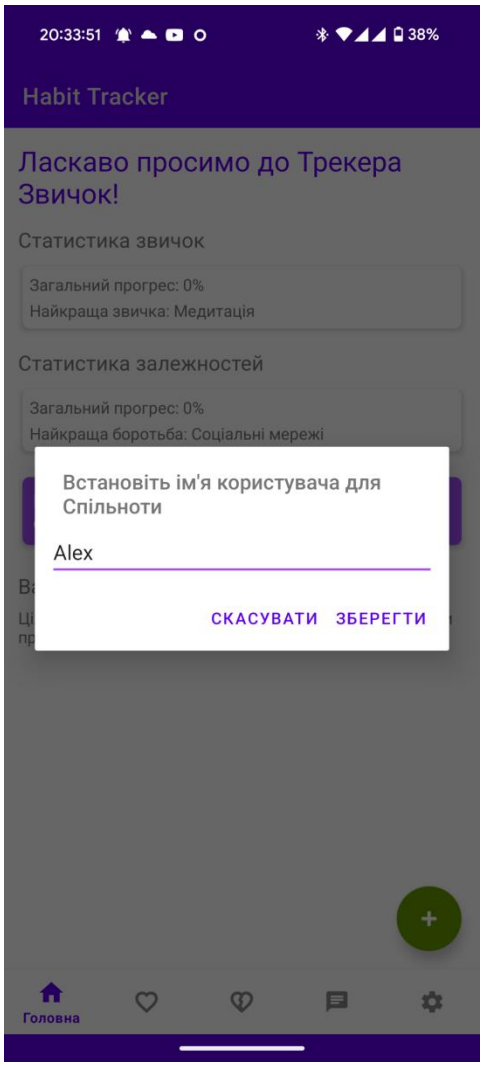


Рисунок 3.5 – Діалогове вікно встановлення імені користувача

2. Основні операції:

Після цього ви побачите головний екран із вкладками (Головна, Звички, Залежності, Спільнота, Налаштування), доступними через нижню панель навігації (Рисунок 3.6).

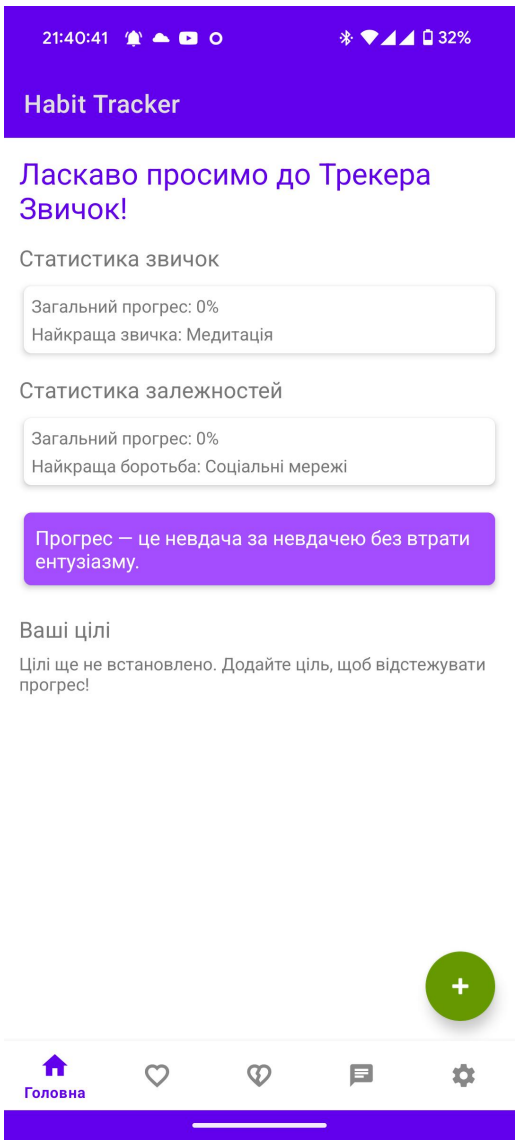


Рисунок 3.6 – Головний екран із вкладками та загальною статистикою

2.1 Додавання нових звичок або залежностей

Перейдіть на вкладку “Звички” (іконка серця) або “Залежності” (іконка розбитого серця) через нижню панель. Торкніться плаваючої кнопки у правому нижньому куті.

У діалоговому вікні виберіть тип (звичка чи залежність) за допомогою радіокнопок (Рисунок 3.7). Також виберіть назву зі спадного списку або введіть власну в текстове поле.

Торкніться “Додати”. Нова звичка/залежність з’явиться у списку як картка з назвою, прогресом і кнопками дій.

Якщо введено некоректні дані (наприклад, порожня назва), з’явиться повідомлення про помилку.

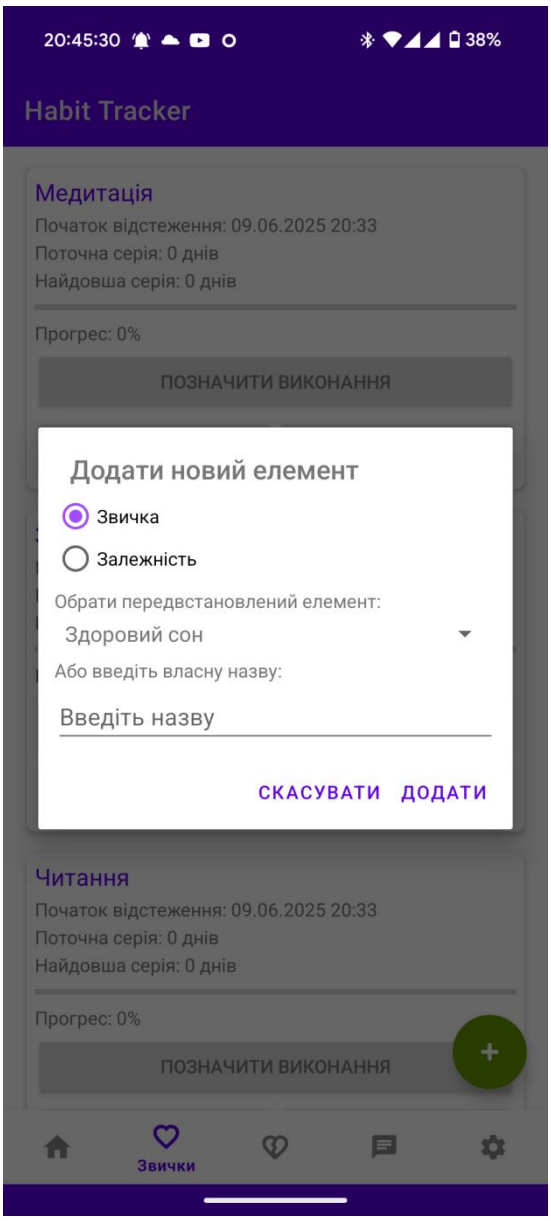


Рисунок 3.7 – Діалогове вікно додавання звички з вибором типу та назви

2.2 Позначення виконання звичок і перемог/рецидивів залежностей

На вкладці “Звички”: знайдіть картку звички та торкніться кнопки “Позначити виконання”. Це потрібно робити щоразу, як дотримались звички. Якщо натискання фіксується перший раз за добу, прогрес і серія оновляться, з’явиться повідомлення “Звичка позначена як виконана!” (Рисунок 3.8).

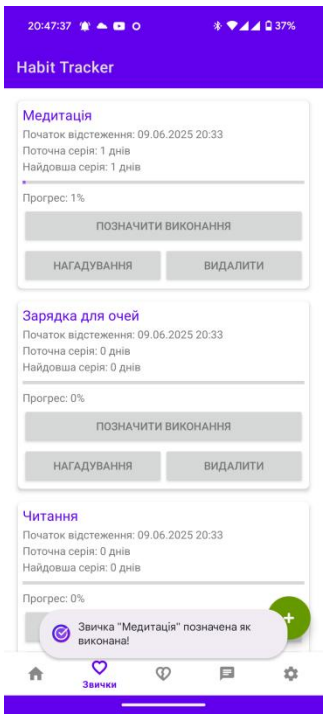


Рисунок 3.8 – Картки звичок

На вкладці “Залежності” для перемоги: торкніться кнопки “Відмітити перемогу” на картці залежності - це потрібно робити раз на день, якщо вдалось прожити день без залежності. Дата фіксується, прогрес оновлюється (Рисунок 3.9).

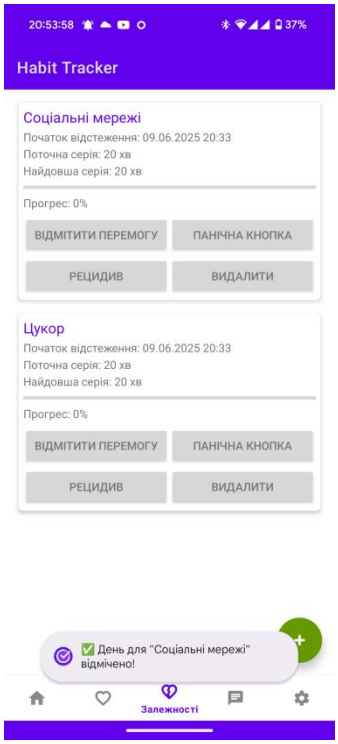


Рисунок 3.9 – Картки залежностей

Для рецидиву: торкніться кнопки “Рецидив”, виберіть дату і час у діалозі, підтвердіть. Це потрібно робити після кожного рецидиву. Кожен рецидив зменшує загальний прогрес залежності на 5% та обнуляє поточну серію.

2.3 Встановлення та відстеження цілей

На вкладці “Головна” торкніться плаваючої кнопки.

У діалоговому вікні виберіть звичку/залежність зі списку (Spinner). Виберіть тип мети (радіокнопки “Серія” або “Прогрес”). Введіть цільове значення (наприклад, 30 днів, 80%) у текстове поле.

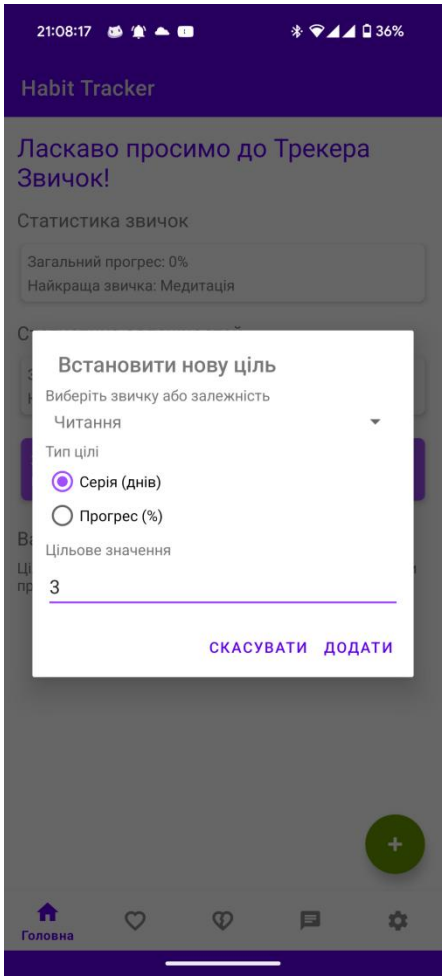


Рисунок 3.10 – Діалогове вікно додавання цілі

Торкніться “Додати”. Ціль з’явиться на головному екрані як картка. Прогрес цілі оновлюється автоматично при досягненні.

Для видалення торкніться кнопки “Видалити” на картці мети та підтвердіть у діалозі.

2.4 Аналіз прогресу

Перейдіть на вкладку “Звички” або “Залежності” та торкніться картки звички/залежності.

На екрані статистики перегляньте календар (з позначками виконань/рецидивів). При натисненні на будь-який минулий день буде показана статистика за цей день (Рисунок 3.11).

Аналізуйте гістограми активності за днями тижня та годинами.

Перегляньте поточну та найдовшу серію, прогрес у відсотках.

Перемикайте місяці за допомогою кнопок “Попередній”/“Наступний” для історичних даних.

Рисунок 3.11 – Екран статистики з календарем і гістограмами

2.5 Взаємодія в чатах

Перейдіть на вкладку “Спільнота”.

Для знаходження партнера підтримки торкніться кнопки “Додати партнера”, виберіть звичку/залежність, підтвердіть (Рисунок 3.12). Або можете прийняти запит іншої людини в списку запитів.

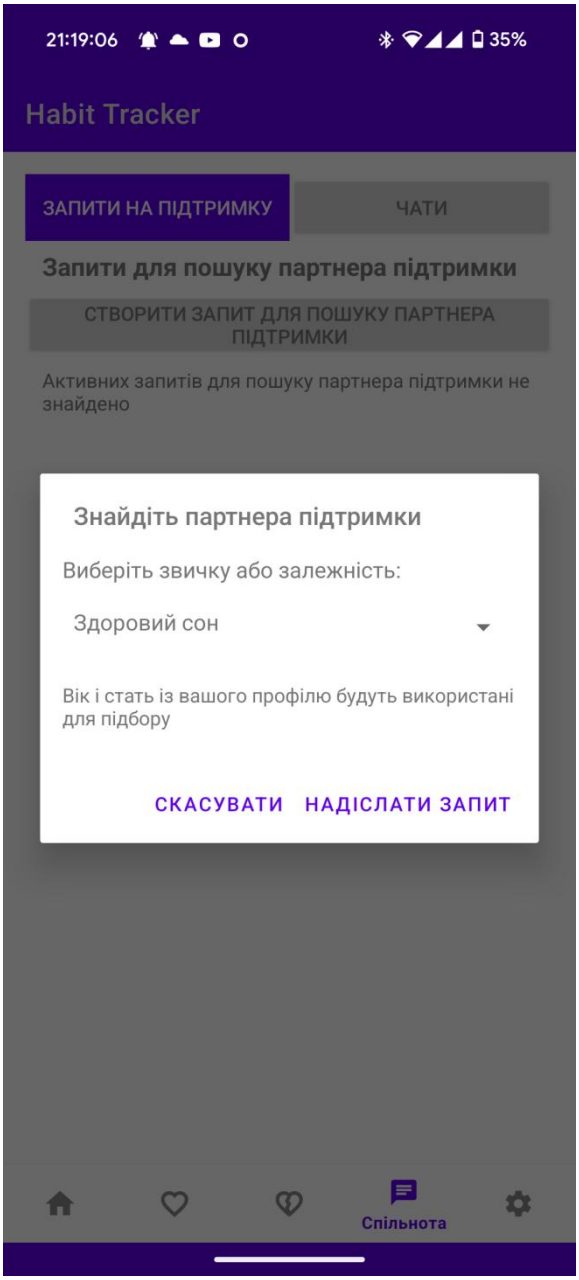


Рисунок 3.12 – Діалогове вікно знаходження партнера підтримки

Після схвалення запиту партнером чат з’явиться у списку у верхній вкладці “Чати” в списку під груповими чатами.

Натисніть “Відкрити чат”. У чаті введіть повідомлення в текстове поле та торкніться кнопки відправки (Рисунок 3.13).

Повідомлення оновлюються в реальному часі, прокручуючись донизу.

Торкніться картки повідомлення, щоб скопіювати або повідомити про порушення.

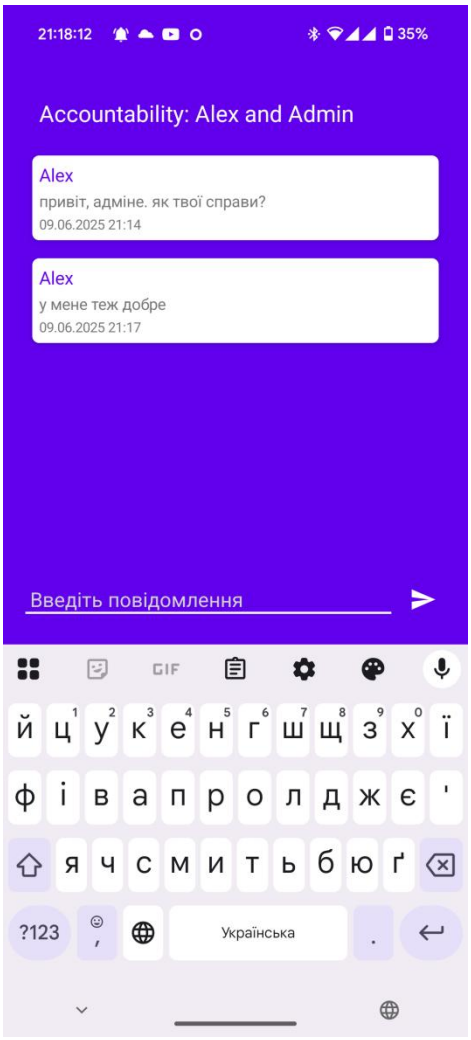


Рисунок 3.13 – Екран чатів із списком повідомлень і полем вводу

Отримуйте push-сповіщення про нові повідомлення від партнерів, торкніться сповіщення для переходу до чату (Рисунок 3.14).

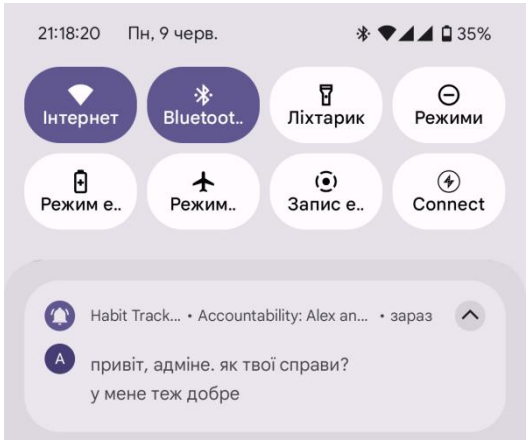


Рисунок 3.14 – Push-сповіщення про нове повідомлення в чаті

2.6 Налаштування профілю, сповіщень і режимів

Перейдіть на вкладку “Налаштування”, відкриється діалогове вікно з налаштуваннями (Рисунок 3.15).

Увімкніть режим сну, виберіть час початку/закінчення.

Увімкніть сповіщення щоденного огляду для залежностей, виберіть час.

Для зворотного зв’язку торкніться “Надіслати відгук” і введіть повідомлення.

Для виходу з облікового запису на вкладці “Налаштування” торкніться “Вийти”. Підтвердіть дію в діалоговому вікні.

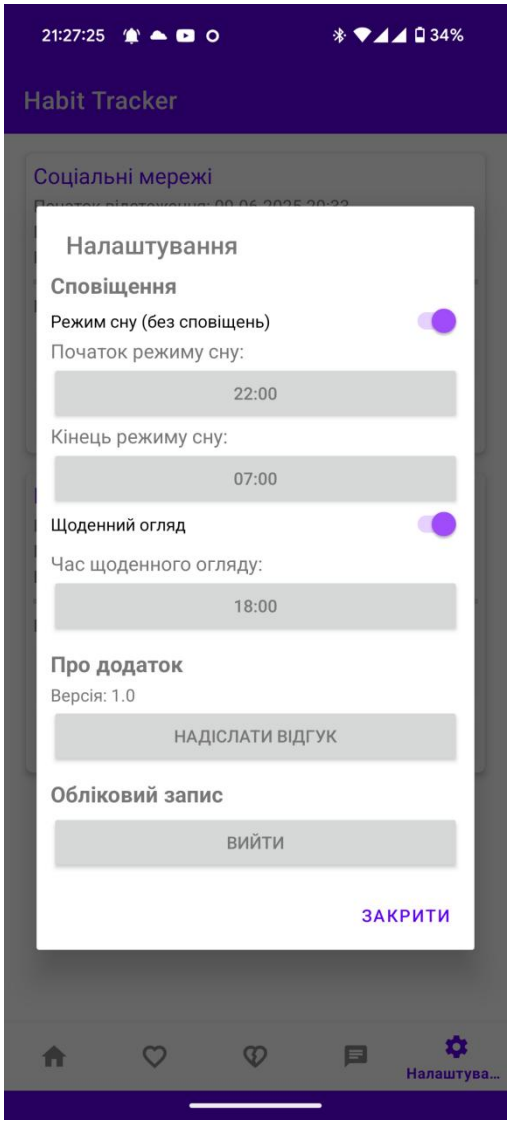


Рисунок 3.15 – Діалогове вікно налаштувань

2.7 Використання панічної кнопки

На вкладці “Залежності” торкніться кнопки “Панічна кнопка” на картці залежності (Рисунок 3.16).

У діалоговому вікні перегляньте мотиваційні повідомлення, що анімовано змінюються кожні 2 секунди.

Торкніться “Закрити”, щоб повернутися до екрану залежностей.

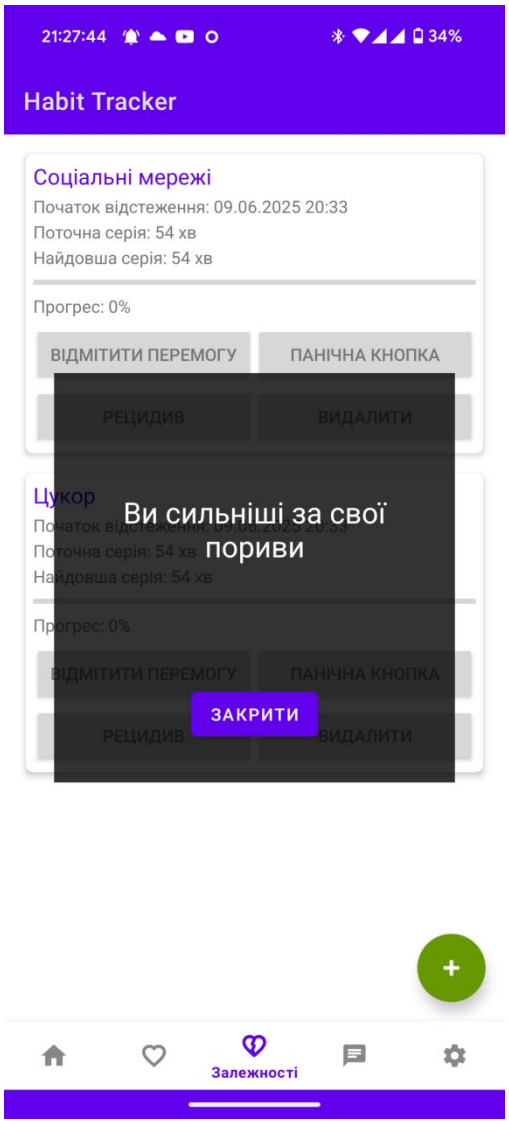


Рисунок 3.16 – Діалогове вікно панічної кнопки з мотиваційним повідомленням

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ВІДСТЕЖЕННЯ ОСОБИСТИХ ЗВИЧОК ТА ПОДОЛАННЯ ЗАЛЕЖНОСТЕЙ

Графічний матеріал

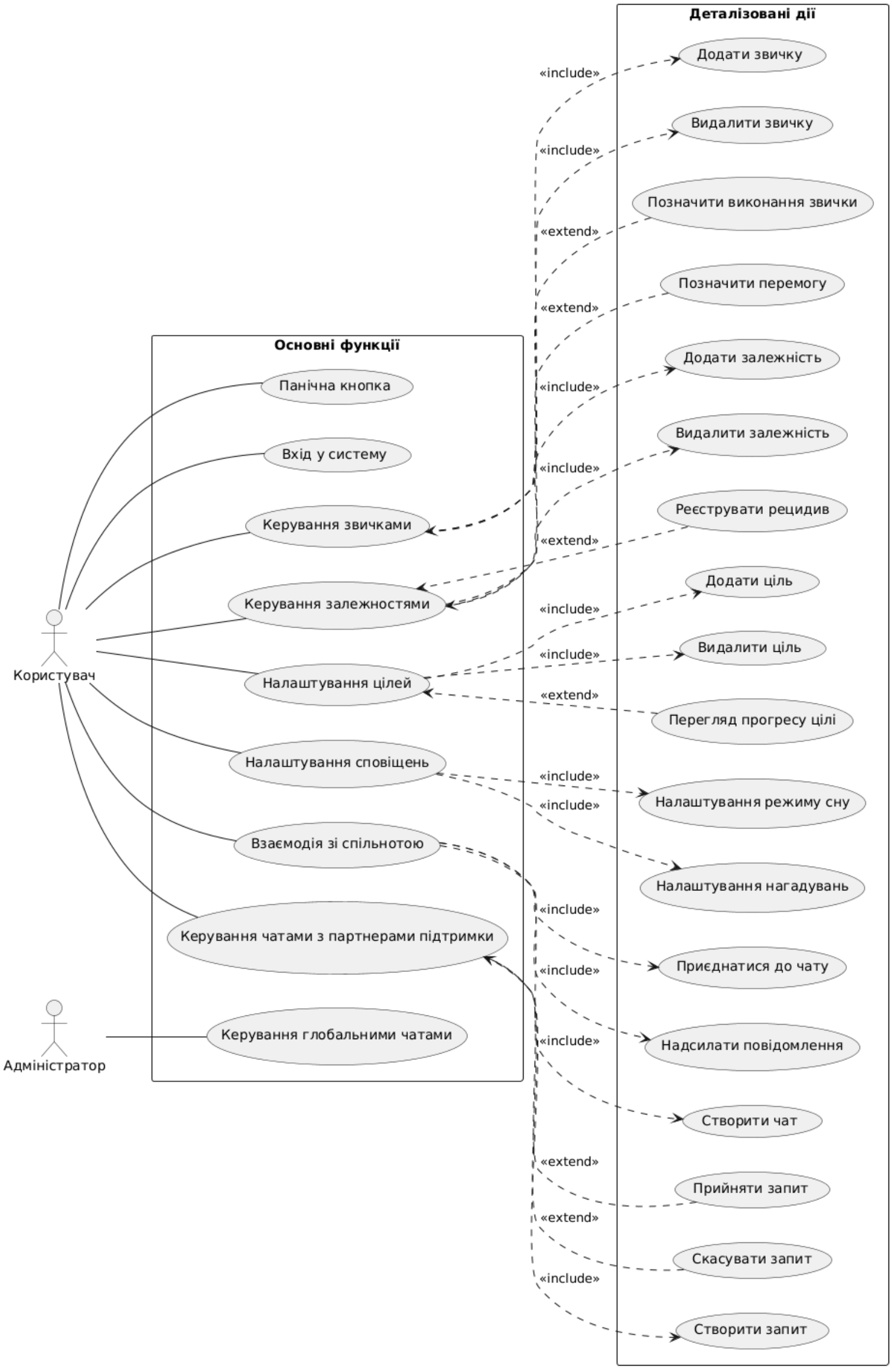
КПІ.ІП-1120.045490.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Антон ДИФУЧИН

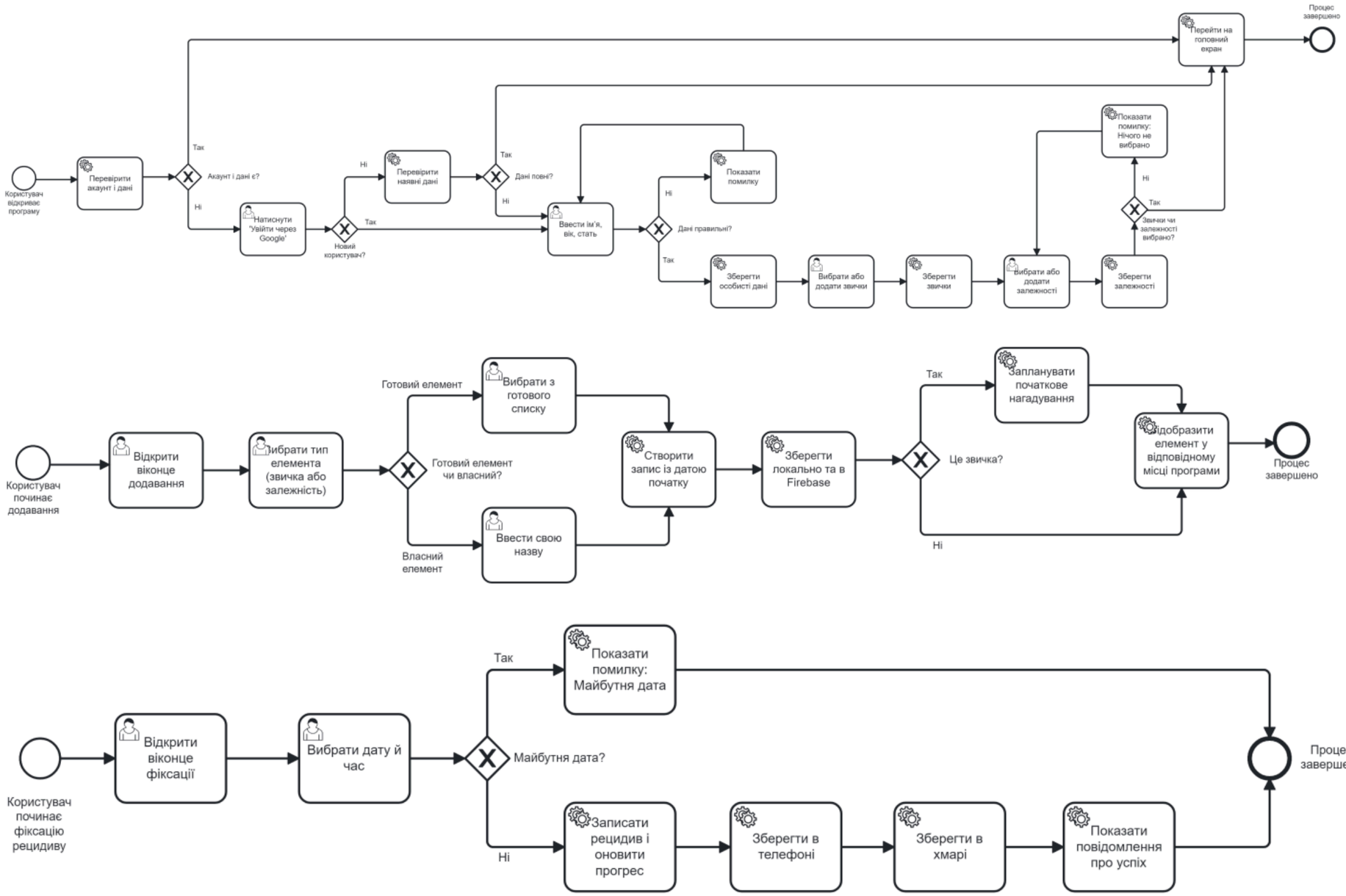
Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Олександр ПЕЧКОВСЬКИЙ

Київ – 2025



					КПІ.ІП-1120.045490.06.99.ССВ	
					Схема структурна варіантів використань	Лит. / Маса / Масштаб
Зм.	Арк.	№ докум.	Підп.	Дата		Аркуш / Аркушів
Розробив		Печковський О.К.			Мобільний застосунок для відстеження особистих звичок та подолання залежностей	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-11
Перевірив		Дифучин А.Ю.				
Т. контр.						
Н. контр.		Ліщук К.І.				
Затвердив		Жаріков Е.В.				



					КПІ.ІП-1120.045490.06.99.ССД	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна діяльності	Літера / Маса / Масштаб
Розробив		Печковський О.К.				
Перевірив		Дифучин А.Ю.				Аркуш / Аркушів
Т. контр.						
					Мобільний застосунок для відстеження особистих звичок та подолання залежностей	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-11
Н. контр.		Ліщук К.І.				
Затвердив		Жаріков Е.В.				

КПІ.ІП-1120.045490.06.99.КЕ

Habit Tracker

Ласкаво просимо до Трекера Звичок!

Статистика звичок
Загальний прогрес: 0%
Найкраща звичка: Медитація

Статистика залежностей
Загальний прогрес: 5%
Найкраща боротьба: Соціальні мережі

Прогрес — це невдача за невдачею без втрати ентузіазму.

Ваші цілі

Ціль для Соціальні мережі: Досягти 6% (Досягнуто)
Встановлено 07.06.2025
ВИДАЛИТИ ЦІЛЬ

Ціль для Читання: Досягти 3 днів (У процесі)
Встановлено 07.06.2025
ВИДАЛИТИ ЦІЛЬ

+

🏠 💔 🤍 💬 ⚙️

Налаштування

Режим сну (без сповіщень)
Початок: 22:00 Кінець: 07:00

Щоденний огляд
Час: 18:00

Про додаток
Версія: 1.0

НАДІСЛАТИ ВІДГУК

Обліковий запис

ВИЙТИ

ЗАКРИТИ

Налаштування нагадувань для звички

Частота нагадувань:
- ○ Щогодини
- ◉ Щодня
- ○ Кожні два дні
- ○ Щотижня
- ○ Щомісяця
- ○ Власний інтервал

Початковий час нагадування:
12 : 30

СКАСУВАТИ ЗБЕРЕГТИ

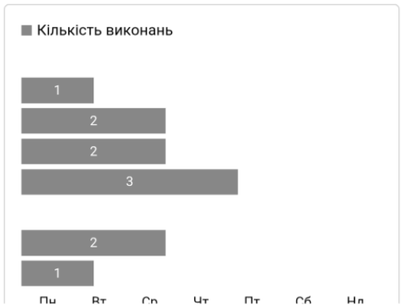
Habit Tracker

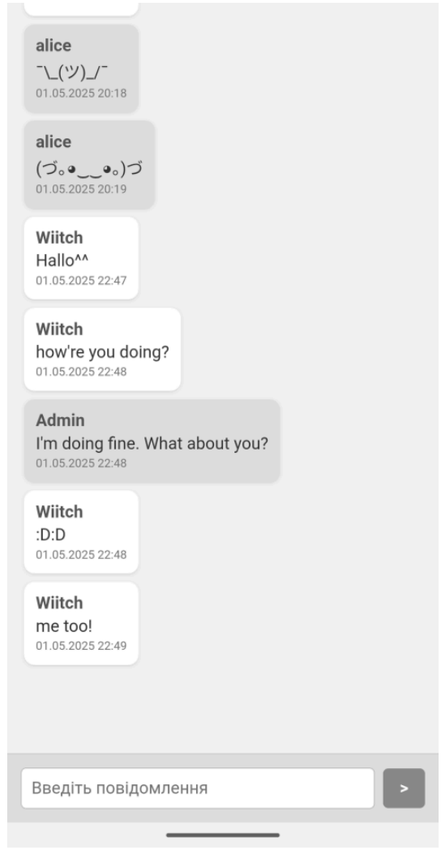
Календар виконань

← травня 2025 →

Пн	Вт	Ср	Чт	Пт	Сб	Нд
			1	2	3	4
5	**6**	**7**	**8**	9	10	11
12	13	14	15	16	17	18
19	20	21	**22**	23	**24**	**25**
26	27	28	**29**	30	31	

Виконання за днями тижня





Встановити нову ціль

Виберіть звичку або залежність
Медитація

Тип цілі
◉ Серія (днів) ○ Прогрес (%)

Цільове значення
Введіть ціль (наприклад, 30 для 30 днів)

СКАСУВАТИ ДОДАТИ

Habit Tracker

Медитація
Початок відстеження: 04.06.2025 12:28
Поточна серія: 1 днів
Найдовша серія: 1 днів
Прогрес: 1%
ПОЗНАЧИТИ ВИКОНАННЯ НАГАДУВАННЯ ВИДАЛИТИ

Читання
Пропущено 4 днів
Початок відстеження: 04.06.2025 12:28
Поточна серія: 0 днів
Найдовша серія: 0 днів
Прогрес: 0%
ПОЗНАЧИТИ ВИКОНАННЯ НАГАДУВАННЯ ВИДАЛИТИ

+

🏠 💔 🤍 💬 ⚙️

Ви сильніші за свої пориви

ЗАКРИТИ

Habit Tracker

Соціальні мережі
Початок відстеження: 04.06.2025 12:28
Поточна серія: 4 днів 3 год 28 хв
Найдовша серія: 4 днів 3 год 28 хв
Прогрес: 6%
ВІДМІТИТИ ПЕРЕМОГУ ПАНІЧНА КНОПКА
РЕЦИДИВ
ВИДАЛИТИ

Цукор
Початок відстеження: 08.06.2025 15:56
Поточна серія: 0 хв
Найдовша серія: 0 хв
Прогрес: 0%
ВІДМІТИТИ ПЕРЕМОГУ ПАНІЧНА КНОПКА
РЕЦИДИВ
ВИДАЛИТИ

+

🏠 💔 💜 💬 ⚙️

Додати новий елемент

◉ Звичка
○ Залежність

Обрати передвстановлений елемент:
Медитація

Або введіть власну назву:
Введіть назву

СКАСУВАТИ ДОДАТИ

					КПІ.ІП-1120.045490.06.99.КЕ		
						Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм			
Розробив		Печковський О.К.				Аркуш	Аркушів	
Перевірив		Дифучин А.Ю.						
Т. контр.								
Н. контр.		Ліщук К.І.			Мобільний застосунок для відстеження особистих звичок та подолання залежностей	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-11		
Затвердив		Жаріков Е.В.						