Вінницький Національний Технічний Університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка мобільного застосунку для управління корисними звичками»

Виконав: студент IV курсу, групи 1ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Гулій Ігор Юрійович
(прізвище та ініціали)
Керівник: к.т.н., доцент каф. ПЗ Романюк О.В.
(прізвище та ініціали)
Рецензент: к.т.н., доцент каф. ЗІ Войтович О.П.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ О.Н. Романюк
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – «Інформаційні технології»
Спеціальність 121 – «Інженерія програмного забезпечення»
Освітньо-професійна програма – «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор
О. Н. Романюк
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Гулію Ігору Юрійовичу

1. Тема роботи – розробка мобільного застосунку для управління корисними звичками.

Керівник роботи: Романюк Оксана Володимирівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

2. Термін подання студентом роботи: 2 червня 2025 року.

3. Вихідні дані до роботи: методи управління звичками – алгоритм динамічного нагадування, метод кореляції звичок на основі коефіцієнтів Пірсона, Спірмена та статистичних критеріїв (Стьюдента, Фішера); моделі – графова модель звичок користувача з ваговими зв'язками; графічний режим – TrueColor; вихідні дані для аналізу – бінарні вектори виконання звичок, часові мітки, категорії звичок; дані про поведінкові патерни – кореляція звичок.

4. Зміст текстової частини: аналіз предметної галузі дослідження, порівняльний аналіз аналогів, аналіз методів розв'язання задачі, постановка задачі дослідження, розробка алгоритмів роботи програми, розробка методу кореляції корисних звичок, розробка методу моніторингу виконання корисних звичок, варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення, вибір середовища розробки, розробка модуля кореляції корисних звичок, опис процесу тестування програмного забезпечення, тестування програмного забезпечення.

5. Перелік ілюстративного матеріалу: титульний слайд, актуальність дослідження, мета дослідження, метод кореляції корисних звичок, метод моніторингу корисних звичок, алгоритми роботи програми, модуль кореляції

корисних звичок, інтерфейс програмного забезпечення, тестування програмного забезпечення, наукова новизна отриманих результатів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Романюк О.В., к.т.н., доцент кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з\п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз технологій управління корисними звичками та аналіз аналогів	25.03.25 – 03.04.2025	виконано
2	Розробка методу кореляції корисних звичок користувача	03.04.25 – 13.04.2025	виконано
3	Розробка методу моніторингу виконання корисних звичок	14.04.25 – 20.04.25	виконано
4	Розробка алгоритмів роботи програмного забезпечення	20.04.25 – 03.05.2025	виконано
5	Розробка програмного забезпечення	03.05.25 – 23.05.2025	виконано
6	Тестування програмного забезпечення	23.05.2025 – 25.05.25	виконано
7	Оформлення матеріалів до захисту БКР	25.05.25 – 01.06.25	виконано

Студент _____ **І. Ю. Гулій**
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ **О. В. Романюк**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

УДК 004.42

Гулій І. Ю. Розробка мобільного застосунку для управління корисними звичками : бакалаврська кваліфікаційна робота зі спеціальності 121 – Інженерія програмного забезпечення, освітня програма – Інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 55 с.

На укр. мові. Бібліогр. : 22 назв ; рис. : 19; табл. 2.

У бакалаврській кваліфікаційній роботі було розроблено методи та програмне забезпечення для автоматизованого моніторингу виконання корисних звичок із використанням статистичних моделей та засобів інтелектуального аналізу даних. Проведено аналіз сучасних програмних рішень для трекінгу звичок, виявлено їх функціональні обмеження, зокрема відсутність персоналізованої аналітики, автоматизованої адаптації та побудови пов’язаних поведінкових патернів.

Розроблено метод моніторингу, що базується на застосуванні кореляційного аналізу виконання звичок, побудові неорієнтованого графа звичок, використанні коефіцієнтів Пірсона, Спірмена та критеріїв достовірності (Стьюдента, Фішера) для виявлення зв’язків між елементами поведінки користувача. Також розроблено метод кореляції поведінкових залежностей у вигляді графа з ваговими ребрами, що дозволяє здійснювати інтерпретацію отриманих аналітичних результатів та генерувати рекомендації щодо корекції структури звичок.

Проведене тестування підтвердило достовірність теоретичних досліджень методів формування корисних звичок.

Ключові слова: моніторинг звичок, мобільний застосунок, кореляційний аналіз, графова модель, поведінковий патерн, візуалізація, програмне забезпечення.

ABSTRACT

UDC 004.42

Guliy I. Yu. Development of a mobile application for managing useful habits: bachelor's thesis in the specialty 121 – software engineering, educational program – software engineering. Vinnytsia: VNTU, 2025. 55 p.

In Ukrainian. Bibliography: 22 titles; figs.: 19; tables: 2.

The bachelor's thesis developed methods and software for automated monitoring of healthy habits using statistical models and intelligent data analysis tools. An analysis of modern software solutions for habit tracking was conducted, and their functional limitations were identified, in particular the lack of personalized analytics, automated adaptation, and the construction of related behavioral patterns.

A monitoring method was developed based on the application of correlation analysis of habit performance, the construction of an undirected habit graph, the use of Pearson and Spearman coefficients, and reliability criteria (Student's, Fisher's) to identify relationships between elements of user behavior. A method for correlating behavioral dependencies in the form of a graph with weighted edges has also been developed, which allows for the interpretation of the obtained analytical results and the generation of recommendations for correcting the structure of habits.

The testing confirmed the reliability of theoretical studies of methods for forming useful habits.

Keywords: habit monitoring, mobile application, correlation analysis, graph model, behavioral pattern, visualization, software.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження.

Сучасний розвиток цифрових технологій та мобільних обчислювальних систем супроводжується стрімким зростанням інтересу до питань підвищення особистої ефективності, самоменеджменту та формування корисних звичок [1]. В умовах інформаційного перевантаження, підвищених когнітивних навантажень та нестабільності соціального середовища, підтримка психоемоційного стану та побудова сталих моделей поведінки стали пріоритетом не лише для окремих користувачів, але й для систем охорони здоров'я, освітніх платформ та корпоративного управління персоналом. Одним із напрямів вирішення цих завдань є використання мобільних застосунків для моніторингу, контролю та корекції повсякденних звичок [2].

Значна кількість користувачів щоденно взаємодіє з трекерами активності, планувальниками завдань або системами нагадувань. Проте більшість наявних мобільних застосунків, орієнтованих на управління звичками, обмежені в функціоналі, не враховують взаємозв'язки між звичками, не забезпечують адаптації до індивідуального ритму життя та ігнорують принципи поведінкової аналітики. Як наслідок, такі системи швидко втрачають актуальність для користувача, не забезпечуючи довготривалого мотиваційного впливу [3].

Одним з перспективних напрямів є застосування методів математичної статистики та елементів інтелектуального аналізу даних для виявлення кореляцій між звичками, прогнозування відхилень у поведінці, формування персоналізованих рекомендацій. Зокрема, використання коефіцієнтів кореляції Пірсона, Спірмена, а також t-критерію Стьюдента та F-критерію Фішера дозволяє реалізувати глибоку багатовимірну аналітику бінарних моделей звичок. Це відкриває можливості для побудови графових структур взаємопов'язаних дій, виявлення звичок-супутників та формування ефективних поведінкових стратегій [4].

Наразі існує невелика кількість програмних засобів, які дозволяють формувати корисні звички. Наявні аналоги не мають функціоналу кореляції звичок та обробку.

Тому розробка мобільного застосунку для формування корисних звичок є актуальною задачею, яка має високу практичну значимість.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою дослідження є підвищення ефективності процесу формування корисних звичок на основі застосування статистичних методів кореляційного аналізу та адаптивного трекінгу звичок.

Відповідно до поставленої мети в бакалаврській кваліфікаційній роботі потрібно вирішити такі **задачі**:

- аналіз предметної галузі;
- розробити алгоритм загальної логіки роботи мобільного застосунку, що включає етапи створення звичок, трекінгу активності, аналітики та динамічного формування рекомендацій;
- розробити метод кореляції корисних звичок, який базується на побудові бінарних векторів виконання звичок користувача, розрахунку необхідних коефіцієнтів для виявлення стійких зв’язків між звичками;
- розробити метод моніторингу виконання корисних звичок, який передбачає оцінку інтенсивності виконання корисних звичок з використанням вагових коефіцієнтів та динамічну модель нагадувань з урахуванням поведінкових залежностей;
- розробка програмного забезпечення для управління корисними звичками;
- тестування розроблених програмних продуктів.

Об'єкт дослідження – процес моніторингу та формування корисних звичок.

Предмет дослідження – методи та засоби моніторингу та формування корисних звичок.

Методи дослідження. У процесі дослідження застосовувалися: теоретичні положення математичної статистики; методи теорії графів для моделювання зв'язків між звичками користувача; поведінкова аналітика; комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів:

1. Розроблено новий метод кореляції корисних звичок у мобільному застосунку, що базується на побудові бінарних векторів виконання звичок користувача, обчисленні коефіцієнтів кореляції Пірсона і Спірмена, а також перевірці статистичної значущості за t-критерієм Стьюдента, що дозволило виявляти стійкі взаємозв'язки між звичками та формувати на їх основі рекомендації щодо оптимізації поведінкових стратегій.

2. Запропоновано новий метод моніторингу виконання корисних звичок, який передбачає побудову двійкової матриці активності, обчислення інтенсивності виконання звичок, аналіз варіації поведінки у часі та виявлення значущих залежностей між звичками, що дозволило динамічно формувати нагадування з урахуванням виявлених кореляцій.

Практичне значення отриманих результатів полягає в тому, що на основі теоретичних досліджень розроблено алгоритми та програми для автоматизованого аналізу, моніторингу та формування корисних звичок.

Особистий внесок здобувача. Усі наукові результати отримано здобувачем самостійно. У працях, опублікованих у співавторстві, студенту належать: [1] – проаналізовано засоби використання мобільних технологій для управління корисними звичками.

Апробація матеріалів бакалаврської квалафікаційної роботи. Основні положення БКР доповідалися та обговорювалися на 2 Міжнародній науково-практичній конференції «nnovative Approaches in Modern Science and Technology» (м. Лісабон, Португалія, 2025).

Публікації. За темою бакалаврської роботи надрукована 1 праця у матеріалах міжнародної науково-практичній конференції.

Структура та обсяг БКР. Бакалаврська кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків, у які винесено технічне завдання, протокол перевірки на плагіат, лістинг програмного забезпечення й ілюстративний матеріал до захисту роботи.

1 АНАЛІЗ ТЕХНОЛОГІЙ УПРАВЛІННЯ КОРИСНИМИ ЗВИЧКАМИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз предметної галузі дослідження

У сучасному цифровому світі зростає увага до психічного та фізичного добробуту людини, і, відповідно, розробка технологій для підтримки здорового способу життя стає дедалі актуальнішою. Поширення смартфонів, підвищення обчислювальної потужності мобільних пристроїв і розвинена інфраструктура мобільних платформ створили сприятливі умови для розробки програмного забезпечення, що дозволяє користувачам ефективно формувати, відстежувати та аналізувати власні повсякденні звички.

Звичка [5] –це автоматизована поведінкова модель, яка виникає внаслідок повторення певної дії в стабільному контексті. Управління звичками охоплює процеси постановки цілей, моніторингу виконання, отримання зворотного зв’язку, а також мотиваційної підтримки. У контексті цифрових технологій, управління звичками реалізується за допомогою спеціалізованих мобільних застосунків, що дозволяють здійснювати реєстрацію звичок, встановлювати нагадування, переглядати статистику прогресу та застосовувати методи позитивного підкріплення.

До основних функціональних компонентів типових мобільних застосунків для управління звичками входять:

– створення звичок**:** користувач може задавати назву, опис, періодичність, час виконання та нагадування;

– трекінг виконання**:** відмітка щоденного виконання звички або автоматичне визначення на основі даних з сенсорів (наприклад, фітнес-трекери);

– візуалізація прогресу**:** графіки, календарі, смуги прогресу, які відображають стабільність і частоту виконання;

– мотиваційна підтримка**:** гейміфікація, нагороди, соціальні інтеграції, що підтримують інтерес користувача до формування звички;

– нагадування та сповіщення**:** системи регулярних повідомлень для забезпечення своєчасного виконання активностей.

Традиційні підходи до формування звичок зазвичай реалізуються через паперові щоденники, таблиці або самоспостереження, однак вони мають низку обмежень. До них належать: низька мотивація до регулярного використання, складність візуалізації прогресу, відсутність миттєвого зворотного зв’язку, а також труднощі з автоматизацією нагадувань. Такі методи значно поступаються сучасним цифровим інструментам, які здатні забезпечити високий рівень персоналізації, інтеграцію з іншими сервісами та постійний супровід користувача у формуванні корисних моделей поведінки.

Одним із ключових аспектів успішного впровадження технологій управління звичками є точне моделювання поведінкових звичок користувача та створення персоналізованих сценаріїв мотивації. Наприклад, застосування алгоритмів машинного навчання дозволяє здійснювати прогнозування ймовірності зриву звички та автоматично пропонувати зміну підходів до формування нових моделей поведінки. Крім того, зростає роль UX-дизайну, який має забезпечити інтуїтивну взаємодію, мінімізацію когнітивного навантаження та візуальну підтримку процесу саморозвитку.

Мобільні застосунки для управління звичками демонструють значні переваги у порівнянні з традиційними методами самостійного розвитку корисних звичок. Головною перевагою є інтегрованість у повсякденне життя користувача, що забезпечується постійною доступністю мобільного пристрою, автоматизованими нагадуваннями та простим у використанні інтерфейсом. Це дозволяє значно підвищити частоту виконання звичок, точність фіксації даних, а також забезпечити об’єктивну оцінку прогресу користувача.

Крім того, сучасні мобільні системи дозволяють формувати складні комбіновані звички, пов’язані з часом доби, геолокацією або біометричними даними. Такі можливості розширюють межі застосування трекерів звичок, зокрема в таких галузях як охорона здоров’я, спорт, освіта, корпоративний менеджмент та психотерапія. Застосунок може автоматично визначати умови, за

яких звичка буде виконана з більшою ймовірністю, і адаптувати відповідні сповіщення або рекомендації.

Таким чином, аналіз предметної галузі управління звичками підтверджує актуальність розробки сучасних мобільних застосунків з високим рівнем персоналізації, зручним інтерфейсом, адаптивною логікою поведінки та ефективними механізмами мотивації. Створення такого програмного забезпечення є важливою складовою цифрової трансформації в галузі підтримки психічного й фізичного здоров’я користувачів. Розробка мобільного застосунку для управління корисними звичками дозволить оптимізувати процес формування здорових поведінкових патернів, підвищити ефективність саморозвитку та забезпечити надійний інструмент контролю особистих досягнень.

1.2 Порівняльний аналіз аналогів

На сьогодні мобільні застосунки для управління звичками займають ключове місце у сфері особистісного розвитку, ментального здоров’я та продуктивності. Завдяки широкій доступності смартфонів, користувачі все частіше звертаються до мобільних рішень для моніторингу, аналізу та формування нових звичок. Такі застосунки дозволяють автоматизувати процес відстеження поведінки, візуалізувати прогрес, надавати зворотний зв'язок та підвищувати мотивацію. Метою цього підрозділу є порівняльний аналіз найпопулярніших рішень у цій галузі, щоб виявити їх переваги, недоліки та визначити напрямки для вдосконалення при розробці власного програмного продукту. До найпоширеніших аналогів застосунків для управління звичками можна віднести:

- Habitive;
- HabitGenius;
- Thefor;
- FunHabit;

- Everyday.

Одним із широко використовуваних додатків є Habitive (рис. 1.1) [6] –це мобільний застосунок, орієнтований на формування та закріплення корисних звичок шляхом щоденного фіксування виконаних дій. Його головна функціональність базується на принципі «один день — один крок», що мотивує користувача до регулярності. Програма має простий і інтуїтивний інтерфейс, де кожна звичка відображається у вигляді плитки, яку потрібно заповнити після виконання.

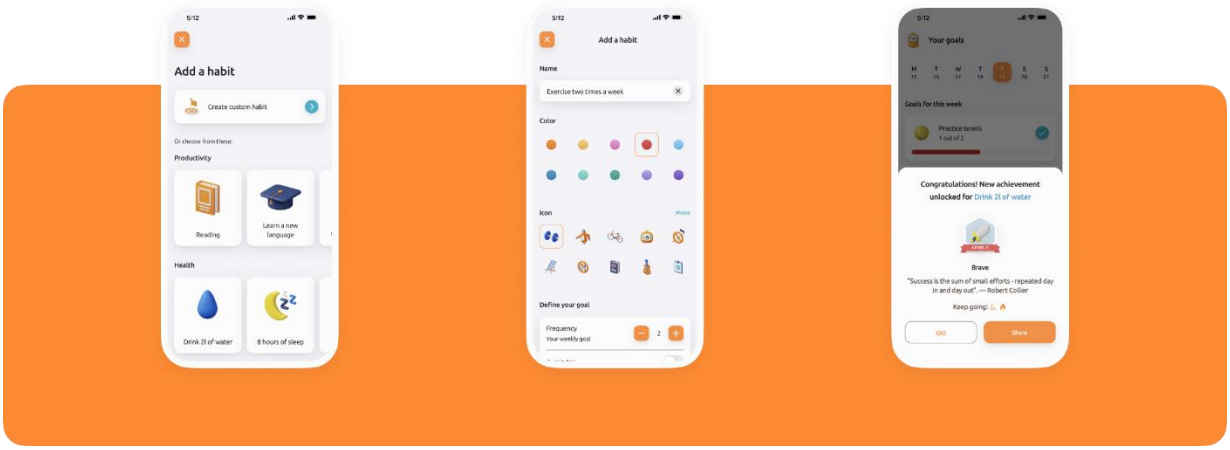


Рисунок 1.1 – Приклад інтерфейсу застосунку Habitive

Користувач може створити будь-яку кількість звичок і встановити частоту повторення (щоденно, щотижнево, довільно). Habitive також включає графічну аналітику прогресу – діаграми та історію успішності, що дозволяє оцінювати стабільність у виконанні завдань. Недоліком є обмежена система нагород і відсутність гейміфікації, що зменшує мотиваційний ефект для користувачів, яким важливо отримувати додаткову стимуляцію в процесі досягнення мети.

Іншим популярним застосунком є HabitGenius [7] (рис. 1.2), який позиціонує себе як інтелектуальний трекер звичок, орієнтований на персоналізацію. Основна особливість цього додатку – використання міні-опитувань, аналізу настрою та мікроаналізу звичок для оптимізації програми користувача.

Алгоритми на основі штучного інтелекту (AI) аналізують поведінку та

адаптують рекомендації до особистих потреб. HabitGenius активно інтегрується з системами моніторингу здоров'я, календарем та нагадуваннями.

Крім стандартного трекінгу, HabitGenius містить аналітичні модулі для визначення тригерів, що впливають на регулярність звичок, а також рекомендації щодо оптимального часу виконання дії. Недоліком є складність інтерфейсу для новачків та обмежений безкоштовний функціонал.

Рисунок 1.2 – Інтерфейс застосунку HabitGenius

Thefor (рис. 1.3) є застосунком з фокусом на ментальне здоров'я та створення мікрозвичок через щоденні короткі ритуали [8]. Застосунок поєднує функціональність трекера звичок і соціальної мережі. Його основна ідея – формування звичок у групах. Користувачі можуть приєднуватися до спільнот за інтересами або створювати власні міні-групи для колективного досягнення мети (наприклад, «30 днів без цукру» або «ранкова зарядка щодня»).

Застосунок має вбудовану систему чатів, голосування, обміну успіхами та змагальні механіки. Це дозволяє створювати ефект колективної відповідальності та підтримки, що є потужним фактором мотивації. Thefor також надає систему рівнів, бейджів та щотижневих звітів, а також API-інтеграцію з іншими сервісами

(фітнес-трекерами, календарем). Візуальна аналітика реалізована у вигляді шкал досягнення та прогрес-барів для кожної звички.

Основний недолік – складна система реєстрації та висока залежність від активності груп, що знижує ефективність при індивідуальному використанні.

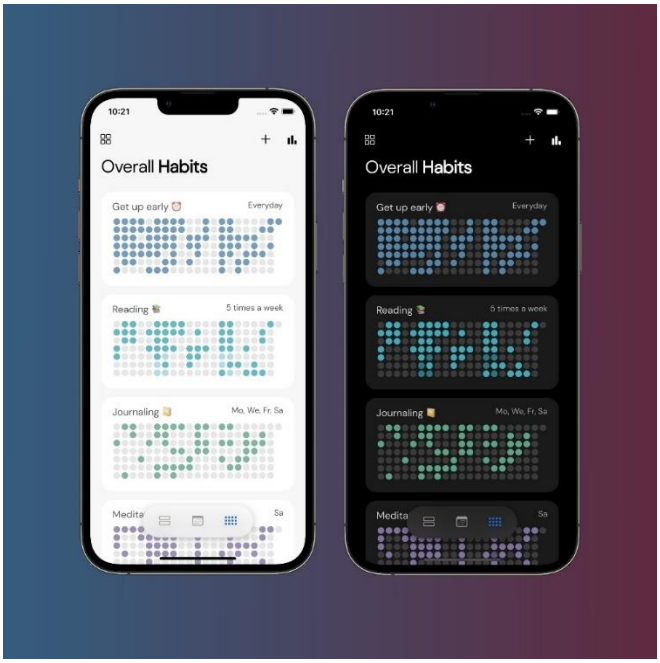


Рисунок 1.3 – Інтерфейс застосунку Thefor

FunHabit (рис. 1.4) [9] – мобільний застосунок з яскраво вираженим ігровим підходом до формування звичок. Він побудований на принципах гейміфікації: кожна дія користувача нагороджується балами досвіду (XP), монетами, рівнями та віртуальними досягненнями. Користувач може обирати аватар, отримувати віртуальні нагороди, відкривати нові локації або функції в залежності від рівня активності. Завдяки інтерактивності, додаток стимулює виконання звичок шляхом постійного зростання внутрішньої цінності досягнень.

FunHabit підходить для широкої цільової аудиторії, зокрема підлітків та молоді. Інтерфейс яскравий, гнучкий, дозволяє створювати необмежену кількість завдань та комбінувати звички у «місії».

Серед мінусів – обмежена аналітика, відсутність глибокого аналізу поведінкових моделей користувача, що робить його менш придатним для професійної чи наукової роботи з поведінковими патернами.

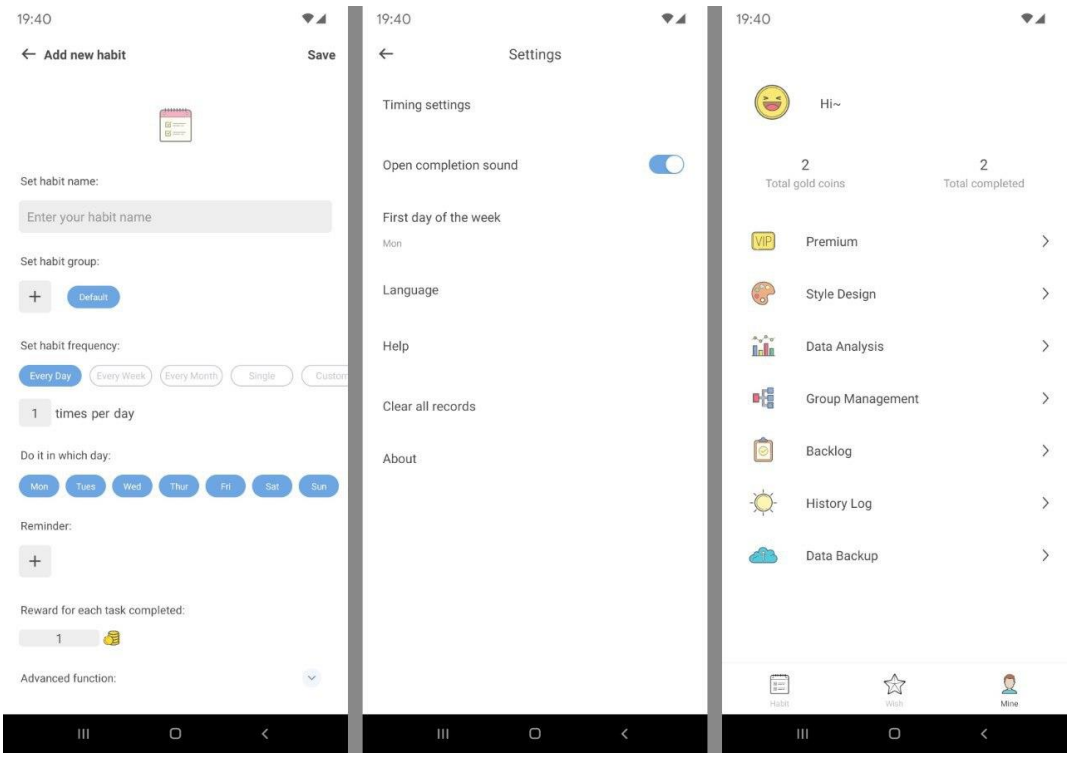


Рисунок 1.4 – Інтерфейс застосунку FunHabit

Останнім додатком, включеним у цей аналіз, є Everyday (рис. 1.5) [10] – популярний трекер звичок із візуальною концепцією “ланцюга звичок”. Його головна особливість — чітка та наочна сітка виконаних дій, що дозволяє користувачам бачити прогрес без зайвих елементів. Однак у додатку відсутній аналітичний модуль, що обмежує глибше розуміння динаміки звичок.

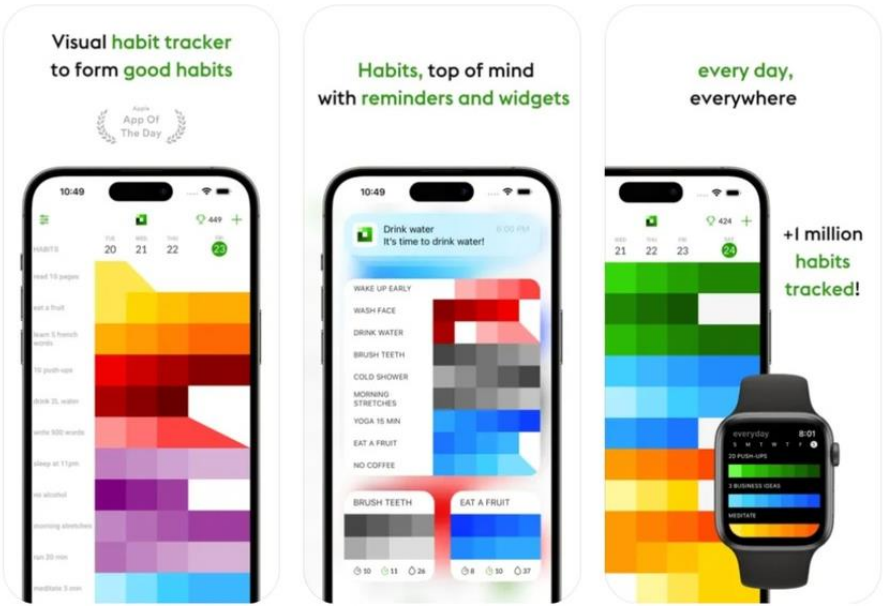


Рисунок 1.5 – Інтерфейс застосунку Everyday

Порівняльний аналіз показує, що кожен із додатків має свої унікальні особливості, проте не забезпечує комплексного підходу до управління звичками з урахуванням індивідуальної поведінки, мотиваційних факторів та адаптивного управління.

Це відкриває можливість для створення нового мобільного застосунку, який поєднує кращі функції аналогів та додає нові механізми персоналізованого аналізу й взаємодії.

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерії	**Habitive**	**HabitGenius**	**The for**	**Fun Habit**	**Everyday**	**Власний застосунок**
Гнучкість налаштування звичок	+	+	–	–	+	+
Адаптивність	–	+	–	–	–	+
Візуалізація прогресу	+	–	–	+	+	+
Елементи гейміфікації	–	–	–	+	–	+
Підтримка інтеграцій	–	–	–	–	–	+
Аналітичний модуль	–	+	–	–	–	+
Загальна оцінка	60%	70%	50%	65%	60%	100%

Аналіз результатів порівняльної таблиці дозволяє зробити висновок, що існуючі рішення переважно покривають лише окремі аспекти процесу управління звичками: або мотивацію, або візуалізацію, або автоматизацію.

Розробка нового мобільного застосунку повинна базуватись на інтеграції всіх основних компонентів: адаптивності, аналітики, гнучких налаштувань, гейміфікації та сучасного UX-дизайну. Це забезпечить конкурентну перевагу і дозволить ефективно впливати на процес формування та підтримання корисних звичок серед користувачів.

Отже, розробка мобільного застосунку для управління корисними звичками відкриває широкі можливості для підвищення ефективності особистісного розвитку, самоменеджменту та формування здорового способу життя. Аналіз існуючих аналогів свідчить про наявність значного попиту на подібні рішення, однак кожен з розглянутих застосунків має певні обмеження — від відсутності гнучкої аналітики до браку персоналізації чи гейміфікації. Власна розробка дозволить поєднати переваги наявних рішень, усунути їх недоліки та реалізувати інноваційний підхід до підтримки звичок із урахуванням сучасних технологічних можливостей, що сприятиме підвищенню мотивації, відповідальності та сталості змін у поведінці користувача.

1.3 Аналіз методів розв’язання задачі

Розробка мобільного застосунку для управління корисними звичками вимагає цілісного підходу до вирішення задач, пов’язаних із збиранням, аналізом, візуалізацією та мотиваційною підтримкою користувача під час формування звичок. Основна мета – забезпечити простий, ефективний і персоналізований інструмент, що сприятиме закріпленню позитивних змін у повсякденній поведінці людини. Для реалізації такого рішення необхідно проаналізувати і визначити найбільш ефективні методи управління звичками, способи візуалізації прогресу, засоби мотивації користувача, а також технічні підходи до реалізації цих функцій.

Одним із ключових компонентів у побудові ефективного застосунку є метод відстеження звичок. Найпоширенішим способом є використання щоденного чек-листа, що дає змогу фіксувати виконання або невиконання

завдань [11]. Такий метод простий у реалізації й дозволяє користувачеві візуально спостерігати за виконанням завдання. Проте він не завжди забезпечує глибоку аналітику, не враховує часові інтервали, мотиваційні коливання та не адаптується до змін у поведінці користувача. Тому даний метод рекомендовано доповнити.

Більш гнучким є метод трекінгу з адаптивною логікою, який передбачає зміну параметрів завдання залежно від поведінки користувача [12]: наприклад, система може рекомендувати змінити ціль, зменшити навантаження або, навпаки, підвищити інтенсивність. Такий підхід значно ефективніший, оскільки враховує індивідуальні особливості користувача та дозволяє утримати його залучення. Однак реалізація адаптивної логіки вимагає алгоритмічної складності та збору персоніфікованих даних, тому доцільним є її реалізація лише в частково автоматизованому вигляді на початковому етапі розробки.

Для мотивації користувача широко використовуються методи гейміфікації [13]: система балів, бейджів, рівнів, а також нагородження за досягнення. Це сприяє підвищенню внутрішньої мотивації, створює елемент змагання або особистої перемоги. Недоліком є можлива втрата зацікавлення з часом, якщо не впроваджується новий контент або не підтримується індивідуальна зацікавленість користувача. Тому мотиваційну систему варто реалізовувати з можливістю персоналізації – наприклад, обирати тип винагород або стилі гейміфікації (досягнення, соціальна активність, нагадування тощо).

Ще одним ефективним підходом є використання аналітики прогресу, яка дозволяє користувачу бачити не лише факт виконання звички, але й її динаміку, статистику, вплив на загальний рівень активності. Тут доцільно використовувати візуальні графіки, діаграми, аналітичні підсумки. Такий метод дає змогу формувати більш усвідомлене ставлення до процесу змін, а також допомагає адаптувати подальші цілі. Важливо при цьому забезпечити простоту інтерпретації даних, уникати складних схем, і натомість орієнтуватися на зручний, інтуїтивний інтерфейс.

У плані технічної реалізації рекомендується використання локального

сховища даних у поєднанні з хмарною синхронізацією, що дозволяє як автономну роботу застосунку, так і можливість резервного копіювання й роботи з кількох пристроїв. Крім того, для реалізації сповіщень можна застосувати алгоритми динамічного нагадування, що враховують активність користувача протягом дня та оптимальний час доби для залучення уваги.

Особливу увагу варто приділити побудові моделі поведінки користувача, яка дозволяє прогнозувати й адаптувати інтерфейс та функціонал до звичок та особистих уподобань людини. Впровадження машинного навчання у майбутньому (на основі накопичених даних) дозволить створити інтелектуального помічника, який буде підказувати оптимальний маршрут формування нової звички.

Розглянувши переваги та недоліки кожного методу, було вирішено використовувати комбінований підхід, який поєднує адаптивний трекінг звичок, гейміфікацію з можливістю персоналізації, аналітику прогресу у вигляді візуалізацій, систему динамічних нагадувань, а також надійне зберігання даних з підтримкою хмарної синхронізації. Додатково заплановано впровадження інтелектуального модуля аналізу поведінки користувача з метою підвищення ефективності формування звичок. Такий підхід дозволить створити функціонально насичений, зручний та гнучкий мобільний застосунок, здатний адаптуватися до індивідуальних потреб користувача й забезпечити стабільне зростання його особистої продуктивності.

1.4 Постановка задачі дослідження

Проаналізувавши переваги та недоліки існуючих мобільних застосунків для управління корисними звичками, було визначено необхідність створення інноваційного застосунку, який поєднує функціональність, персоналізацію, ефективну мотивацію та аналітичні можливості. Для успішної реалізації проєкту необхідно виконати такі завдання:

- розробити загальний алгоритм роботи мобільного застосунку, що описує етапи взаємодії з користувачем: створення звичок, їх налаштування, реєстрацію активностей, обробку даних, побудову зв’язків та надання рекомендацій;
- розробити метод кореляції корисних звичок , що дозволить виявляти взаємозв’язки між виконанням окремих звичок, їхнім впливом одна на одну, та рекомендувати оптимальні комбінації для досягнення цілей користувача;
- розробити метод моніторингу виконання корисних звичок, що дозволить автоматично відслідковувати виконання корисних звичок користувачем з можливістю динамічного нагадування з урахуванням активності користувача, часу доби, контексту та історії виконання завдань;
- розробити модуль аналітики, який відображатиме статистику виконання звичок, прогрес, ефективність та зони, які потребують уваги, за допомогою графіків і діаграм;
- створити адаптивний інтерфейс користувача з інтуїтивною навігацією, що забезпечить швидкий доступ до основних функцій та дозволить легко вносити зміни;
- розробити програмне забезпечення для управління корисними звичками, реалізоване у вигляді мобільного застосунку на основі Flutter, Kotlin та SQLite;
- провести тестування функціональних модулів застосунку та оцінити їх ефективність за критеріями стабільності, зручності використання, точності обліку та мотиваційного впливу на користувача.

Виконання зазначених завдань дозволить створити сучасний мобільний застосунок для ефективного формування та підтримки корисних звичок, що адаптується до індивідуальних потреб користувача.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі було проведено огляд сучасного стану мобільних

застосунків для управління корисними звичками. Було розглянуто такі популярні аналоги, як Habitive, HabitGenius, Thefor, FunHabit та Habitify. Кожен з них має свої сильні сторони, зокрема зручний інтерфейс, гнучкі нагадування, гейміфікацію або аналітику, проте також виявлено низку недоліків: обмежена персоналізація, відсутність аналізу кореляції звичок, перевантаження функціями або, навпаки, їх нестача.

На основі проведеного порівняльного аналізу було сформульовано завдання для подальшої розробки власного мобільного застосунку для управління корисними звичками. Основними напрямами роботи визначено: розробку адаптивного інтерфейсу, гнучкої системи нагадувань, гейміфікації, потужного модуля аналітики та унікального методу кореляції звичок, що дозволить виявляти їх взаємозв’язки для покращення загальної ефективності саморозвитку користувача. Таким чином, результати аналізу підтвердили доцільність створення нового програмного рішення, що поєднує переваги існуючих аналогів із новими підходами, зокрема кореляційним аналізом звичок, що відкриває нові можливості в галузі персоналізованого трекінгу звичок.

2 РОЗРОБКА МЕТОДІВ МОНІТОРИНГУ ТА КОРЕЛЯЦІЇ КОРИСНИХ ЗВИЧОК ТА АЛГОРИТМІВ ПРОГРАМИ

2.1 Розробка алгоритмів роботи програми

Для ефективного функціонування мобільного застосунку з управління корисними звичками необхідно створити узагальнений алгоритм, що описує основну логіку його роботи (рис. 2.1). Розробка такого алгоритму дозволяє систематизувати функціональні модулі, забезпечити стабільну взаємодію з користувачем та гарантувати точність збору й обробки даних.

На першому етапі програма завантажує початкове середовище, у якому користувач отримує доступ до особистого кабінету, реєструється або входить у вже створений обліковий запис. Після цього користувач створює або імпортує список корисних звичок, які він планує відстежувати, наприклад: «пити воду», «читати щодня», «медитувати» тощо.

Далі активується модуль налаштування параметрів звичок – визначення частоти виконання, встановлення нагадувань, прив’язки до часу доби чи конкретного контексту (геолокація, події календаря, індекс активності тощо). Після цього застосунок переходить у режим активного трекінгу, реєструючи кожну взаємодію користувача із завданнями, аналізуючи, які звички були виконані, пропущені чи відкладені.

Наступним етапом є обробка та збереження зібраних даних у базі даних користувача. Щоденно програма виконує аналітичний розрахунок успішності виконання звичок, відображає коротку статистику, графіки прогресу, а також надає користувачу зворотний зв’язок у вигляді мотиваційних повідомлень. Проте найважливішим і унікальним елементом алгоритму є блок кореляційного аналізу звичок.

Модуль аналізу кореляції працює у фоновому режимі, зчитуючи частотність, періодичність, зв’язок між звичками за часом виконання та контекстом. Наприклад, якщо користувач регулярно виконує звичку «ранкова

зарядка», і в ті самі дні значно частіше виконується звичка «здоровий сніданок», система зафіксує статистично значущий зв’язок між ними.

Після достатнього накопичення даних, застосунок формує аналітичні закономірності – наприклад, «виконання звички А підвищує ймовірність виконання звички В на 23%». Такі закономірності зберігаються, візуалізуються у вигляді графів або таблиць, і можуть бути використані для рекомендацій, комбінування звичок у звички-комплекси та створення персоналізованих маршрутів саморозвитку.

Рисунок 2.1 – Загальний алгоритм роботи

Для кращого розуміння принципів реалізації кореляційного аналізу звичок, було створено окремий допоміжний алгоритм (рис. 2.2). Його першим етапом є накопичення історичних даних про виконання звичок, з обов'язковою фіксацією часових міток, частоти, типу звички, психологічного стану користувача (за бажанням), та інших змінних.

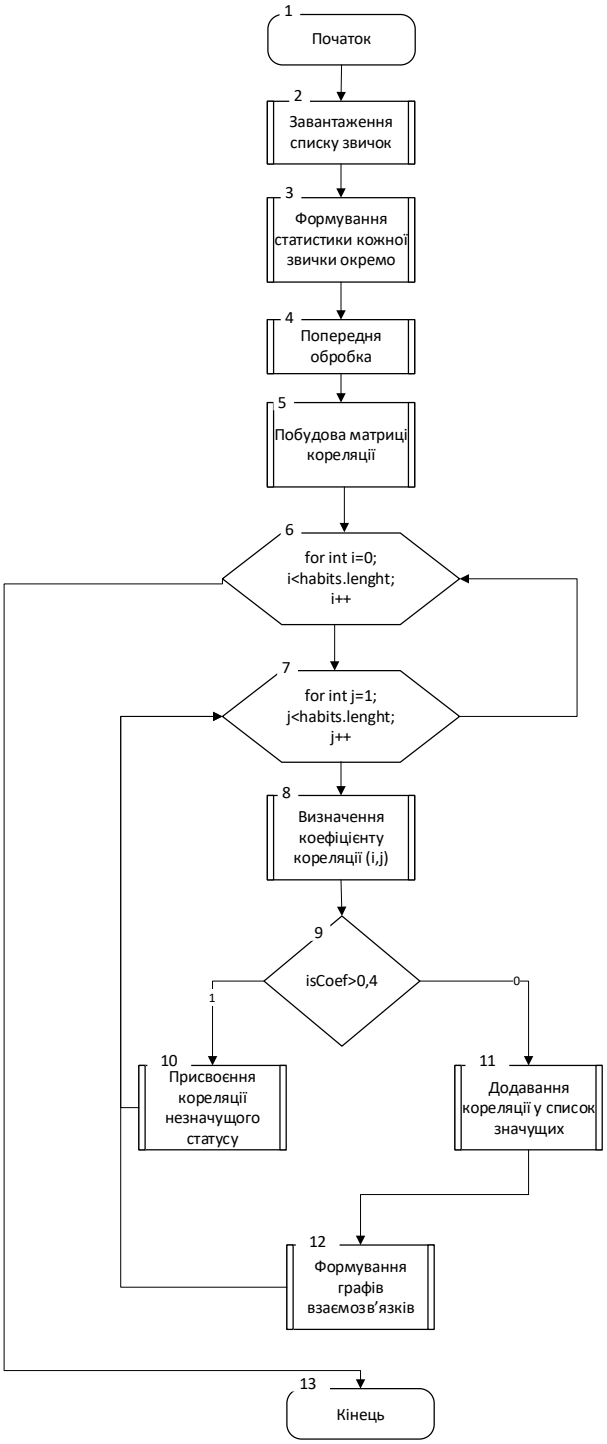


Рисунок 2.2 – Алгоритм побудови системи кореляційного аналізу звичок

Далі виконується попередня обробка даних – видалення пропущених значень, нормалізація даних, створення векторів виконання звичок. На наступному етапі застосовується алгоритм побудови кореляційної матриці. Використовуючи коефіцієнти Фішера, Пірсона або точкову бісерійну кореляцію (залежно від типу даних), система обчислює ступінь взаємозв’язку між кожною парою звичок. Якщо кореляція перевищує визначений поріг (наприклад, 0.4), вона визнається значущою. Визначені залежності автоматично передаються до модуля візуалізації, де формуються графи взаємозв’язків, що відображають, як звички впливають одна на одну.

Після визначення значущих зв’язків між звичками, система надає користувачу рекомендації – наприклад, поєднати певні звички в один блок (ранкову рутину), змінити порядок виконання або обрати інші тригери для запуску звички. Таким чином, розроблений алгоритм дозволяє не лише фіксувати поведінку користувача, а й активно адаптувати систему до його стилю життя, підвищуючи ефективність формування нових звичок за рахунок виявлення прихованих закономірностей.

Запропонований підхід дозволяє створити не просто трекер звичок, а інтелектуальну систему, що адаптується до поведінки користувача і допомагає йому оптимізувати власний маршрут особистого розвитку. Це суттєво розширює функціонал стандартних застосунків, надаючи індивідуалізований підхід до формування й підтримки здорових звичок.

2.2 Розробка методу кореляції корисних звичок

Для реалізації інтелектуальної системи підтримки формування корисних звичок у мобільному застосунку необхідно розробити метод, який дозволяє встановлювати кореляційні зв’язки між звичками користувача на основі його щоденних активностей. З цією метою був запропонований власний статистико-аналітичний підхід до оцінювання ступеня взаємозв’язку між звичками з використанням математичних критеріїв, таких як коефіцієнт кореляції Пірсона,

коефіцієнт рангової кореляції Спірмена, t-критерій Стьюдента та F-критерій Фішера.

Основною метою розробки є виявлення закономірностей між діями користувача, що сприяють або навпаки – заважають формуванню стабільних звичок, а також подальше використання цієї інформації для адаптації системи під індивідуальні потреби.

Першим кроком реалізації методу є формування структурованих даних про звички користувача. Для цього кожна звичка представлена у вигляді об’єкта, що містить такі параметри: назва звички, періодичність виконання, часові мітки, кількість спроб, рівень успішності та тип звички (фізична, ментальна, соціальна тощо). Дані зберігаються у вигляді бази даних, яка регулярно оновлюється під час використання мобільного застосунку.

Основою для оцінювання кореляцій є багатовимірна матриця звичок, яка формується на основі щоденних відміток користувача про виконання певної дії. Для кожного користувача створюється таблиця, де стовпці відповідають конкретним звичкам (наприклад, «випити воду», «читати книгу», «фізичні вправи»), а рядки – це дні, у які зафіксовано виконання звички (1 – виконано, 0 – не виконано).

Для вимірювання лінійного зв’язку між двома звичками A і B використовується коефіцієнт кореляції Пірсона [14]:

$$r_{AB} = \frac{\sum_{i=1}^{n}(A_i - \bar{A})(B_i - \bar{B})}{\sqrt{\sum_{i=1}^{n}(A_i - \bar{A})^2} * \sqrt{\sum_{i=1}^{n}(B_i - \bar{B})^2}} \quad (2.1)$$

, де A_i, B_i – значення виконання звичок у день i; $\bar{A}$, $\bar{B}$ – середні значення по кожній звичці.

Коефіцієнт $r_{AB} \in [-1\ ;\ 1]$, де значення, близькі до ±1, свідчать про сильну позитивну або негативну кореляцію.

Якщо між звичками не простежується лінійний зв'язок, але існує

монотонна залежність, застосовується коефіцієнт Спірмена [15]:

$$\rho_{AB} = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \quad (2.2)$$

, де d_i = ранг (A_i) − ранг (B_i); n – кількість спостережень.

Коефіцієнт Спірмена також належить до інтервалу від -1 до 1, але є стійкішим до аномалій та нерівномірних розподілів.

Щоб перевірити статистичну значущість обчисленого коефіцієнта кореляції, використовується t-критерій Стьюдента [16]:

$$t = \frac{r_{AB}\sqrt{n-2}}{\sqrt{1 - {r_{AB}}^2}} \quad (2.3)$$

, де: r_{AB} – обчислений коефіцієнт кореляції Пірсона, n – кількість днів у вибірці. Результат порівнюється з критичним значенням t_{crit}, яке залежить від рівня значущості (наприклад, $\alpha = 0{,}05$) та кількості ступенів свободи $n - 2$.

Для визначення, чи статистично відрізняється дисперсія виконання звички між двома різними групами днів (наприклад, будні та вихідні), використовується F-критерій:

$$\mathrm{F} = \frac{r_{AB}}{t} \quad (2.4)$$

F-критерій дозволяє виявити залежність виконання звички від контексту середовища або часу.

Особливістю запропонованого методу є врахування не лише статистичних значень, але й контексту, в якому виконувались звички. Наприклад, система враховує добу, день тижня, емоційний стан (якщо він відмічений користувачем), що дозволяє створити більш точну модель взаємозв’язків. Додатково в алгоритмі

реалізовано фільтрацію псевдокореляцій – випадків, коли дві звички співпадають у часі, але не мають причинно-наслідкового зв’язку.

В основі методу лежить побудова графа звичок, де вузли – це звички, а ребра – кореляційні зв’язки між ними з відповідними вагами. Застосування алгоритмів графової аналітики дозволяє знаходити кластери взаємопов’язаних звичок, які можна рекомендувати як зв’язки для одночасного формування (так звані «звички-супутники»).

Для автоматизації процесу була реалізована система адаптивних рекомендацій, що аналізує знайдені кореляції та генерує поради користувачу, наприклад: «Звичка X частіше виконується вранці після звички Y. Рекомендуємо об’єднати їх у ланцюг звичок». Такий підхід дозволяє не лише підвищити ефективність формування нових звичок, але й забезпечити стійке збереження вже сформованих.

Вибір використаних статистичних методів є обґрунтованим з точки зору їх здатності виявляти різні типи залежностей між ознаками (звичками) у вибірці бінарних даних. Зокрема, коефіцієнт кореляції Пірсона дозволяє виявити чітко виражені лінійні зв’язки між звичками. Наприклад, звички «ранкова зарядка» та «контрастний душ» можуть проявляти сильну позитивну лінійну кореляцію у тих користувачів, для яких ці дії є частинами єдиного ранкового ритуалу.

У свою чергу, коефіцієнт рангової кореляції Спірмена доцільно застосовувати у випадках, коли зв’язки між звичками не є лінійними, але мають стабільну узгодженість. Це може бути характерно, наприклад, для звички «читати книжку» та звички «писати щоденник», які можуть регулярно з’являтися у вечірній час, але не обов’язково щоденно.

Критерій Стьюдента виконує важливу функцію фільтрації: він дозволяє уникнути включення у фінальний список пов’язаних звичок тих зв’язків, які є випадковими та виникли через недостатню кількість спостережень або через викривлення розподілу. Таким чином, він підвищує достовірність аналітичних висновків.

Критерій Фішера, хоча й не входить безпосередньо у процедуру

кореляційного аналізу, використовується додатково для оцінювання варіативності виконання окремих звичок у різних контекстах. Наприклад, можна оцінити, чи значуще змінюється виконання звички у будні дні порівняно з вихідними, або в період стресу, що дозволяє ще глибше інтерпретувати поведінку користувача.

Отже, було виконано розробку методу кореляційного аналізу корисних звичок користувача у форматі мобільного застосунку з використанням інструментів математичної статистики. Запропонований алгоритм охоплює повний цикл аналізу — від зчитування бінарних векторів до обчислення коефіцієнтів кореляції та перевірки статистичної значущості виявлених зв’язків. Використано формули та критерії, що забезпечують надійність і достовірність результатів аналізу, зокрема: коефіцієнти Пірсона і Спірмена, t-критерій Стьюдента та F-критерій Фішера. Реалізований алгоритм дозволяє автоматизовано визначати вплив однієї звички на іншу, що є основою для побудови інтелектуальної системи рекомендацій. Це створює підґрунтя для глибшого розуміння користувачем своєї поведінки та ефективнішого формування стійких корисних звичок.

2.3 Розробка методу моніторингу виконання корисних звичок

Моніторинг виконання корисних звичок є ключовим елементом у побудові цифрових систем підтримки самоменеджменту, що дозволяють відстежувати поведінкову активність користувача, здійснювати аналіз динаміки виконання звичок та виявляти взаємозв'язки між ними. Для реалізації цього процесу необхідно розробити математично формалізований метод, який дозволить визначати виконання звички, її частотність, контекстуальні умови та статистичні закономірності.

На першому етапі формується двійкова матриця виконання звичок. Нехай $H = [h_{ij}]$, де $h_{ij} \in \{0,1\}$ означає факт виконання і-тої звички у j-тий день. Таким чином, кожен рядок H_i представляє собою часовий вектор активності для

відповідної звички.

Для визначення інтенсивності виконання звички i за період T, використовується формула середньої частоти:

$$f_i = \frac{1}{T}\sum_{j=1}^{T} h_{ij} \tag{2.5}$$

Далі, для аналізу взаємозв'язків між звичками, будується кореляційна матриця $C = [c_{ik}]$, де кожен елемент c_{ik} відображає силу статистичного зв'язку між звичками i та k. Кореляційний коефіцієнт Пірсона обчислюється за формулою:

$$c_{ik} = \frac{\sum_{j=1}^{T}(h_{ij} - \overline{h_\iota})(h_{kj} - \overline{h_k})}{\sqrt{\sum_{j=1}^{T}(h_{ij} - \overline{h_\iota})^2} * \sqrt{\sum_{j=1}^{T}(h_{kj} - \overline{h_k})^2}} \tag{2.6}$$

, де $\overline{h_\iota}$ – середнє значення по рядку H_i.

Для візуалізації взаємозв'язків будується зважений неорієнтований граф $G = (V, E)$, де V – множина звичок, E – множина ребер між ними, а вага ребра $w_{ik} = |c_{ik}|$. Це дозволяє застосовувати кластерний аналіз для виявлення груп звичок, що формують взаємозалежні патерни поведінки. Щоб оцінити стабільність виконання звички в часовому аспекті, вводиться метрика варіації виконання v_i:

$$v_i = \sqrt{\frac{1}{T}\sum_{j=1}^{T}(h_{ij} - \overline{h_\iota})^2} \tag{2.7}$$

Наступним кроком є побудова функції адаптивного нагадування. Нехай $r_i(j)$ – функція ймовірності нагадування для звички i у день j. Її значення залежить від частоти виконання, стабільності та сили кореляції з іншими

звичками, що вже були активовані. Математична модель виглядає так, де α, β, γ – вагові коефіцієнти, що налаштовуються емпірично:

$$r_i(j) = \alpha f_i + \beta(1 - v_i) + \gamma \sum_{k \neq i} c_{ik} * h_{ik} \quad (2.8)$$

Для перевірки достовірності кореляцій між звичками проводиться статистичний тест Стьюдента, де t_{ik} порівнюється з критичним значенням t_{crit} для вибраного рівня значущості:

$$t_{ik} = \frac{c_{ik} * \sqrt{T - 2}}{\sqrt{1 - {c_{ik}}^2}} \quad (2.9)$$

Реалізація цього методу в програмному середовищі мобільного застосунку забезпечується модульною структурою: модуль логування подій, модуль аналітики, модуль кореляції та модуль візуалізації. Завдяки формалізованим алгоритмам користувач отримує адаптивні інтерфейсні елементи, графи звичок та рекомендації на основі обчисленої поведінкової моделі.

Таким чином, запропонований метод моніторингу забезпечує систематичний підхід до аналізу звичок, дозволяє виявляти приховані залежності та створювати персоналізовані механізми підтримки користувача у процесі самовдосконалення.

2.4 Висновки

У другому розділі було розроблено основний алгоритм роботи програмних засобів та алгоритм побудови кореляційного аналізу звичок. Також було проведено розробку методу кореляційного аналізу для виявлення взаємозв’язків між корисними звичками користувача та методу моніторингу виконання корисних звичок. Розроблені методи, використовують обробку бінарних

векторів активності, виконують розрахунок коефіцієнтів кореляції Пірсона та Спірмена, а також застосовує критерій Стьюдента для фільтрації статистично незначущих зв’язків.

3 РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ КОРИСНИМИ ЗВИЧКАМИ

1.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Під час розробки мобільного застосунку для управління корисними звичками важливо підібрати сучасні й оптимальні технології, які дозволять реалізувати як логіку застосунку, так і зручний інтерфейс користувача, забезпечивши швидкість, надійність та масштабованість. Розробка такого застосунку передбачає створення багатомодульної структури, в якій кожен компонент відповідає за певний набір функцій: облік звичок, аналіз даних, персоналізовані поради, візуалізацію результатів та збереження історії активності.

Для реалізації користувацького інтерфейсу було обрано Flutter [17] – кросплатформену фреймворк-технологію від Google, що використовує мову програмування Dart. Flutter дозволяє створювати сучасні, адаптивні й анімовані інтерфейси з єдиною кодовою базою для Android та iOS. Його перевагами є гарна продуктивність, широка екосистема віджетів, гнучке керування станом інтерфейсу, а також активна спільнота розробників. Завдяки Flutter можлива реалізація динамічних елементів управління звичками, персоналізованих підказок і ефективної навігації додатку.

Для логіки застосунку, особливо для реалізації алгоритму кореляції звичок, використовується мова програмування Kotlin [18] – сучасна мова, офіційно підтримувана для Android-розробки. Kotlin має лаконічний синтаксис, підтримку функціонального програмування та високу сумісність з Java. У контексті проєкту Kotlin використовується для обробки бінарних векторів активності звичок, обчислення коефіцієнтів кореляції Пірсона і Спірмена, а також реалізації перевірки статистичної значущості за критерієм Стьюдента. Це дозволяє ефективно інтегрувати математичні модулі без необхідності залучення

зовнішніх інтерпретованих мов, таких як Python.

З метою забезпечення збереження локальних даних користувача обрано базу даних SQLite, що підтримується як у Flutter, так і у Kotlin. Це дозволяє зберігати матрицю кореляцій звичок, історію виконання, налаштування користувача та інші критично важливі дані без необхідності підключення до мережі. Перевагою SQLite є її легкість, вбудованість та висока швидкодія при роботі з відносно невеликими обсягами структурованої інформації.

Для візуалізації результатів аналізу, таких як кореляційна матриця та динаміка змін звичок, у Flutter використовується бібліотека fl_chart, яка забезпечує побудову інтерактивних графіків та діаграм без втрати продуктивності. Ця бібліотека дозволяє відображати дані у вигляді теплових карт, гістограм, лінійних графіків та інших візуальних форматів, що сприяє кращому розумінню статистичних зв’язків між звичками.

Зовнішні сервіси або серверна частина в межах поточної реалізації не використовуються, оскільки вся логіка обробки виконується на пристрої користувача. Такий підхід дозволяє забезпечити конфіденційність даних та автономну роботу застосунку. У перспективі можливе розширення архітектури за рахунок використання хмарних сервісів для колективного аналізу даних або резервного копіювання.

Отже, поєднання таких технологій, як Flutter для інтерфейсу, Kotlin для обробки статистичних алгоритмів, SQLite для локального збереження даних та fl_chart для інтерактивної візуалізації, дозволяє створити ефективний, сучасний та функціональний мобільний застосунок для управління корисними звичками. Такий підхід забезпечує повну автономність програми, високу продуктивність, зручність для користувача та можливість подальшого масштабування.

1.2 Вибір середовища розробки

У процесі розробки мобільного застосунку для управління корисними звичками особливу увагу слід приділити вибору середовища розробки, адже саме

від цього залежатиме ефективність написання коду, підтримка проєкту, зручність налагодження та тестування, а також швидкість реалізації функціональних можливостей. Існує велика кількість сучасних середовищ розробки, орієнтованих на мобільні платформи Android та iOS. Проведемо порівняльний аналіз найбільш поширених з них.

Android Studio [19] – офіційне інтегроване середовище розробки (IDE) від Google, що є стандартом де-факто для створення застосунків під Android. Android Studio базується на IntelliJ IDEA і надає розробникам потужний набір інструментів, включаючи редактор коду з підсвіткою синтаксису, засоби автодоповнення, емулятор Android-пристроїв, профілювання продуктивності та налагодження застосунку в режимі реального часу. Android Studio підтримує мови Kotlin та Java, що робить її особливо зручною для створення нативних Android-застосунків. Середовище також забезпечує глибоку інтеграцію з Google-послугами, Android SDK та бібліотеками Jetpack, що дозволяє реалізовувати сучасні підходи до архітектури та інтерфейсу користувача. Однією з ключових переваг Android Studio є наявність вбудованого емулятора, який дозволяє швидко тестувати застосунок на різних віртуальних пристроях. Середовище також має розвинену систему збірки Gradle, яка спрощує керування залежностями, налаштування проєктів та автоматизацію процесів. Крім того, Android Studio дозволяє інтегрувати сторонні бібліотеки, використовувати інструменти для юніт-тестування та CI/CD. Це середовище є безкоштовним та має активну спільноту розробників.

Visual Studio with Xamarin [20] – це потужне кросплатформне середовище розробки від компанії Microsoft, яке дозволяє створювати мобільні застосунки на C# з використанням платформи Xamarin. Основною перевагою цього підходу є можливість повторного використання значної частини коду між платформами Android та iOS. Visual Studio надає інтегроване середовище з підтримкою емуляторів, засобів налагодження, візуального дизайну інтерфейсів і широкого вибору бібліотек. Інструменти Xamarin.Forms дозволяють розробникам створювати інтерфейси, що адаптуються до обох платформ, з використанням

єдиного коду. Середовище Visual Studio підтримує високу продуктивність, має розширену систему доповнень, підтримує хмарні сервіси Azure, а також дозволяє легко реалізувати інтеграцію з базами даних, сервісами автентифікації та REST API. Проте слід зазначити, що для розробки під iOS необхідне наявність Mac-системи та підключення до Xcode. Також Xamarin має деякі обмеження у порівнянні з нативною розробкою, особливо у реалізації складних або специфічних інтерфейсних елементів.

Visual Studio Code [21] – це легке, кросплатформне, безкоштовне середовище розробки, створене компанією Microsoft. Хоча VS Code не є повноцінною IDE, воно підтримує мобільну розробку за допомогою додаткових розширень, таких як Flutter, React Native, Cordova, або навіть підключення до Android SDK. Основна перевага Visual Studio Code полягає в його гнучкості: за допомогою плагінів можна створити середовище під будь-який стек технологій, у тому числі Dart+Flutter або JavaScript+React Native. VS Code має вбудовану підтримку підсвітки синтаксису, автодоповнення коду, інтеграцію з Git, термінал, дебаггер та широкий каталог розширень. Завдяки функції hot reload при роботі з Flutter, VS Code дозволяє значно прискорити розробку UI. Розробники також мають можливість одночасно тестувати застосунок у кількох емуляторах, що розширює зручність у створенні адаптивного інтерфейсу. Незважаючи на те, що VS Code не має власного емулятора, його можна інтегрувати з Android Emulator або симулятором iOS на MacOS.

Проведемо порівняльну характеристику згаданих середовищ розробки у таблиці 3.1.

Таблиця 3.1 – Порівняння середовищ розробки мобільних застосунків

Характеристика	**Android Studio**	**Visual Studio**	**Visual Studio Code**
Підтримка Android	+	+	+
Підтримка iOS	-	+	-
Кросплатформеність	+	+	+

Продовження таблиці 3.1

Характеристика	**Android Studio**	**Visual Studio**	**Visual Studio Code**
Безкоштовність	+	-	+
Підтримка сучасних UI-фреймворків	+	-	+
Розширюваність	+	+	+
Підтримка емуляції та налагодження	+	+	-
Сумарний бал	6	5	5

Таким чином, серед наведених середовищ розробки найбільш універсальним, гнучким та ефективним є Android Studio з використанням Flutter та мов програмування Dart та Kotlin.

3.3 Розробка модуля кореляції корисних звичок

Головною задачею модуля кореляції корисних звичок є виявлення статистично значущих зв’язків між сформованими звичками користувача для подальшої персоналізації рекомендацій у мобільному застосунку. Для реалізації цієї задачі було створено клас HabitCorrelator (рис. 3.1), який приймає на вхід список звичок разом із історією їх виконання, що подається у вигляді матриці виконання за днями.

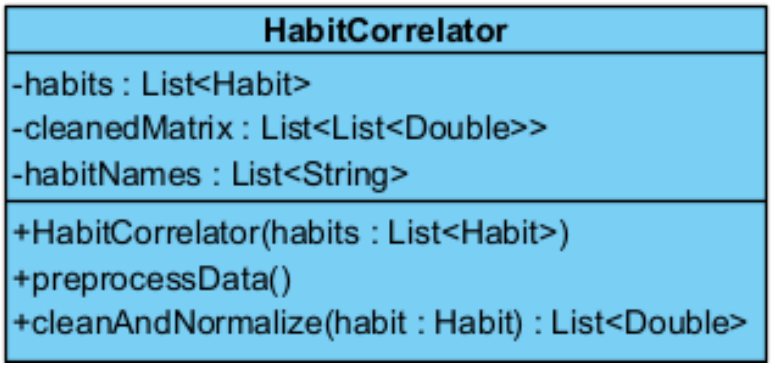


Рисунок 3.1 – Клас HabitCorrelator

При створенні та ініціалізації класу автоматично виконується попередня обробка даних: нормалізація, виявлення пропусків, а також побудова матриці бінарних або інтервальних значень залежно від типу звички.

Крім того, було розроблено окрему функцію analyzeHabits(), яка аналізує наявні дані на предмет можливих помилок у введенні або виявлення неінформативних звичок. Функція видаляє ознаки, які мають нульову варіацію або є надто рідкісними, а також виявляє аномальні записи, що можуть суттєво вплинути на кореляційний аналіз (рис. 3.2).

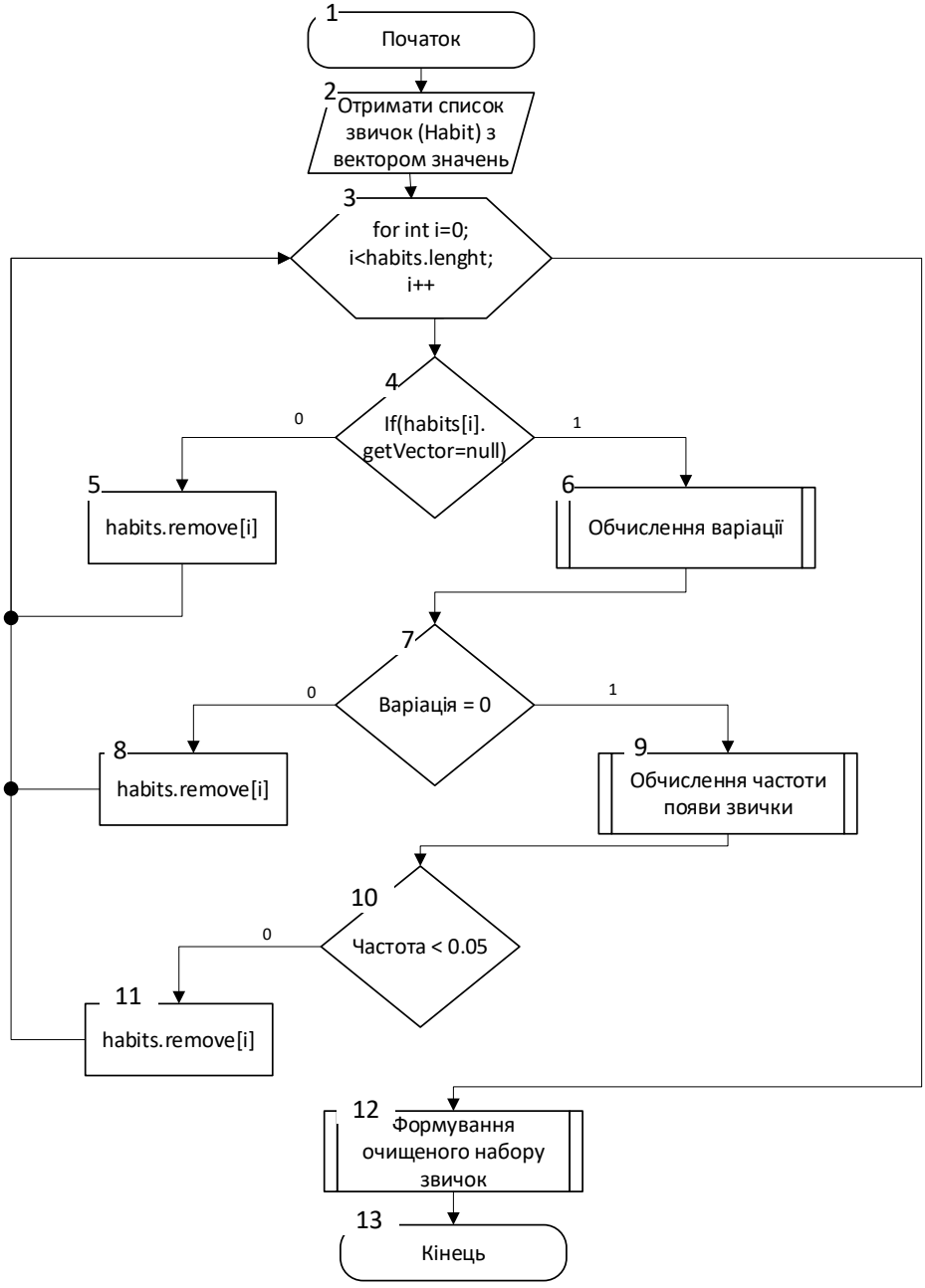


Рисунок 3.2 – Функція аналізу вхідних даних про звички

Наступним кроком стало створення методів обчислення коефіцієнтів кореляції. Зокрема, було реалізовано функції для розрахунку коефіцієнтів кореляції Пірсона та Спірмена, а також перевірки їхньої статистичної значущості з використанням критерію Стьюдента. На рисунку 3.3 зображено функцію computeCorrelationMatrix(), яка приймає матрицю звичок, обирає метод кореляції та повертає дві матриці: коефіцієнтів та відповідних p-значень. Якщо p-значення менше за 0.05, відповідна кореляція вважається статистично значущою.

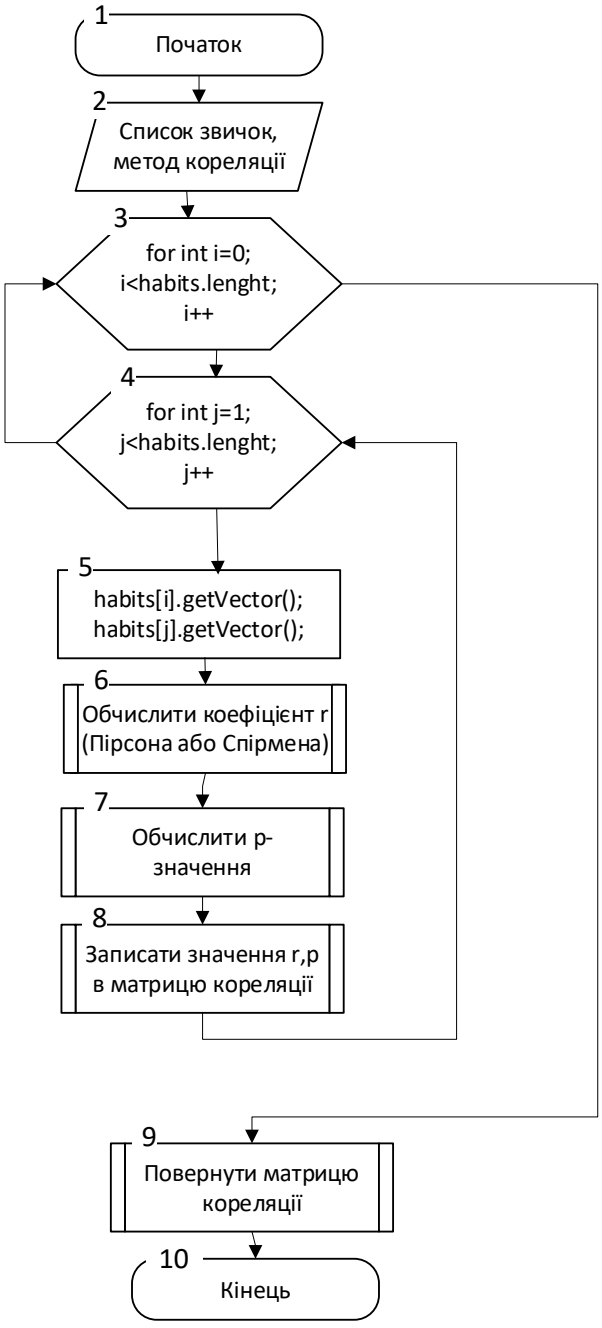


Рисунок 3.3 – Функція побудови матриці кореляцій та p-значень

Для побудови структури значущих зв’язків між звичками було реалізовано візуалізацію у вигляді графа. На рисунку 3.4 представлена функція visualizeHabitGraph(), яка будує неорієнтований граф на основі кореляційної матриці, у якому вузли відповідають звичкам, а ребра – наявності значущого зв’язку між ними. Вага ребра визначається абсолютним значенням коефіцієнта кореляції, а його колір – позитивною або негативною залежністю.

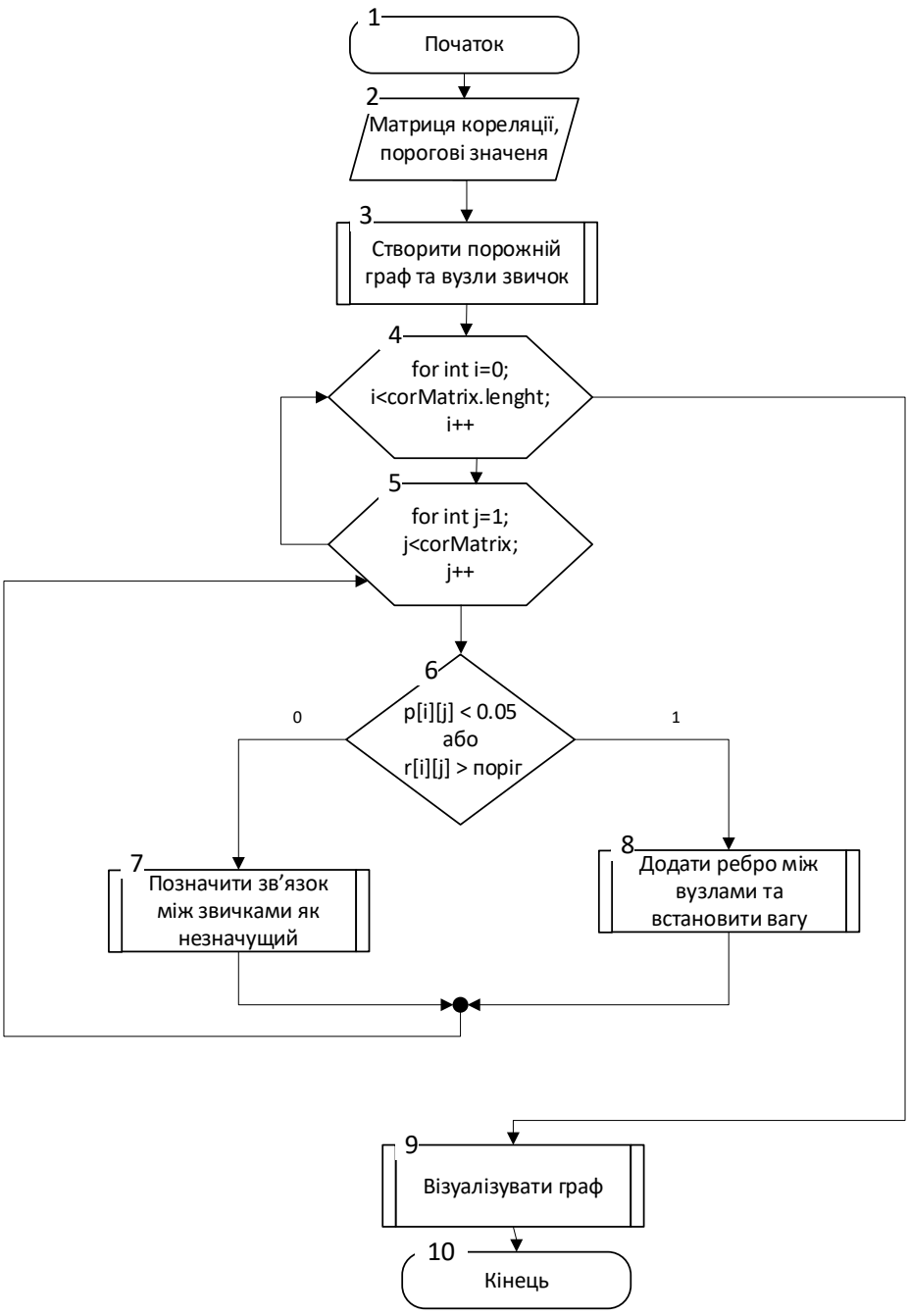


Рисунок 3.4 – Візуалізація кореляційного графа звичок

Окрему увагу було приділено модулю статистичної перевірки гіпотез щодо

залежності між групами звичок. На рисунку 3.5 подано реалізацію функції evaluateHabitGroups(), яка використовує дисперсійний аналіз (ANOVA) та критерій Фішера для виявлення залежностей між групами звичок (наприклад, фізична активність, харчування, сон). Для кожної групи формується масив середніх значень звичок, після чого перевіряється гіпотеза про рівність середніх. Якщо значення критерію Фішера перевищує критичне – між групами звичок існує статистично значущий зв’язок.

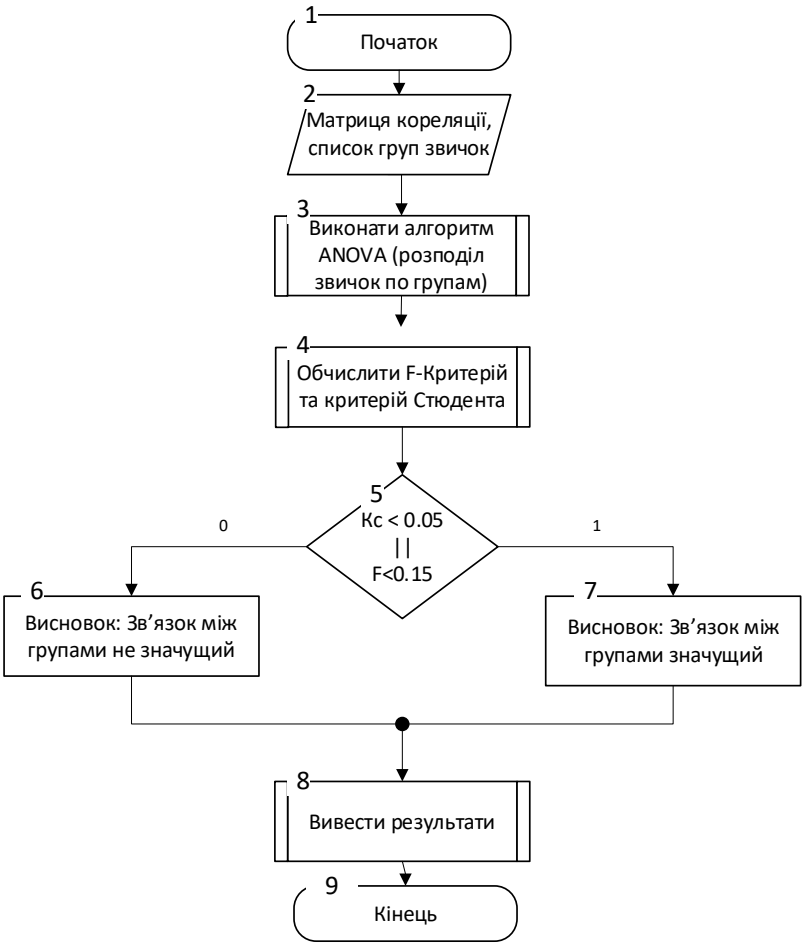


Рисунок 3.5 – Функція оцінювання взаємозв’язку між групами звичок

Таким чином, розроблений модуль кореляції корисних звичок дозволяє не лише аналізувати структуру сформованих поведінкових шаблонів, а й формувати індивідуальні пропозиції щодо вдосконалення стилю життя. Це значно підвищує ефективність мобільного застосунку, роблячи його не лише трекером, але й інтелектуальним порадником для користувача.

3.4 Висновки

У третьому розділі було обґрунтовано вибір технологій та інструментів, що використовувались для реалізації мобільного застосунку управління корисними звичками. Зокрема, було обрано середовище розробки Android Studio з використанням мови програмування Dart та фреймворку Flutter для створення кросплатформеного інтерфейсу користувача. Розроблено модуль кореляції корисних звичок, необхідний для персоналізованого аналізу звичок користувача, що забезпечують інтелектуальну підтримку в процесі формування корисних звичок.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Опис процесу тестування програмного забезпечення

Тестування програмного забезпечення є критично важливим етапом розробки мобільного застосунку для управління корисними звичками, оскільки саме воно дозволяє перевірити коректність роботи усіх функціональних модулів, стабільність застосунку в різних умовах та відповідність функціоналу поставленим вимогам.

Основною метою тестування [22] – є виявлення помилок і дефектів у реалізації логіки формування, відстеження та аналізу звичок, а також перевірка зручності користувацького інтерфейсу на різних мобільних пристроях.

Перед початком тестування було сформовано тестове середовище, що включає емулятор Android, а також фізичні пристрої з різними розмірами екранів. У тестуванні брали участь модулі, написані за допомогою Flutter (Dart) з використанням Firebase, SQLite та графічних бібліотек для побудови інтерфейсу. Була підготовлена документація, що включає специфікації функціональних вимог, технічне завдання, тест-кейси та чеклісти перевірок.

Процес тестування включав кілька основних етапів.

Функціональне тестування – цей етап передбачав перевірку основного функціоналу застосунку: реєстрації та авторизації користувача, створення та редагування звичок, налаштування нагадувань, перегляду статистики та рекомендацій. Окремо тестувалась робота модуля кореляції звичок: виявлення зв’язків, побудова графів залежностей та формування персоналізованих порад. Також тестувались сценарії негативного вводу – наприклад, введення некоректних або порожніх полів, надто довгих назв звичок або неочікуваних символів.

Тестування на відмову та стійкість – було проведено ряд тестів, що імітували непередбачувані ситуації: раптове закриття застосунку, втрата підключення до інтернету під час синхронізації з Firebase, перевантаження бази

даних локальними записами. Система була протестована на відновлення після збоїв, збереження даних після перезапуску та коректне відображення помилок користувачу.

Тестування продуктивності – під час цього етапу перевірялась швидкодія застосунку на різних пристроях. Було виміряно час відгуку при відкритті екранів, збереженні нових звичок, завантаженні статистики та побудові графіків зв’язків. Особливу увагу було приділено оптимізації запитів до бази даних та затримкам при візуалізації графів кореляцій.

Тестування сумісності – було протестовано роботу застосунку на Android (версії 8–14) та iOS (версії 13–17), на пристроях з різними розмірами екрана та щільністю пікселів. Також перевірялась сумісність з темною та світлою темами, системними шрифтами та мовними локалізаціями. Жодних критичних збоїв або несумісностей виявлено не було.

Таким чином, процес тестування дозволив виявити низку дрібних багів, що були оперативно усунені. Тестування підтвердило стабільну роботу мобільного застосунку в основних сценаріях використання та його готовність до публікації в маркетплейсах.

4.2 Тестування програмного забезпечення

У рамках тестування мобільного застосунку для управління корисними звичками було проведено низку тестів, спрямованих на перевірку функціональності, коректності введених даних, стабільності роботи інтерфейсу, а також на відповідність заявленим функціональним вимогам. Тестування проводилося на пристроях з операційною системою Android. Усі тести виконувалися вручну відповідно до розробленого плану тестування.

Першим етапом стало тестування коректності введення дати. Було перевірено, чи система приймає лише валідні формати дати (наприклад, "01.06.2025"), і чи відображає повідомлення про помилку у випадку введення дати в неправильному форматі або у вигляді недійсного значення (наприклад,

"31.02.2025"). Застосунок успішно обробляв такі випадки, повідомляючи користувача про необхідність введення коректної дати.

Аналогічним чином перевірялося введення часу. Було протестовано реакцію застосунку на коректні значення (наприклад, "08:30", "23:59") та на помилкові (наприклад, "25:00", "99:99"). У всіх випадках застосунок правильно інформував користувача про помилку та не дозволяв зберегти значення, що виходить за межі допустимого (рис. 4.1).

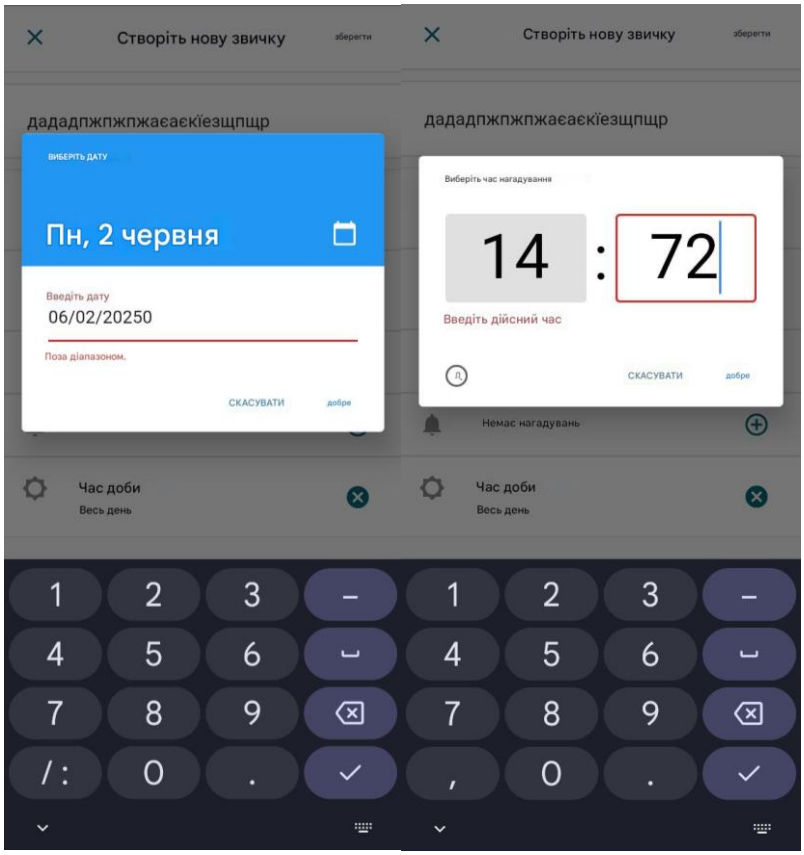


Рисунок 4.1 – Перевірка формату дати та часу

Наступним кроком було тестування функціоналу додавання власної звички. При створенні нової звички користувач мав можливість ввести назву звички, вибрати час нагадування, встановити повторюваність (щоденно, раз на тиждень, у вибрані дні тощо), а також задати мету звички (наприклад, "випити 2 л води щодня"). Застосунок дозволяв зручно обирати параметри з відповідних меню. Результати тестування підтвердили правильність збереження всіх введених параметрів та їхнє подальше відображення у головному інтерфейсі застосунку.

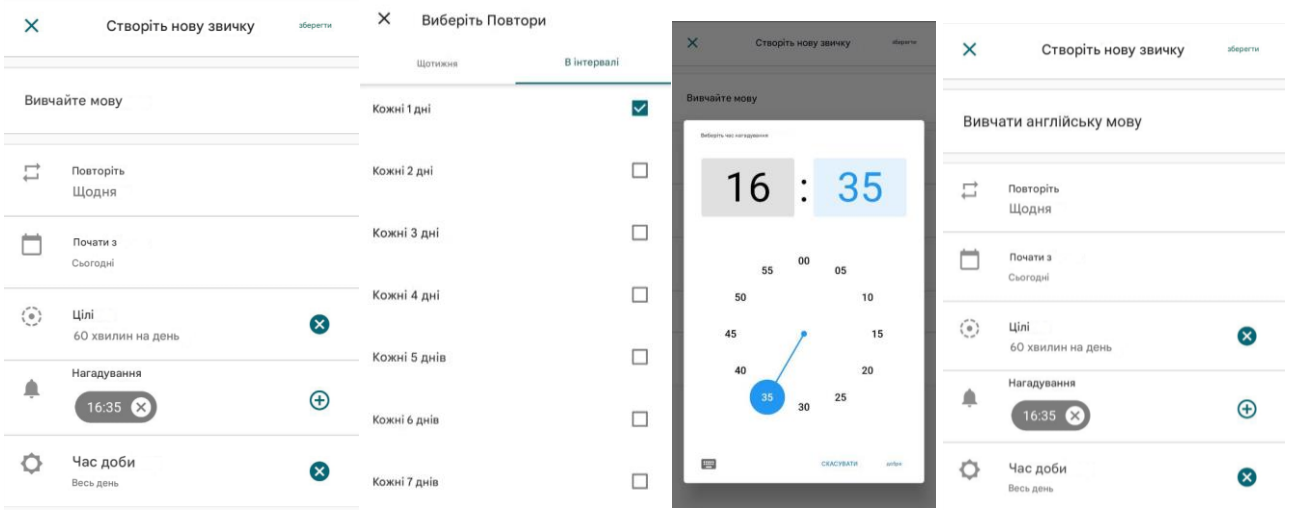


Рисунок 4.2 – Додавання нової корисної звички у застосунок

Окремо тестувалося виконання звички, тобто відмітка про її виконання в поточний день (рис. 4.3). Користувач натискає кнопку «Виконано», і звичка фіксується як виконана на сьогоднішню дату. Після цього у головному вікні застосунку змінюється її статус. Усі такі взаємодії з інтерфейсом відбувалися без затримок, а статус виконувався миттєво. Також, якщо звичка передбачає поетапне виконання, у програмі було реалізовано шлаку прогресу виконання звички.

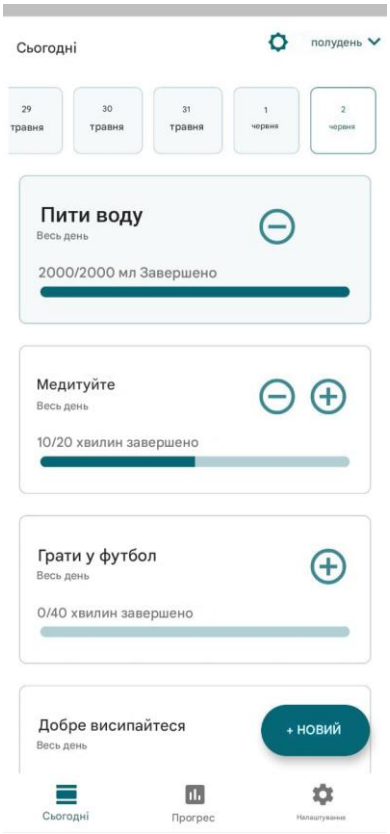


Рисунок 4.3 – Результат тестування відміток про виконання

Додатково було протестовано функціонал додавання звички з шаблонів (рис. 4.4). У застосунку передбачено 6 категорій шаблонів корисних звичок. Для кожної категорії користувач міг обрати одну з типових звичок, встановити її розклад та мету. Під час тестування перевірено, чи всі шаблони завантажуються, чи коректно працює фільтрація за категоріями та чи можливо редагувати параметри шаблонної звички перед її додаванням до головного списку.

Також перевірено створення власної звички без використання шаблонів. Застосунок коректно обробляв усі введені користувачем параметри, а після збереження нова звичка з’являлася в головному списку. Жодних технічних помилок або візуальних конфліктів при цьому не виявлено.

Рисунок 4.4 – Додавання звичок із шаблонів

Далі було протестовано функціонал перегляду прогресу звичок (рис. 4.5). Застосунок дозволяв переглядати статистику виконання на сьогодні, за останній тиждень, за місяць, а також за періоди у 3, 6 і 12 місяців. Графіки формувалися автоматично на основі даних, зібраних із трекера звичок. Було перевірено, що при переході між часовими інтервалами графіки оновлюються, зберігаючи точність та коректну візуалізацію (наприклад, стовпчикові діаграми або лінійні графіки).

Особливу увагу приділено трекеру виконання конкретної звички. У цьому режимі користувач міг побачити хронологію виконання обраної звички, включно з днями, коли вона була виконана, та днями пропуску. Перевірено точність обліку таких дій, а також правильність оновлення даних у загальній статистиці.

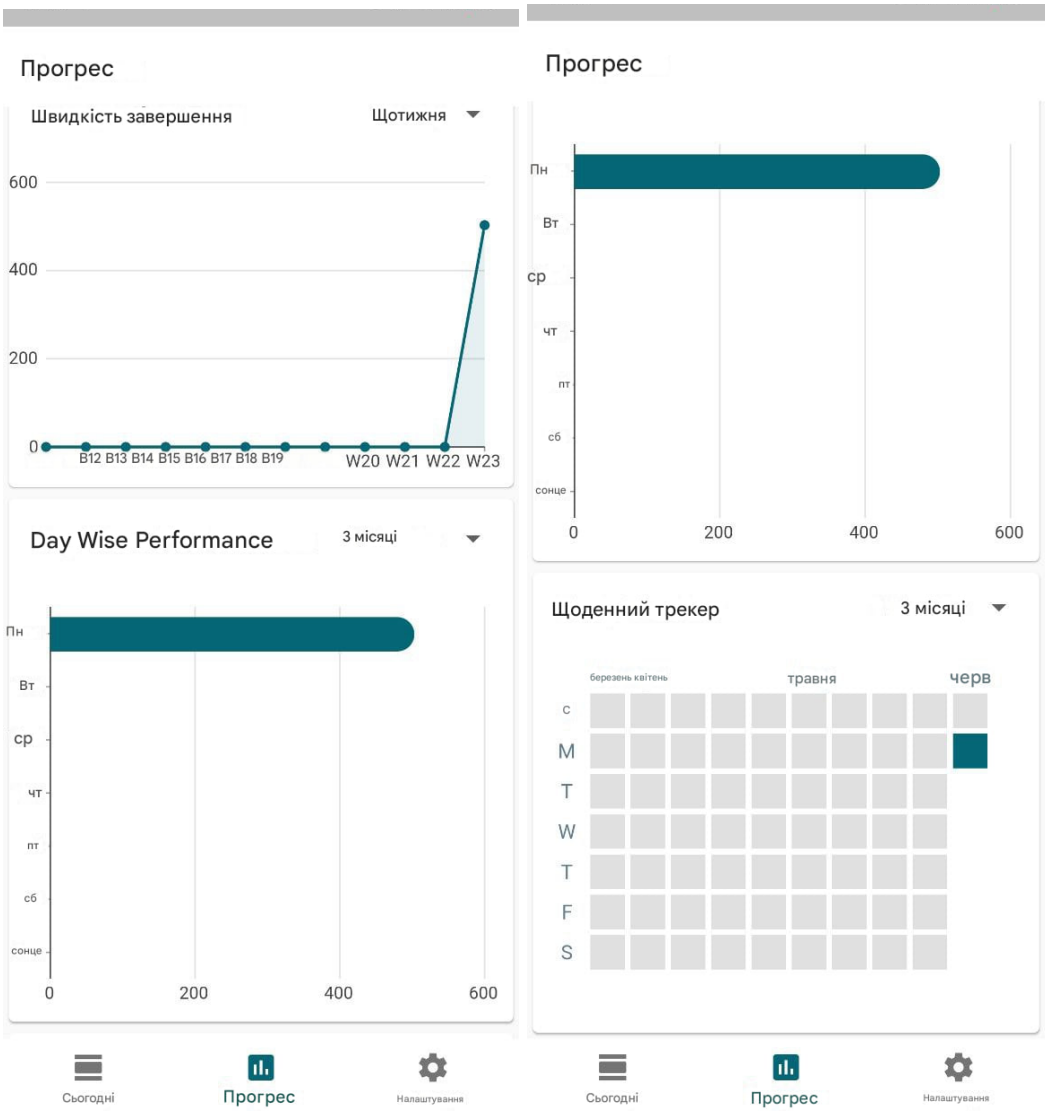


Рисунок 4.5 – Перегляд статистики звичок

Після додавання кількох звичок головне вікно застосунку відображало повний список активних звичок з відповідними індикаторами прогресу, повторюваністю, метою та часом нагадування. Перевірено також можливість зміни мови застосунку а також вигляд головного вікна в темній темі: усі елементи інтерфейсу адаптувалися до темного фону, не було помітно проблем з читабельністю або з відображенням графіки (рис. 4.6).

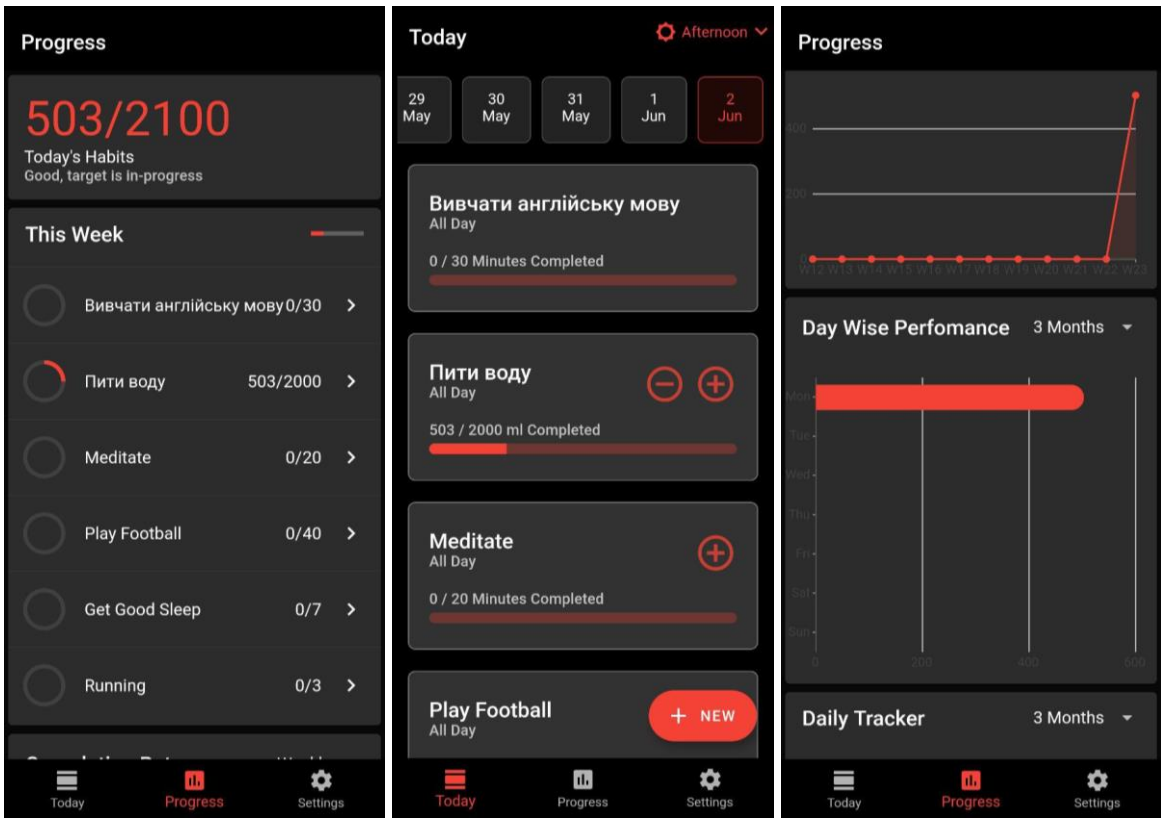


Рисунок 4.6 – Вигляд застосунку з темною темою

Останнім етапом тестування стала перевірка модуля перегляду кореляції звичок. Система автоматично аналізувала частоту виконання звичок та виявляла статистичні зв’язки між ними, наприклад, підвищення регулярності у фізичних вправах при одночасному веденні щоденника. Результати відображались у вигляді графіків або діаграм з підписами коефіцієнтів Пірсона або Спірмена, що дозволяло користувачу краще зрозуміти взаємозв’язки між власними звичками. Модуль працював коректно, оновлюючи дані в реальному часі при додаванні нових звичок або зміні прогресу виконання (рис. 4.7).

Рисунок 4.7 – Тестування кореляції звичок

У результаті тестування мобільного застосунку було підтверджено стабільну роботу основного функціоналу: перевірка введення дати й часу, додавання власних звичок, вибір повторюваності, цілі та часу нагадування, імпорт шаблонних звичок, виконання звичок, перегляд прогресу та трекінгу у зручному інтерфейсі.

Програма реагує на помилки введення та забезпечує користувача зрозумілими повідомленнями і підказками. Додатково успішно реалізовано і протестовано функціонал кореляційного аналізу звичок, що дає змогу

користувачеві візуально оцінити залежності між виконанням різних звичок за допомогою кореляційної матриці, діаграм і текстових пояснень. Це підвищує аналітичну цінність застосунку та дозволяє краще формувати корисні звички на основі виявлених взаємозв’язків.

4.3 Висновки

У четвертому розділі було проведено тестування мобільного застосунку для управління корисними звичками. Було перевірено правильність обробки введених даних (дата, час), функціонал додавання звичок — як власних, так і з шаблонів — а також виконання звичок, перегляд трекінгу прогресу та взаємодію з інтерфейсом застосунку. Тестування підтвердило стабільність і коректність роботи програмного продукту на всіх ключових етапах його використання. Результати тестування довели ефективність застосунку у створенні, контролі та аналітичному супроводі корисних звичок користувача.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі було розроблено методи та програмне забезпечення для моніторингу та управління корисними звичками з використанням засобів мобільної розробки та інструментів математичного аналізу. Робота охопила повний цикл створення мобільного застосунку: від аналізу предметної галузі та порівняння аналогів до розробки інноваційних методів обробки даних і тестування реалізованого програмного продукту.

Проведено аналіз сучасних мобільних застосунків для формування звичок, що дозволило ідентифікувати їх обмеження, зокрема відсутність персоналізованої аналітики, автоматизованого формування рекомендацій та моделювання взаємозалежностей між звичками.

Уперше запропоновано метод кореляційного аналізу виконання звичок із використанням коефіцієнтів Пірсона та Спірмена, t-критерію Стьюдента та F-критерію Фішера. Особливістю підходу є побудова неорієнтованого зваженого графа звичок, де зв’язки між звичками візуалізуються як ребра з вагами, що відповідають силі кореляції. Це дозволяє виявити закономірності у поведінці користувача та автоматизувати рекомендації щодо вдосконалення структури звичок.

Також уперше розроблено математично формалізований метод моніторингу виконання звичок з урахуванням адаптивного механізму нагадувань, що базується на частоті виконання, стабільності патернів та взаємозв’язках із іншими звичками. Це дозволило реалізувати адаптивну логіку сповіщень, яка підвищує ефективність нагадувань без перевантаження користувача.

Розроблено мобільний застосунок із використанням технологій Flutter (інтерфейс), Kotlin (логіка кореляційного аналізу), SQLite (зберігання даних) та fl_chart (візуалізація). Проведене тестування підтвердило функціональність і достовірність реалізованих методів аналізу звичок. Розроблена система дозволяє не лише фіксувати виконання звичок, а й адаптуватися до стилю життя

користувача, пропонуючи рекомендації та індивідуалізовані стратегії формування звичок.

Практичне значення результатів полягає у створенні інтелектуального інструменту підтримки особистісного розвитку, що може бути використаний у сфері охорони здоров’я, освіти, корпоративного менеджменту або як особистий асистент.

Ключові слова: мобільний застосунок, корисні звички, кореляційний аналіз, графова модель, інтелектуальний трекінг, візуалізація, моніторинг поведінки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Романюк О.В., Стахов О.Я. та Гулій І.Ю. Використання інтелектуальних мобільних технологій для управління поведінковими звичками / 2 Міжнародна науково-практична конференція «Innovative Approaches in Modern Science and Technology» / Лісабон, Португалія, 2025, с. 75–77.

2. M. P. Sokolova Influence of lifestyle and environmental habits on the health of youth (literature review) / *Environ. & Health*, № 3, с. 72–80, 2024.

3. Рудюк В. В. та Коц В.П. ЯК СФОРМУВАТИ КОРИСНІ ЗВИЧКИ / Scientific Research and Innovation: Proceedings of the 4th International Scientific and Practical Internet Conference / , April 3-4, 2025. FOP Marenichenko VV, Dnipro, Ukraine, 2025, с. 147-155.

4. Кошель В. М., Мацак Я. С. та Мамай Ю. М. ІНТЕГРАЦІЯ ЦИФРОВОГО АНІМАЦІЙНОГО КОНТЕНТУ В ОСВІТНІЙ ПРОСТІР ЗДО ЯК КОМПОНЕНТ ФОРМУВАННЯ ЗДОРОВ'ЯЗБЕРЕЖУВАЛЬНОЇ КОМПЕТЕНТНОСТІ / VIII International Scientific and Theoretical Conference «Modernization of today's science: experience and trends» / Glasgow, Scotland, UK, 30.05.2025. p. 270.

5. Чому важливо привчати себе до корисних звичок з Нового року та як це зробити? [Електронний ресурс] – Режим доступу до ресурсу: https://onclinic.ua/blog/digest-comu-vazlivo-privcati-sebe-do-korisnix-zvicok-z-novogo-roku-ta-iak-ce-zrobiti (дата звернення: 21.03.2025).

6. Habitive [Електронний ресурс] – Режим доступу до ресурсу: https://habitive.app/ (дата звернення: 24.03.2025).

7. HabitGenius [Електронний ресурс] – Режим доступу до ресурсу: https://habit-genius.netlify.app/ (дата звернення: 24.03.2025).

8. TheFor: Habit Tracker [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=xyz.thefor.habits.habits&hl=uk (дата

звернення: 24.03.2025).

9. Fun Habit - Habit Tracker [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.habits.todolist.plan.wish&hl (дата звернення: 24.03.2025).

10. everyday | Habit tracker to help you form good habits [Електронний ресурс] – Режим доступу до ресурсу: https://everyday.app/ (дата звернення: 24.03.2025).

11. Журнал звичок (Чек-Лист). Метод Дена Ноултона [Електронний ресурс] – Режим доступу до ресурсу: https://shop.djournal.com.ua/blog/habbit-journal/?srsltid=AfmBOoqxHJeSJINPR5cFr9bNOggv7oZz4Huvax5Q05K0MYM-7-mt_VJZ (дата звернення: 12.04.2025).

12. Трекер корисних звичок. Як визначити пріоритети самовдосконалення [Електронний ресурс] – Режим доступу до ресурсу: https://bogosvyatska.com/ (дата звернення: 12.04.2025).

13. Жолубак, Л. В., Х. В. Мечус та О. О. Смотр Гейміфікація як інструмент підвищення мотивації до навчання. / Одеський національний політехнічний університет, Докторська дисертація, 2020.

14. Коефіцієнт кореляції Пірсона (r-Пірсона) [Електронний ресурс] – Режим доступу до ресурсу: https://www.eztests.xyz/criteria/pearsonr/ (дата звернення: 12.04.2025).

15. Коефіцієнт кореляції Спірмена [Електронний ресурс] – Режим доступу до ресурсу: https://www.eztests.xyz/criteria/spearmanr/ (дата звернення: 12.04.2025).

16. Критерій Стьюдента [Електронний ресурс] – Режим доступу до ресурсу: https://studfile.net/preview/2398476/page:16/ (дата звернення: 12.04.2025).

17. Про Flutter [Електронний ресурс] – Режим доступу до ресурсу: https://devzone.org.ua/post/pro-flutter-korotko-osnovy (дата звернення: 25.04.2025).

18. Kotlin Programming Language [Електронний ресурс] – Режим доступу до ресурсу: https://kotlinlang.org/ (дата звернення: 25.04.2025).

19. Android Studio & App Tools [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio?hl=uk (дата звернення: 07.05.2025).

20. Visual Studio Xamarin [Електронний ресурс] – Режим доступу до ресурсу: https://visualstudio.microsoft.com/ru/xamarin/ (дата звернення: 07.05.2025).

21. Visual Studio Code [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/ (дата звернення: 07.05.2025).

22. Тестування мобільних додатків [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/testuvannya-mobilnih-dodatkiv/ (дата звернення: 07.05.2025).

ДОДАТКИ

Додаток А – Технічне завдання

Додаток А – Технічне завдання 57

Міністерство освіти і науки України
Вінницький національний технічний університет
факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор
_____ О. Н. Романюк
«25» березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка мобільного застосунку для управління корисними звичками» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
_____ к.т.н., доцент О. В. Романюк
«24» березня 2025 р.

Виконав:
_____ студент гр. 1ПІ-21б І. Ю. Гулій
«24» березня 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка мобільного застосунку для управління корисними звичками».

Галузь застосування – цифрова підтримка здоров’я та самоменеджменту користувача.

2. Підстава для розробки

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від «20» березня 2025 р. ректора ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки

Метою дослідження є підвищення ефективності процесу формування корисних звичок на основі застосування статистичних методів кореляційного аналізу та адаптивного трекінгу звичок.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР.

1. Журнал звичок (Чек-Лист). Метод Дена Ноултона [Електронний ресурс] – Режим доступу до ресурсу: https://shop.djournal.com.ua/blog/habbit-journal/?srsltid=AfmBOoqxHJeSJINPR5cFr9bNOggv7oZz4Huvax5Q05K0MYM-7-mt_VJZ (дата звернення: 12.04.2025).

2. Трекер корисних звичок. Як визначити пріоритети самовдосконалення [Електронний ресурс] – Режим доступу до ресурсу: https://bogosvyatska.com/ (дата звернення: 12.04.2025).

5. Технічні вимоги

Вихідні дані до роботи: алгоритм динамічного нагадування, метод кореляції звичок на основі коефіцієнтів Пірсона, Спірмена та статистичних критеріїв (Стьюдента, Фішера); графічний режим – TrueColor; дані про поведінкові патерни – кореляція звичок.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БКР.

2. Технічне завдання.

3. Лістинги програми

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ

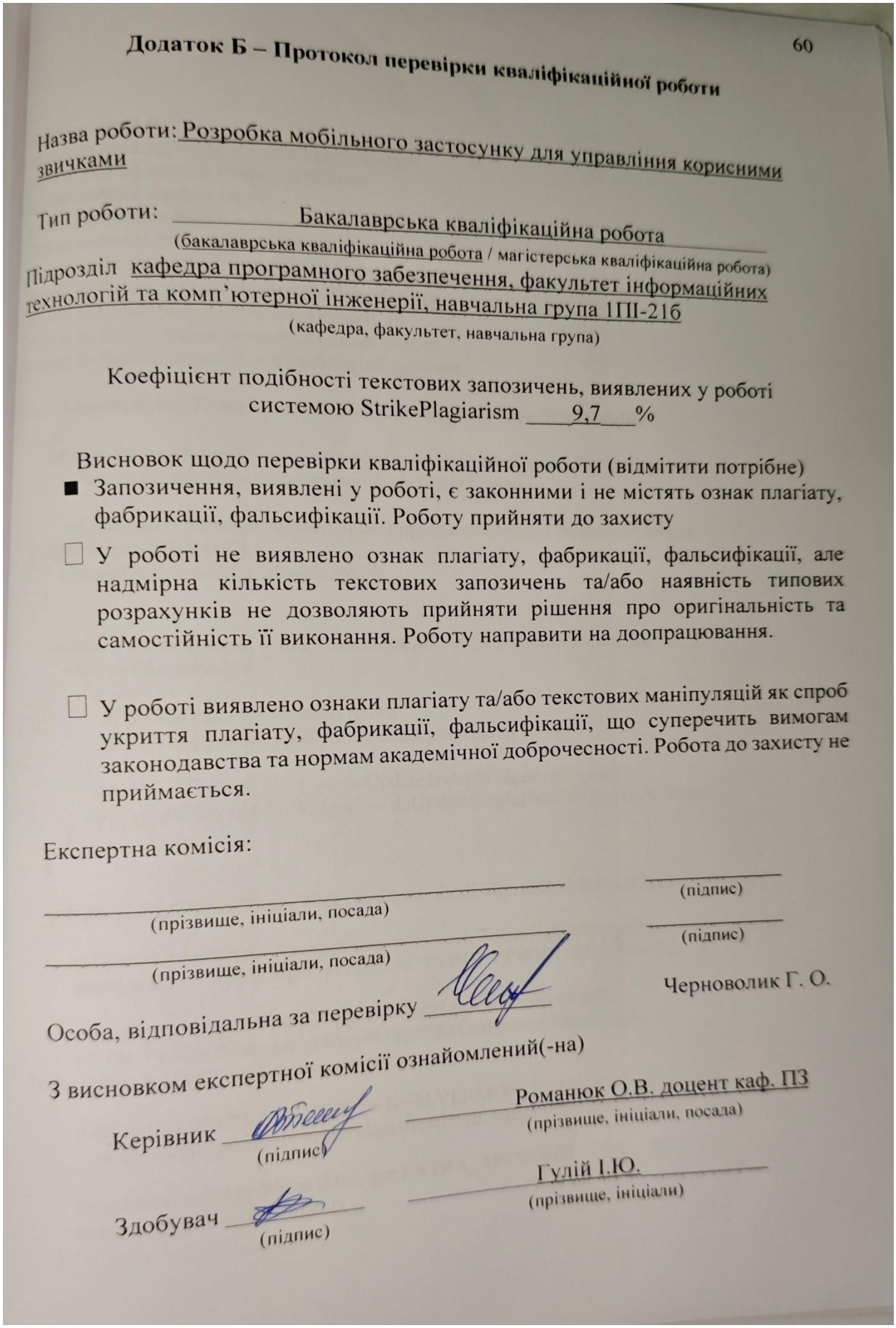
9. Стадії та етапи розробки:

№ з\п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз технологій управління корисними звичками та аналіз аналогів	25.03.25 – 03.04.2025
2	Розробка методу кореляції корисних звичок користувача	03.04.25 – 13.04.2025
3	Розробка методу моніторингу виконання корисних звичок	14.04.25 – 20.04.25
4	Розробка алгоритмів роботи програмного забезпечення	20.04.25 – 03.05.2025
5	Розробка програмного забезпечення	03.05.25 – 23.05.2025
6	Тестування програмного забезпечення	23.05.2025 – 25.05.25
7	Оформлення матеріалів до захисту БКР	25.05.25 – 01.06.25

10. Порядок контролю та прийняття

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б – Протокол перевірки кваліфікаційної роботи



Додаток Б – Протокол перевірки кваліфікаційної роботи 60

Назва роботи: Розробка мобільного застосунку для управління корисними звичками

Тип роботи: Бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, факультет інформаційних технологій та комп'ютерної інженерії, навчальна група 1ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 9,7 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

(прізвище, ініціали, посада) (підпис)

(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник (підпис) Романюк О.В. доцент каф. ПЗ
(прізвище, ініціали, посада)

Здобувач (підпис) Гулій І.Ю.
(прізвище, ініціали)

Додаток В – Лістинг програмного забезпечення

```
import android.R
import android.appwidget.AppWidgetManager
import android.content.ComponentName
import android.content.Intent
import android.os.Build
import androidx.annotation.NonNull
import io.flutter.embedding.android.FlutterActivity
import io.flutter.embedding.engine.FlutterEngine
import io.flutter.plugin.common.MethodCall
import io.flutter.plugin.common.MethodChannel
import io.flutter.plugins.GeneratedPluginRegistrant

class MainActivity: FlutterActivity(), MethodChannel.MethodCallHandler {

    override fun configureFlutterEngine(@NonNull flutterEngine: FlutterEngine) {
        GeneratedPluginRegistrant.registerWith(flutterEngine)

        val channel = MethodChannel(flutterEngine.dartExecutor.binaryMessenger,
WidgetHelper.CHANNEL)
        channel.setMethodCallHandler(this)
    }

    override fun onMethodCall(call: MethodCall, result: MethodChannel.Result) {
        when (call.method) {
            "initialize" -> {
                if (call.arguments == null) return
                WidgetHelper.setHandle(this, call.arguments as Long)
            }
            "doUpdateTodayWidget" -> doUpdateTodayWidget(call, result)
            "doUpdateSingleHabitWidget" -> doUpdateSingleHabitWidget(call, result)
        }
    }

     private fun doUpdateTodayWidget(call: MethodCall, result: MethodChannel.Result) {

        val intent = Intent(context.applicationContext, TodayHabitsWidgetProvider::class.java)
        intent.action = AppWidgetManager.ACTION_APPWIDGET_UPDATE

        val widgetManager = AppWidgetManager.getInstance(context)
        val ids = widgetManager.getAppWidgetIds(ComponentName(context,
TodayHabitsWidgetProvider::class.java))

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB)
            widgetManager.notifyAppWidgetViewDataChanged(ids, R.id.list)

        intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_IDS, ids)
        context.sendBroadcast(intent)
     }
```

```
    private fun doUpdateSingleHabitWidget(call: MethodCall, result: MethodChannel.Result) {

        val intent = Intent(context.applicationContext, SingleHabitWidgetProvider::class.java)
        intent.action = AppWidgetManager.ACTION_APPWIDGET_UPDATE

        val widgetManager = AppWidgetManager.getInstance(context)
        val ids = widgetManager.getAppWidgetIds(ComponentName(context,
SingleHabitWidgetProvider::class.java))

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB)
            widgetManager.notifyAppWidgetViewDataChanged(ids, R.id.list)

        intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_IDS, ids)
        context.sendBroadcast(intent)
    }
}
import android.app.Activity
import android.appwidget.AppWidgetManager
import android.content.Context
import android.content.Intent
import android.content.SharedPreferences
import android.os.Bundle
import android.util.Log
import android.view.View
import android.widget.AdapterView
import android.widget.ArrayAdapter
import android.widget.Spinner
import io.flutter.embedding.engine.FlutterEngine
import io.flutter.embedding.engine.dart.DartExecutor
import io.flutter.plugin.common.MethodChannel
import io.flutter.plugins.GeneratedPluginRegistrant
import io.flutter.view.FlutterCallbackInformation
import io.flutter.view.FlutterMain
import java.util.*

class SingleHabitWidgetConfigureActivity : Activity(), MethodChannel.Result {

    private val TAG = this::class.java.simpleName
    private var isDark:Boolean = false

    private var appWidgetId = AppWidgetManager.INVALID_APPWIDGET_ID
    private lateinit var context: Context

    private var types = arrayOf( "All" )
    private lateinit var adapter: ArrayAdapter<String>

    private var onClickListener = View.OnClickListener {
        val context = this@SingleHabitWidgetConfigureActivity
        initializeFlutter()

        var prefs: SharedPreferences = context.getSharedPreferences("FlutterSharedPreferences",
```

```
Context.MODE_PRIVATE);
        isDark = prefs.getBoolean("flutter.dark-mode", false);

        var fetchType:String = loadTypePref(context, appWidgetId)
        SingleHabitWidgetProvider.channel?.invokeMethod("getSingleHabitAppWidgetData",
"$appWidgetId#$fetchType", this)

        val resultValue = Intent()
        resultValue.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, appWidgetId)
        setResult(RESULT_OK, resultValue)
        finish()
    }

    public override fun onCreate(icicle: Bundle?) {
        super.onCreate(icicle)

        context = applicationContext

        var fetchType = "All";
        SingleHabitWidgetProvider.channel?.invokeMethod("getSingleHabitAppWidgetData",
"$appWidgetId#$fetchType#0", this)

        setResult(RESULT_CANCELED)

        setContentView(R.layout.single_habit_widget_configure)

        val spinner: Spinner = findViewById<View>(R.id.singleHabitSelection) as Spinner

        adapter = ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, types)
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
        spinner.setAdapter(adapter)
        spinner.setOnItemSelectedListener(object : AdapterView.OnItemSelectedListener {
            override fun onItemSelected(p0: AdapterView<*>?, view: View?, position: Int, id: Long) {
                saveTypePref(context, appWidgetId, types[position])
            }

            override fun onNothingSelected(p0: AdapterView<*>?) {
                deleteTypePref(context, appWidgetId)
            }
        })

        findViewById<View>(R.id.add_button).setOnClickListener(onClickListener)

        // Find the widget id from the intent.
        val intent = intent
        val extras = intent.extras
        if (extras != null) {
            appWidgetId = extras.getInt(
                    AppWidgetManager.EXTRA_APPWIDGET_ID,
AppWidgetManager.INVALID_APPWIDGET_ID)
        }
```

```
        // If this activity was started with an intent without an app widget ID, finish with an error.
        if (appWidgetId == AppWidgetManager.INVALID_APPWIDGET_ID) {
            finish()
            return
        }
    }

    private fun initializeFlutter() {
        if (SingleHabitWidgetProvider.channel == null) {
            FlutterMain.startInitialization(context)
            FlutterMain.ensureInitializationComplete(context, arrayOf())

            val handle = WidgetHelper.getRawHandle(context)
            if (handle == WidgetHelper.NO_HANDLE) {
                Log.w(TAG, "Couldn't update widget because there is no handle stored!")
                return
            }

            val callbackInfo = FlutterCallbackInformation.lookupCallbackInformation(handle)
            val entryPointFunctionName = callbackInfo.callbackName

            // Instantiate a FlutterEngine.
            val engine = FlutterEngine(context.applicationContext)
            val entryPoint = DartExecutor.DartEntrypoint(FlutterMain.findAppBundlePath(),
entryPointFunctionName)
            engine.dartExecutor.executeDartEntrypoint(entryPoint)

            GeneratedPluginRegistrant.registerWith(engine)
            SingleHabitWidgetProvider.channel =
MethodChannel(engine.dartExecutor.binaryMessenger, WidgetHelper.CHANNEL)
        }
    }

    override fun success(result: Any?) {
        val args = result as HashMap<*, *>
        val habits = args["habits"] as List<Map<String, *>>
        val type = args["type"] as String
        val isInit = args["init"] as Boolean

        if( type == "All" ) {

            for(habit in habits) {
                val title = habit["title"] as String
                types += title
            }

            adapter = ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, types)
            adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)

            val spinner: Spinner = findViewById<View>(R.id.singleHabitSelection) as Spinner
            spinner.setAdapter(adapter)
```

```
            val selected = loadTypePref(context, appWidgetId)
            val selectionIndex = types.indexOf(selected)
            spinner.setSelection( if(selectionIndex < 0 ) 0 else selectionIndex )

            adapter.notifyDataSetChanged()
            if(!isInit){
                val id = args["id"] as String
                val progress = args["progress"] as Map<String, Int>

                updateSingleHabitWidget(id.toInt(), type, habits, progress, context, isDark)
            }
        } else {
            val id = args["id"] as String
            val progress = args["progress"] as Map<String, Int>
            updateSingleHabitWidget(id.toInt(), type, habits, progress, context, isDark)
        }
    }

    override fun notImplemented() {
        Log.d(TAG, "notImplemented")
    }

    override fun error(errorCode: String?, errorMessage: String?, errorDetails: Any?) {
        Log.d(TAG, "onError $errorCode")
    }
}

private const val PREFS_NAME = "com.example.cleanhabits.singleHabitWidget.type"
private const val PREF_PREFIX_KEY = "singleHabitWidget_"
internal fun saveTypePref(context: Context, appWidgetId: Int, text: String) {
    val prefs = context.getSharedPreferences(PREFS_NAME, 0).edit()
    prefs.putString(PREF_PREFIX_KEY + appWidgetId, text)
    prefs.apply()
}
internal fun loadTypePref(context: Context, appWidgetId: Int): String {
    val prefs = context.getSharedPreferences(PREFS_NAME, 0)
    val type =  prefs.getString(PREF_PREFIX_KEY + appWidgetId, null)
    return type ?: "All"
}

internal fun deleteTypePref(context: Context, appWidgetId: Int) {
    val prefs = context.getSharedPreferences(PREFS_NAME, 0).edit()
    prefs.remove(PREF_PREFIX_KEY + appWidgetId)
    prefs.apply()
}
import android.app.PendingIntent
import android.appwidget.AppWidgetManager
import android.appwidget.AppWidgetProvider
import android.content.Context
import android.content.Intent
import android.content.SharedPreferences
import android.net.Uri
```

```
import android.util.Log
import android.widget.RemoteViews
import io.flutter.embedding.engine.FlutterEngine
import io.flutter.embedding.engine.dart.DartExecutor
import io.flutter.plugin.common.MethodChannel
import io.flutter.plugins.GeneratedPluginRegistrant
import io.flutter.view.FlutterCallbackInformation
import io.flutter.view.FlutterMain
import java.io.Serializable
import java.time.LocalDate
import java.time.format.TextStyle
import java.util.*

class SingleHabitWidgetProvider : AppWidgetProvider(), MethodChannel.Result {

    private val TAG = this::class.java.simpleName
    private var isDark:Boolean = false

    companion object {
        var channel: MethodChannel? = null;
    }

    private lateinit var context: Context

    override fun onUpdate(context: Context, appWidgetManager: AppWidgetManager,
appWidgetIds: IntArray) {
        this.context = context

        initializeFlutter()

        var prefs:SharedPreferences = context.getSharedPreferences("FlutterSharedPreferences",
Context.MODE_PRIVATE);
        isDark = prefs.getBoolean("flutter.dark-mode", false);

        for (appWidgetId in appWidgetIds){
            val fetchType = loadTypePref(context, appWidgetId)
            channel?.invokeMethod("getSingleHabitAppWidgetData", "$appWidgetId#$fetchType",
this)
        }
    }

    private fun initializeFlutter() {
        if (channel == null) {
            FlutterMain.startInitialization(context)
            FlutterMain.ensureInitializationComplete(context, arrayOf())

            val handle = WidgetHelper.getRawHandle(context)
            if (handle == WidgetHelper.NO_HANDLE) {
                Log.w(TAG, "Couldn't update widget because there is no handle stored!")
                return
            }
```

```
            val callbackInfo = FlutterCallbackInformation.lookupCallbackInformation(handle)
            val entryPointFunctionName = callbackInfo.callbackName
           val engine = FlutterEngine(context.applicationContext)
            val entryPoint = DartExecutor.DartEntrypoint(FlutterMain.findAppBundlePath(),
entryPointFunctionName)
            engine.dartExecutor.executeDartEntrypoint(entryPoint)
            GeneratedPluginRegistrant.registerWith(engine)

            channel = MethodChannel(engine.dartExecutor.binaryMessenger,
WidgetHelper.CHANNEL)
        }
    }

    override fun success(result: Any?) {

        val args = result as HashMap<*, *>
        val id = args["id"] as String
        val type = args["type"] as String
        val habits = args["habits"] as List<Map<String, *>>
        val progress = args["progress"] as Map<String, Int>

        updateSingleHabitWidget(id.toInt(), type, habits, progress, context, isDark)
    }

    override fun notImplemented() {
        Log.d(TAG, "notImplemented")
    }

    override fun error(errorCode: String?, errorMessage: String?, errorDetails: Any?) {
        Log.d(TAG, "onError $errorCode")
    }

    override fun onDisabled(context: Context?) {
        super.onDisabled(context)
        channel = null
    }
}

internal fun updateSingleHabitWidget(appWidgetId: Int, type: String, habits: List<Map<String,
*>>, progress: Map<String, Int>, context: Context, isDark: Boolean) {

    var layout = if(isDark) R.layout.single_habit_widget_dark else R.layout.single_habit_widget
    val views = RemoteViews(context.packageName, layout).apply{

        setTextViewText(R.id.singleHabitWidgetHeaderText, type)

        val now = LocalDate.now()
        var month = now.month.getDisplayName(TextStyle.SHORT, Locale.getDefault())
        var year = now.year

        val monthText = "$month, $year"
        setTextViewText(R.id.singleHabitWidgetMonthText, monthText)
```

```
        val intentConfig = Intent(context, SingleHabitWidgetConfigureActivity::class.java)
        intentConfig.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, appWidgetId) //set
widget id
        val pendingIntentConfig = PendingIntent.getActivity(context, 0, intentConfig, 0)
        setOnClickPendingIntent(R.id.singleHabitWidgetHeaderIcon, pendingIntentConfig)

        ////////////////////////////////////////////////////
        val intent = Intent(context, SingleHabitWidgetRemoteViewsService::class.java)
        intent.putExtra("progress", progress as Serializable)
        intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, appWidgetId)
        intent.setData(Uri.parse(intent.toUri(Intent.URI_INTENT_SCHEME)))
        intent.setAction(System.currentTimeMillis().toString())
        setRemoteAdapter(R.id.singleHabitCalenderGridView, intent)
    }

    val manager = AppWidgetManager.getInstance(context)
    manager.updateAppWidget(appWidgetId, views)
}
import android.appwidget.AppWidgetManager
import android.content.Context
import android.content.Intent
import android.content.SharedPreferences
import android.graphics.Color
import android.widget.RemoteViews
import android.widget.RemoteViewsService
import java.time.LocalDate
import java.time.format.DateTimeFormatter

class SingleHabitWidgetRemoteViewsFactory: RemoteViewsService.RemoteViewsFactory  {

    private val TAG = this::class.java.simpleName
    private val _dtFormat = DateTimeFormatter.ofPattern("dd-MM-yyyy")

    private val mContext: Context
    private var appWidgetId = 0

    private var days = listOf("SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT")
    private var progress: Map<LocalDate, Boolean> = mapOf<LocalDate, Boolean>()

    private var isDark:Boolean = false

    constructor(mContext: Context, intent: Intent?) {
        this.mContext = mContext
        appWidgetId = intent?.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
AppWidgetManager.INVALID_APPWIDGET_ID)!!;

        var prefs: SharedPreferences = mContext.getSharedPreferences("FlutterSharedPreferences",
Context.MODE_PRIVATE);
        isDark = prefs.getBoolean("flutter.dark-mode", false);

        var pProgress = intent?.getSerializableExtra("progress") as Map<String, Int>
```

```
        for((strDate, progress) in pProgress) {
            var ldDate = LocalDate.parse(strDate, _dtFormat);
            this.progress += Pair(ldDate, progress > 0);
        }
    }

    override fun onCreate() {
    }

    override fun onDataSetChanged() {
    }

    override fun onDestroy() {
    }

    override fun getCount(): Int {
        return 42
    }

    override fun getViewAt(position: Int): RemoteViews {

        if(position < days.size){
            return RemoteViews(mContext.packageName, R.layout.calender_item).apply{
                setTextViewText(R.id.calenderItemText, days[position])
                setTextColor(R.id.calenderItemText, Color.parseColor( if (isDark ) "#FFDE0000" else
"#FF056676" ))
            }
        } else {

            val now = LocalDate.now()
            val monthStart = now.minusDays((now.dayOfMonth - 1).toLong())
            val dayOfWeek = monthStart.dayOfWeek.value
            val calStart = monthStart.minusDays((dayOfWeek - 0).toLong())
            val dateToShow = calStart.plusDays((position - days.size).toLong())

            var doHighlight = this.progress[dateToShow];
            if ( doHighlight != null && doHighlight ) {
                val layout = if(isDark) R.layout.calender_item_selected_dark else
R.layout.calender_item_selected
                return RemoteViews(mContext.packageName, layout).apply {

                    val textColorHex = if(isDark) "#FF000000" else "#FFFFFFFF"

                    setTextColor(R.id.calenderItemText, Color.parseColor(textColorHex))
                    setTextViewText(R.id.calenderItemText, dateToShow.dayOfMonth.toString())
                }

            } else if ( dateToShow.isEqual(now) ) {
                val layout = if(isDark) R.layout.calender_item_unselected_dark else
R.layout.calender_item_unselected
                return RemoteViews(mContext.packageName, layout).apply {
```

```
                val textColorHex = if(isDark) "#FFFFFF" else "#000000"

                setTextColor(R.id.calenderItemText, Color.parseColor(textColorHex))
                setTextViewText(R.id.calenderItemText, dateToShow.dayOfMonth.toString())
            }

        } else {
            return RemoteViews(mContext.packageName, R.layout.calender_item).apply {

                val currentMonth = now.month.value
                val monthToShow = dateToShow.month.value
                val textColorHex = if (currentMonth == monthToShow) if(isDark) "#FFFFFF" else
"#000000" else  if(isDark) "#60ffd5d5" else "#A39E93"

                setTextColor(R.id.calenderItemText, Color.parseColor(textColorHex))
                setTextViewText(R.id.calenderItemText, dateToShow.dayOfMonth.toString())
            }
        }
    }

    override fun getLoadingView(): RemoteViews {
        val rv = RemoteViews(mContext.packageName, R.layout.widget_loading)
        rv.setTextViewText(R.id.widgetLoadingText, ".")
        return rv
    }

    override fun getViewTypeCount(): Int {
        return 3
    }

    override fun getItemId(position: Int): Long {
        return position.toLong()
    }

    override fun hasStableIds(): Boolean {
      return true
    }
}
import android.app.PendingIntent
import android.appwidget.AppWidgetManager
import android.appwidget.AppWidgetProvider
import android.content.Context
import android.content.Intent
import android.content.SharedPreferences
import android.net.Uri
import android.util.Log
import android.widget.RemoteViews
import io.flutter.embedding.engine.FlutterEngine
import io.flutter.embedding.engine.dart.DartExecutor
import io.flutter.plugin.common.MethodChannel
import io.flutter.plugins.GeneratedPluginRegistrant
```

```
import io.flutter.view.FlutterCallbackInformation
import io.flutter.view.FlutterMain
import java.io.Serializable

class TodayHabitsWidgetProvider : AppWidgetProvider(), MethodChannel.Result {

    private val TAG = this::class.java.simpleName
    private var isDark:Boolean = false

    companion object {
        private var channel: MethodChannel? = null;
    }

    private lateinit var context: Context

    override fun onUpdate(context: Context, appWidgetManager: AppWidgetManager,
appWidgetIds: IntArray) {
        this.context = context

        initializeFlutter()

        var prefs: SharedPreferences = context.getSharedPreferences("FlutterSharedPreferences",
Context.MODE_PRIVATE);
        isDark = prefs.getBoolean("flutter.dark-mode", false);

        for (appWidgetId in appWidgetIds)
            channel?.invokeMethod("getTodayAppWidgetData", appWidgetId, this)

    }

    private fun initializeFlutter() {
        if (channel == null) {
            FlutterMain.startInitialization(context)
            FlutterMain.ensureInitializationComplete(context, arrayOf())

            val handle = WidgetHelper.getRawHandle(context)
            if (handle == WidgetHelper.NO_HANDLE) {
                Log.w(TAG, "Couldn't update widget because there is no handle stored!")
                return
            }

            val callbackInfo = FlutterCallbackInformation.lookupCallbackInformation(handle)
            val entryPointFunctionName = callbackInfo.callbackName

            // Instantiate a FlutterEngine.
            val engine = FlutterEngine(context.applicationContext)
            val entryPoint = DartExecutor.DartEntrypoint(FlutterMain.findAppBundlePath(),
entryPointFunctionName)
            engine.dartExecutor.executeDartEntrypoint(entryPoint)
            GeneratedPluginRegistrant.registerWith(engine)

            channel = MethodChannel(engine.dartExecutor.binaryMessenger,
```

```
WidgetHelper.CHANNEL)
        }
    }

    override fun success(result: Any?) {

        val args = result as HashMap<*, *>
        val id = args["id"] as Int
        val habits = args["habits"] as List<Map<String, *>>
        val progress = args["progress"] as List<Map<String, *>>

        updateTodayWidget(id, habits, progress, context, isDark)
    }

    override fun notImplemented() {
        Log.d(TAG, "notImplemented")
    }

    override fun error(errorCode: String?, errorMessage: String?, errorDetails: Any?) {
        Log.d(TAG, "onError $errorCode")
    }

    override fun onDisabled(context: Context?) {
        super.onDisabled(context)
        channel = null
    }

}

internal fun updateTodayWidget(id: Int, habits: List<Map<String, *>>, progress: List<Map<String,
*>>, context: Context, isDark: Boolean) {

    val layout = if(isDark) R.layout.today_habits_widget_dark else R.layout.today_habits_widget
    val views = RemoteViews(context.packageName, layout).apply {
        val widgetText: CharSequence = context.getString(R.string.todayWidgetHeaderText)
        setTextViewText(R.id.todayWidgetHeaderText, widgetText)

        val intent = Intent(context, TodayWidgetRemoteViewsService::class.java)
        intent.putExtra("habits", habits as Serializable)
        intent.putExtra("progress", progress as Serializable)
        intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, id)
        intent.setData(Uri.parse(intent.toUri(Intent.URI_INTENT_SCHEME)))
        intent.setAction(System.currentTimeMillis().toString())

        setOnClickPendingIntent(
                R.id.todayWidget,
                PendingIntent.getActivity(
                        context,
                        0,
                        Intent(context, MainActivity::class.java),
                        0
                )
```

```
        )

        setRemoteAdapter(R.id.todayWidgetListView, intent)
    }

    val manager = AppWidgetManager.getInstance(context)
    manager.updateAppWidget(id, views)
}

import android.app.PendingIntent
import android.appwidget.AppWidgetManager
import android.content.Context
import android.content.Intent
import android.content.SharedPreferences
import android.widget.AdapterView
import android.widget.RemoteViews
import android.widget.RemoteViewsService

class TodayWidgetRemoteViewsFactory : RemoteViewsService.RemoteViewsFactory {

    private val TAG = this::class.java.simpleName

    private val mContext: Context
    private var appWidgetId = 0

    private var habits: List<Map<String, *>> = emptyList()
    private var progress: List<Map<String, *>> = emptyList()

    private var isDark:Boolean = false

    constructor(mContext: Context, intent: Intent?) {
        this.mContext = mContext
        appWidgetId = intent?.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
AppWidgetManager.INVALID_APPWIDGET_ID)!!;

        var prefs: SharedPreferences = mContext.getSharedPreferences("FlutterSharedPreferences",
Context.MODE_PRIVATE);
        isDark = prefs.getBoolean("flutter.dark-mode", false);

        this.habits = intent?.getSerializableExtra("habits") as List<Map<String, *>>
        this.progress = intent?.getSerializableExtra("progress") as List<Map<String, *>>

    }

    override fun onCreate() {
    }
    override fun onDataSetChanged() {
    }
    override fun onDestroy() {
    }

    override fun getCount(): Int {
```

```
        return habits.size
    }

    override fun getViewAt(position: Int): RemoteViews {
        if (position == AdapterView.INVALID_POSITION) {
            val rv = RemoteViews(mContext.packageName, R.layout.widget_loading)
            rv.setTextViewText(R.id.widgetLoadingText, "#N/A")
            return rv
        }

        val hbData = habits.get(position)
        val progData = progress.get(position)
        val title = hbData["title"] as String
        val isYNType = ( hbData["is_yn_type"] as Int ) == 1
        val timeOfDay = hbData["time_of_day"] as String
        val target = hbData["times_target"] as String
        val targetType = if (isYNType) null else (hbData["times_target_type"] as String)
        val progress = progData["progress"] as String

        var complete = target == progress

        val layout = if(isDark) R.layout.today_widget_listitem else
R.layout.today_widget_listitem_dark
        return RemoteViews(mContext.packageName, layout).apply {
            setTextViewText(R.id.todayWidgetListItemHabitTitle, title)
            setTextViewText(R.id.todayWidgetListItemHabitSubtitle1, timeOfDay)

            var listSubtitle2 = if (isYNType) " " else "$progress/$target $targetType"
            setTextViewText(R.id.todayWidgetListItemHabitSubtitle2, listSubtitle2)

            var listIcon = if (complete) ( if(isDark)R.drawable.ic_baseline_check_24_white else
R.drawable.ic_baseline_check_24 ) else (if(isDark)  R.drawable.ic_baseline_check_24_dark else
R.drawable.ic_baseline_check_24_white)
            setImageViewResource(R.id.todayWidgetListItemHabitProgress, listIcon)

            setOnClickPendingIntent(
                    R.id.todayWidgetListItem,
                    PendingIntent.getActivity(
                            mContext,
                            0,
                            Intent(mContext, MainActivity::class.java),
                            0
                    )
            )
        }
    }

    override fun getLoadingView(): RemoteViews {
        val rv = RemoteViews(mContext.packageName, R.layout.widget_loading)
        rv.setTextViewText(R.id.widgetLoadingText, "... loading ...")
        return rv
    }
```

```
    override fun getViewTypeCount(): Int {
        return 1
    }

    override fun getItemId(position: Int): Long {
        return position.toLong()
    }

    override fun hasStableIds(): Boolean {
        return true
    }

}
import android.content.Context

class WidgetHelper {
    companion object  {
        private const val WIDGET_PREFERENCES_KEY = "widget_preferences"
        private const val WIDGET_HANDLE_KEY = "handle"

        const val CHANNEL = "com.babanomania.cleanhabits/updateWidget"
        const val NO_HANDLE = -1L

        fun setHandle(context: Context, handle: Long) {
            context.getSharedPreferences(
                    WIDGET_PREFERENCES_KEY,
                    Context.MODE_PRIVATE
            ).edit().apply {
                putLong(WIDGET_HANDLE_KEY, handle)
                apply()
            }
        }

        fun getRawHandle(context: Context): Long {
            return context.getSharedPreferences(
                    WIDGET_PREFERENCES_KEY,
                    Context.MODE_PRIVATE
            ).getLong(WIDGET_HANDLE_KEY, NO_HANDLE)
        }
    }
}
```

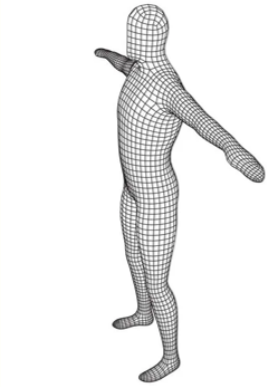
Додаток Г – Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

ДЛЯ УПРАВЛІННЯ КОРИСНИМИ ЗВИЧКАМИ

Вінницький Національний Технічний Університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення



БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка мобільного застосунку для управління корисними звичками»

Виконав:
ст. групи 1ПІ-21б
Гулій І.Ю.

Науковий керівник:
к.т.н., доцент каф. ПЗ
Романюк О.В.

1

Рисунок Г.1 – Титульний аркуш

Актуальність дослідження

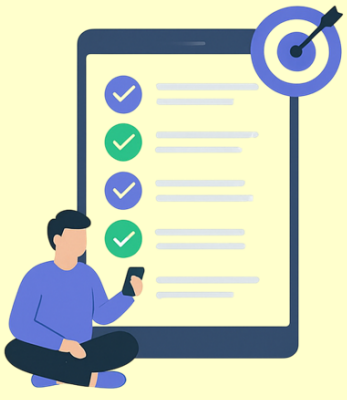
1. Сучасні умови життя вимагають високого рівня самоменеджменту, самодисципліни та підтримки психічного і фізичного здоров'я.

2. Більшість наявних мобільних трекерів звичок обмежені у функціоналі: відсутня аналітика, гнучке нагадування, персоналізація.

3. Недостатня кількість застосунків, які аналізують звички користувача, виявляють їх взаємозв'язки та формують рекомендації.

Запропонований підхід об'єднує інтелектуальний аналіз поведінки, візуалізацію взаємозв'язків та адаптивний механізм нагадувань.

Підсумок:
Розробка інтелектуального мобільного застосунку дозволить підвищити ефективність формування звичок, використовуючи кореляційний аналіз та поведінкове моделювання.



2

Рисунок Г.2 – Актуальність дослідження

Мета дослідження

- **Мета та завдання дослідження.** Метою дослідження є підвищення ефективності процесу формування корисних звичок на основі застосування статистичних методів кореляційного аналізу та адаптивного трекінгу звичок.
- **Об'єкт дослідження** – процес моніторингу та формування корисних звичок.
- **Предмет дослідження** – методи та засоби моніторингу та формування корисних звичок.
- **Методи дослідження.** У процесі дослідження застосовувалися: теоретичні положення математичної статистики; методи теорії графів для моделювання зв'язків між звичками користувача; поведінкова аналітика; комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.
- Відповідно до поставленої мети в бакалаврській кваліфікаційній роботі потрібно вирішити такі **задачі**:

– аналіз предметної галузі;
– розробити алгоритм загальної логіки роботи мобільного застосунку, що включає етапи створення звичок, трекінгу активності, аналітики та динамічного формування рекомендацій;
– розробити метод кореляції корисних звичок, який базується на побудові бінарних векторів виконання звичок користувача, розрахунку необхідних коефіцієнтів для виявлення стійких зв'язків між звичками;
– розробити метод моніторингу виконання корисних звичок, який передбачає оцінку інтенсивності виконання корисних звичок з використанням вагових коефіцієнтів та динамічну модель нагадувань з урахуванням поведінкових залежностей;
– розробка програмного забезпечення для управління корисними звичками;
– тестування розроблених програмних продуктів.

3

Рисунок Г.3 – Мета дослідження

Метод кореляції корисних звичок

Для вимірювання лінійного зв'язку між двома звичками A і B використовується коефіцієнт кореляції Пірсона [14]:

$$r_{AB} = \frac{\sum_{i=1}^{n}(A_i - \bar{A})(B_i - \bar{B})}{\sqrt{\sum_{i=1}^{n}(A_i - \bar{A})^2} * \sqrt{\sum_{i=1}^{n}(B_i - \bar{B})^2}}$$

, де A_i, B_i – значення виконання звичок у день i; $\bar{A}$, $\bar{B}$ – середні значення по кожній звичці.

Якщо між звичками не простежується лінійний зв'язок, але існує монотонна залежність, застосовується коефіцієнт Спірмена [15]:

$$\rho_{AB} = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

, де di = ранг (Ai) – ранг (Bi); n – кількість спостережень.

Щоб перевірити статистичну значущість обчисленого коефіцієнта кореляції, використовується t-критерій Стьюдента [16]:

$$t = \frac{r_{AB}\sqrt{n-2}}{\sqrt{1-{r_{AB}}^2}}$$

, де: r_{AB} – обчислений коефіцієнт кореляції Пірсона, n – кількість днів у вибірці. Результат порівнюється з критичним значенням t_{crit}, яке залежить від рівня значущості (наприклад, $\alpha = 0{,}05$) та кількості ступенів свободи $n - 2$.

4

Рисунок Г.4 – Метод кореляції корисних звичок

Метод моніторингу виконання корисних звичок

На першому етапі формується двійкова матриця виконання звичок. Нехай $H = [h_{ij}]$, де $h_{ij} \in \{0,1\}$ означає факт виконання i-тої звички у j-тий день. Таким чином, кожен рядок H_i представляє собою часовий вектор активності для відповідної звички.

Для визначення інтенсивності виконання звички i за період T, використовується формула середньої частоти:

$$f_i = \frac{1}{T}\sum_{j=1}^{T} h_{ij}$$

Далі, для аналізу взаємозв'язків між звичками, будується кореляційна матриця $C = [c_{ik}]$, де кожен елемент c_{ik} відображає силу статистичного зв'язку між звичками i та k. Кореляційний коефіцієнт Пірсона обчислюється за формулою:

$$c_{ik} = \frac{\sum_{j=1}^{T}(h_{ij} - \bar{h}_i)(h_{kj} - \bar{h}_k)}{\sqrt{\sum_{j=1}^{T}(h_{ij} - \bar{h}_i)^2} * \sqrt{\sum_{j=1}^{T}(h_{kj} - \bar{h}_k)^2}}$$

Для візуалізації взаємозв'язків будується зважений неорієнтований граф $G = (V, E)$, де V – множина звичок, E – множина ребер між ними, а вага ребра $w_{ik} = |c_{ik}|$. Це дозволяє застосовувати кластерний аналіз для виявлення груп звичок, що формують взаємозалежні патерни поведінки. Щоб оцінити стабільність виконання звички в часовому аспекті, вводиться метрика варіації виконання v_i:

$$v_i = \sqrt{\frac{1}{T}\sum_{j=1}^{T}(h_{ij} - \bar{h}_i)^2}$$

Наступним кроком є побудова функції адаптивного нагадування. Нехай $r_i(j)$ – функція ймовірності нагадування для звички i у день j. Її значення залежить від частоти виконання, стабільності та сили кореляції з іншими звичками, що вже були активовані. Математична модель виглядає так, де α, β, γ – вагові коефіцієнти, що налаштовуються емпірично:

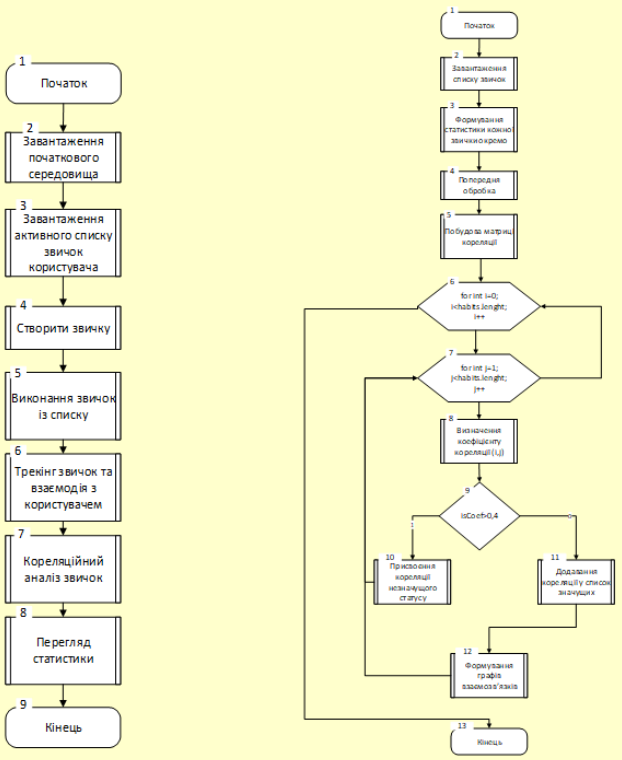
$$r_i(j) = \alpha f_i + \beta(1 - v_i) + \gamma \sum_{k \neq i} c_{ik} * h_{ik}$$

$$t_{ik} = \frac{c_{ik} * \sqrt{T-2}}{\sqrt{1 - {c_{ik}}^2}}$$

5

Рисунок Г.5 – Метод моніторингу корисних звичок

Алгоритми роботи програмного забезпечення



6

Рисунок Г.6 – Алгоритми роботи програмного забезпечення

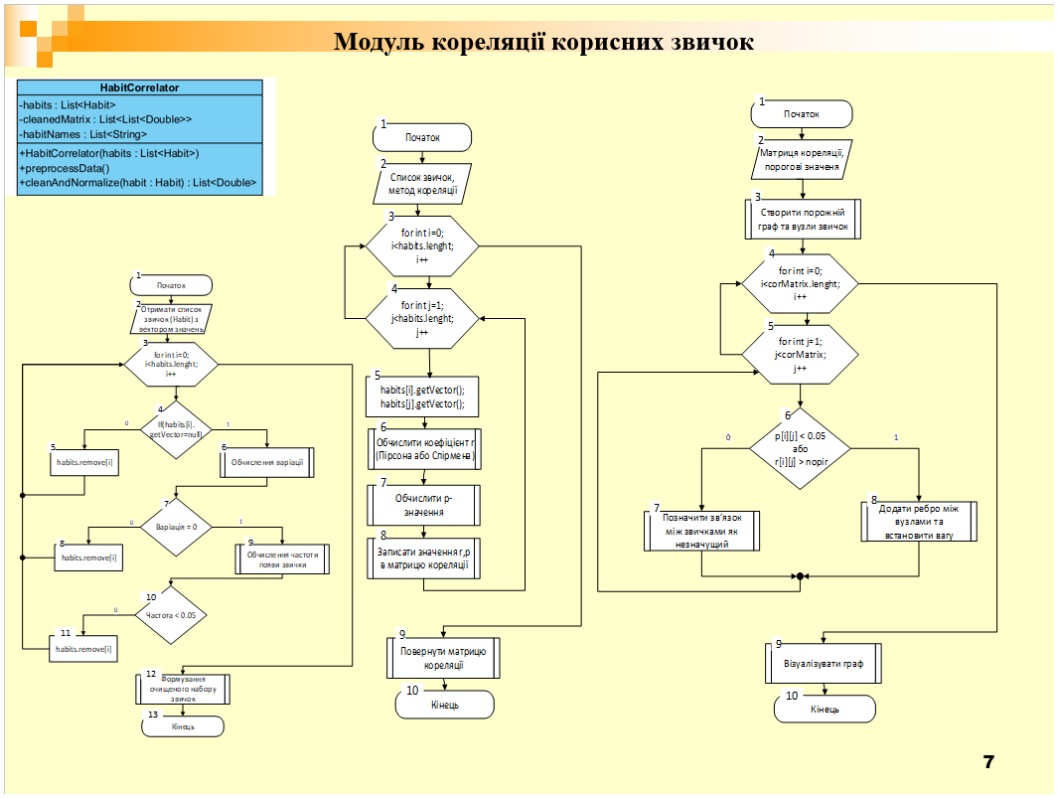


Рисунок Г.7– Модуль кореляції корисних звичок

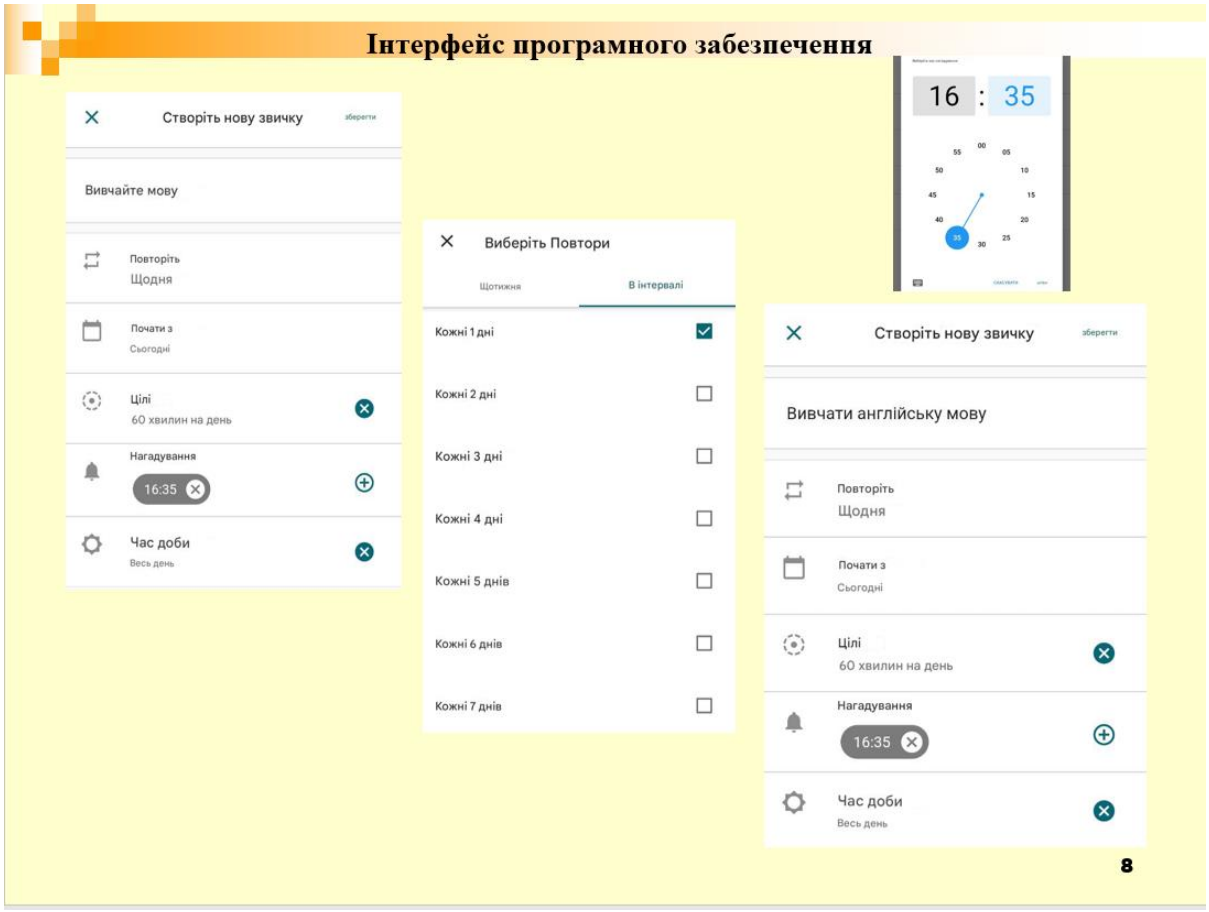


Рисунок Г.8 – Інтерфейс програмного забезпечення

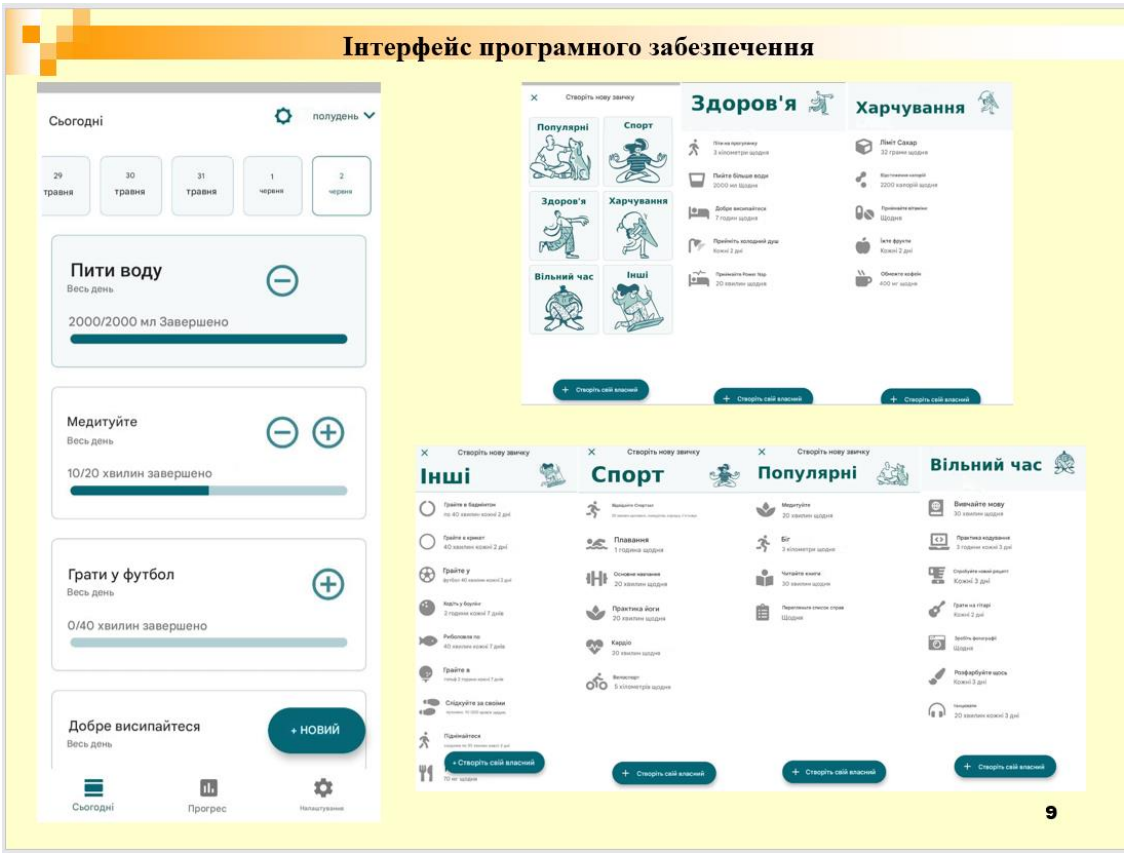


Рисунок Г.9 – Інтерфейс програмного забезпечення

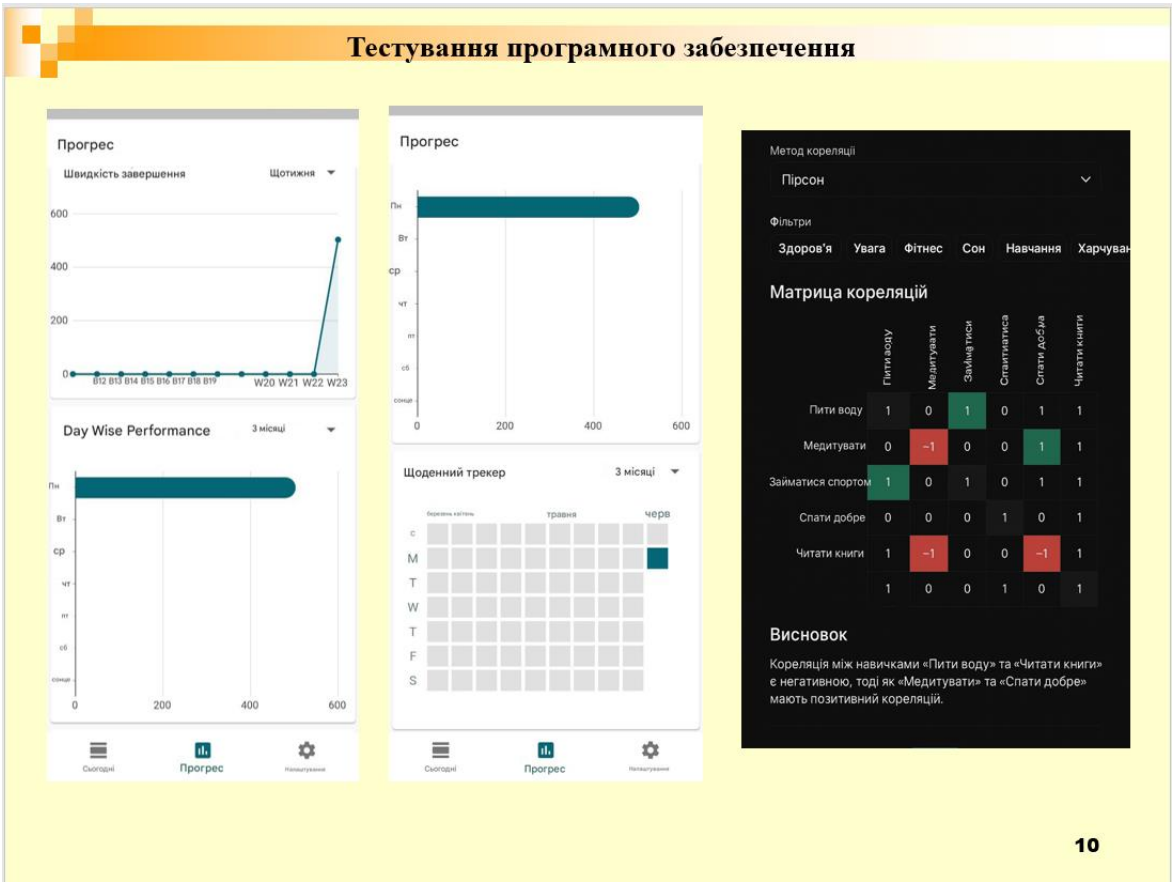


Рисунок Г.10 – Тестування програмного забезпечення

Наукова новизна отриманих результатів

1. 1. Розроблено новий метод кореляції корисних звичок у мобільному застосунку, що базується на побудові бінарних векторів виконання звичок користувача, обчисленні коефіцієнтів кореляції Пірсона і Спірмена, а також перевірці статистичної значущості за t-критерієм Стьюдента, що дозволило виявляти стійкі взаємозв'язки між звичками та формувати на їх основі рекомендації щодо оптимізації поведінкових стратегій.
2. 2. Запропоновано новий метод моніторингу виконання корисних звичок, який передбачає побудову двійкової матриці активності, обчислення інтенсивності виконання звичок, аналіз варіації поведінки у часі та виявлення значущих залежностей між звичками, що дозволило динамічно формувати нагадування з урахуванням виявлених кореляцій.

- **Практичне значення отриманих результатів** полягає в тому, що на основі теоретичних досліджень розроблено алгоритми та програми для автоматизованого аналізу, моніторингу та формування корисних звичок.

- За темою бакалаврської роботи надруковано 1 праць у матеріалах міжнародних конференціях.

11

Рисунок Г.11 – Наукова новизна

Дякую за увагу!

12

Рисунок Г.12 – Закінчення презентації