Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

# Бакалаврська кваліфікаційна робота

на тему: «Інтелектуальна програмна система персоналізованого формування корисних звичок на основі динамічної оцінки користувацького прогресу»

Виконав: студент 4 курсу
групи 5ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Вознюк Я.А.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Чехместрук Р.Ю.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Дудник О.В.
(прізвище та ініціали)

**Допущено до захисту**
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«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
Романюк О.Н.
« 21 » березня 2025 р.

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

Вознюк Яни Андріївни

1. Тема роботи – «Інтелектуальна програмна система персоналізованого формування корисних звичок на основі динамічної оцінки користувацького прогресу»

Керівник роботи: Чехместрук Р.Ю., к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025 р.

3. Вихідні дані до роботи: середовище розробки Android Studio, мова програмування Kotlin, бібліотеки Room, Jetpack Compose, Hilt.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз розвитку технологій системи персоналізованого формування корисних звичок з урахуванням користувацького прогресу; розробка структури та алгоритмів програмного застосунку; варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення; розробка програмних модулів персоналізованого формування корисних звичок з урахуванням користувацького прогресу; тестування програми; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження, наукова новизна, практична

цінність одержаних результатів; порівняльний аналіз аналогів, викори технології; розробка методу динамічної персоналізованої оцінки прог розробка методу інтелектуального аналізу консистентності звичок; розр архітектури застосунку; розробка моделі застосунку; база даних застос тестування застосунку; висновки; апробація результатів роботи та публікації.

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада Консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийня |
| 1-4 | Чехместрук Р.Ю., к.т.н., доцент кафедри ПЗ | 24.03.25 | 01.06.2 |

7. Дата видачі завдання 24 березня 2025 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів бакалаврської кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз розвитку технологій системи персоналізованого формування корисних звичок з урахуванням користувацького прогресу | 25.03.25 - 05.04.25 | вик |
| 2 | Розробка структури та алгоритмів програмного застосунку | 06.04.25 - 18.04.25 | вик |
| 3 | Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення | 19.04.25 - 21.04.25 | вик |
| 4 | Розробка програмних модулів персоналізованого формування корисних звичок з урахуванням користувацького прогресу | 22.04.25 - 17.05.25 | вик |
| 5 | Тестування програми | 18.05.25 - 25.05.25 | вик |
| 6 | Оформлення матеріалів до захисту БКР | 26.05.25 - 30.05.25 | вик |

Студент _____ Вознюк Я.А.
(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи _____ Чехместрук Р.Ю.
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

У бакалаврській кваліфікаційній роботі розроблено програмні модулі мобільної системи персоналізованого формування корисних звичок з урахуванням користувацького прогресу. Робота фокусується на створенні інтуїтивно зрозумілого та адаптивного мобільного додатку, який допомагає користувачам формувати, відстежувати та підтримувати корисні звички шляхом індивідуалізованих рекомендацій та мотиваційних механізмів.

Система розроблена з використанням мови програмування Kotlin та архітектури MVVM (Model-View-ViewModel), що забезпечує чіткий розподіл відповідальності між компонентами та спрощує подальшу модифікацію і тестування. Особлива увага приділяється обробці та аналізу користувацьких даних для динамічного коригування рекомендацій, складності завдань та мотиваційних стратегій відповідно до прогресу користувача.

Програмні модулі, реалізовані в рамках роботи, забезпечують функціональність для відстеження звичок, аналізу прогресу, персоналізованих сповіщень та гейміфікованих елементів взаємодії. Розроблена система може бути застосована для допомоги користувачам у формуванні широкого спектру корисних звичок у сферах здоров'я, продуктивності, навчання та особистісного розвитку.

# ABSTRACT

In the bachelor's qualification work, software modules of a mobile system for personalized habit formation based on user progress have been developed. The work focuses on creating an intuitive and adaptive mobile application that helps users form, track, and maintain beneficial habits through individualized recommendations and motivational mechanisms.

The system is developed using the Kotlin programming language and the MVVM (Model-View-ViewModel) architecture, which ensures a clear separation of concerns between components and facilitates further modification and testing. Special attention is given to the processing and analysis of user data for dynamically adjusting recommendations, task difficulty, and motivational strategies according to the user's progress.

The software modules implemented in this work provide functionality for habit tracking, progress analysis, personalized notifications, and gamified interaction elements. The developed system can be applied to assist users in forming a wide range of beneficial habits in areas such as health, productivity, learning, and personal development.

ЗМІСТ

# ВСТУП

**Обґрунтування вибору теми дослідження**. У сучасному світі формування та підтримка корисних звичок стає все більш складним завданням через постійне зростання інформаційного навантаження, стрес та швидкий ритм життя. Сучасна людина щодня стикається з безліччю відволікаючих факторів, які заважають концентрації на важливих цілях та послідовному виконанні корисних дій. Традиційні підходи до вироблення звичок, які базуються на статичних рекомендаціях та одноманітних методиках, часто виявляються неефективними через індивідуальні особливості кожної людини, різний рівень мотивації та мінливість життєвих обставин. Дослідження показують, що успішність формування звичок залежить від численних факторів, включаючи тип особистості, поточний емоційний стан, соціальне оточення та навіть час доби [1].

Існуючі мобільні додатки та веб-платформи для трекінгу звичок, хоча і надають базовий функціонал відстеження прогресу, мають недостатній функціонал для персоналізації підходу до кожного користувача [2]. Більшість таких систем використовує спрощені алгоритми нагадувань та статистики, не враховуючи психологічні особливості формування звичок, динаміку мотивації користувача або контекстуальні фактори, що впливають на успішність виконання запланованих дій [3]. Крім того, такі системи зазвичай не адаптуються до змін у поведінці користувача, не аналізують причини невдач та не пропонують альтернативних стратегій досягнення цілей. Це призводить до того, що користувачі часто втрачають мотивацію та припиняють використання додатків через кілька тижнів після початку.

Сучасні дослідження у галузі поведінкової психології та машинного навчання демонструють потенціал створення адаптивних систем, здатних аналізувати паттерни поведінки користувача, прогнозувати ймовірність виконання завдань та динамічно коригувати стратегії мотивації [4]. Інтеграція методів штучного інтелекту з принципами формування звичок може забезпечити значно вищу ефективність порівняно з існуючими рішеннями. Зокрема, використання

алгоритмів машинного навчання дозволяє виявляти складні залежності між різними факторами, що впливають на поведінку користувача, та створювати персоналізовані рекомендації у реальному часі.

Динамічна оцінка прогресу користувача передбачає не лише фіксацію факту виконання або невиконання завдань, але й аналіз якості виконання, емоційного стану користувача, зовнішніх обставин та інших контекстуальних даних. Така комплексна оцінка дозволяє системі краще розуміти індивідуальні потреби користувача та адаптувати свою роботу відповідно до його поточного стану і цілей[5]. Водночас необхідним є забезпечення приватності та безпеки персональних даних користувачів, що вимагає розробки спеціальних підходів до збору, обробки та зберігання інформації [6].

Враховуючи недоліки наявних систем, потенціал сучасних технологій для створення по-справжньому персоналізованого досвіду формування звичок, а також зростаючу потребу людей у ефективних інструментах самовдосконалення, актуальною є розробка власної системи.

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

**Мета та завдання дослідження.** Метою бакалаврської кваліфікаційної роботи є підвищення ефективності процесу формування корисних звичок за рахунок використання спеціалізованої мобільної системи, яка дозволяє адаптуватися до індивідуального прогресу користувача.

**Основними задачами дослідження є:**

- розробити модуль управління звичками, що забезпечуватиме створення, редагування, видалення та категоризацію користувацьких звичок з можливістю встановлення періодичності, нагадувань та цілей;

- розробити модуль відстеження прогресу, який забезпечить збір даних про виконання звичок, візуалізацію прогресу у вигляді графіків та діаграм, а також аналіз успішності формування звичок;

- розробити модуль аналізу користувацьких даних, що дозволятиме

проводити глибокий аналіз поведінкових патернів, визначати оптимальний час для виконання звичок та адаптувати систему до індивідуальних особливостей користувача;

- розробити модуль персоналізованих сповіщень, який забезпечить своєчасне нагадування про заплановані звички з урахуванням режиму дня користувача та аналізу найбільш ефективного часу взаємодії;

- розробити модуль статистики та звітності, який забезпечить генерацію детальних звітів про прогрес формування звичок, виявлення патернів поведінки та надання персоналізованих рекомендацій;

- розробити модуль адаптивного користувацького інтерфейсу з використанням Jetpack Compose, який забезпечить зручне відображення контенту та інтуїтивно зрозумілу навігацію;

- провести тестування розроблених модулів на відповідність функціональним вимогам, продуктивність та користувацький досвід.

**Об'єктом дослідження** є процес розробки мобільної системи персоналізованого формування корисних звичок.

**Предметом дослідження** є методи та засоби розробки мобільної системи персоналізованого формування корисних звичок з використанням мови програмування Kotlin та архітектури MVVM.

**Методи дослідження.** У процесі досліджень використовувались:

- методи розробки застосунків для формування корисних звичок;
- методи проєктування мобільних застосунків;
- методи оцінки прогресу набуття звички;
- методи аналізу постійності звичок;
- методи тестування застосунків.

**Наукова новизна отриманих результатів.**

1. Подальшого розвитку отримав метод динамічної персоналізованої оцінки прогресу, який, на відміну від відомих, аналізує індивідуальні патерни поведінки користувача для генерації адаптивних рекомендацій на основі реальних даних використання, враховуючи оптимальний час виконання, фактори що заважають

регулярності, та взаємозв'язки між різними звичками, що забезпечує підвищення ефективності формування корисних звичок завдяки точному налаштуванню під особливості кожного користувача.

2.Подальшого розвитку отримав метод інтелектуального аналізу консистентності звичок, який, на відміну від відомих, застосовує комплексну модель для оцінки стабільності звички, враховуючи не лише факт виконання, але й регулярність інтервалів, нещодавні тенденції та довгострокову стабільність відповідно до запланованої частоти, що дозволяє користувачам отримувати об'єктивну оцінку прогресу у формуванні стійких звичок.

**Практична цінність отриманих результатів**. Практична цінність одержаних результатів полягає у можливості використання створеної мобільної системи, що покриватиме різні аспекти формування корисних звичок з урахуванням індивідуального прогресу користувача.

**Особистий внесок здобувача.** Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто.

**Апробація результатів роботи**. Результати бакалаврської кваліфікаційної роботи доповідались на конференції «Modern Scientific Research: Theoretical and Practical Aspects» у 2025 році.

**Публікації**. Результати роботи були опубліковані в матеріалах конференцій: «Modern Scientific Research: Theoretical and Practical Aspects» [7].

**Аналіз**. У пояснювальній записці до бакалаврської кваліфікаційної роботи було розглянуто 4 розділи та було використано 26 літературних джерел.

# 1 АНАЛІЗ РОЗВИТКУ ТЕХНОЛОГІЙ СИСТЕМИ ПЕРСОНАЛІЗОВАНОГО ФОРМУВАННЯ КОРИСНИХ ЗВИЧОК З УРАХУВАННЯМ КОРИСТУВАЦЬКОГО ПРОГРЕСУ

## 1.1 Аналіз стану технологій системи персоналізованого формування корисних звичок з урахуванням користувацького прогресу

Ринок мобільних додатків для формування корисних звичок демонструє стабільне зростання, досягнувши у 2025 році обсягу понад 6 мільярдів доларів. Додатки по формуванню звичок та особистісному розвитку становлять близько 25% загального обсягу ринку програм для здоров'я та благополуччя, що свідчить про фундаментальні зміни в підходах до самовдосконалення та розвитку особистості. За прогнозами аналітиків, цей тренд посилюватиметься, і до 2028 року обсяг ринку може збільшитися вдвічі.

Така трансформація пов'язана із загальним зростанням уваги до ментального здоров'я, поширенням практик усвідомленості, розширенням можливостей мобільних технологій та зміною поведінкових патернів нового покоління користувачів, які прагнуть до постійного самовдосконалення за допомогою цифрових інструментів.

Сучасні системи формування корисних звичок базуються на складних архітектурних та алгоритмічних рішеннях. Багаторівнева архітектура забезпечує гнучкість, масштабованість та адаптивність платформи через розподіл функціональності на взаємопов'язані модулі. Технології машинного навчання, такі як системи рекомендацій та предиктивна аналітика, дозволяють ефективно адаптувати програми формування звичок під індивідуальні потреби користувачів. Хмарна інфраструктура, побудована на базі Firebase, AWS, або Google Cloud, забезпечує зберігання та синхронізацію користувацьких даних, а також надає інструменти для аналізу та візуалізації прогресу [8].

На стороні клієнта домінують сучасні мобільні технології, такі як Kotlin для Android та Swift для iOS, що дозволяють створювати інтуїтивно зрозумілі та високопродуктивні інтерфейси користувача. Архітектурні патерни MVVM, Clean

Architecture та Jetpack Compose забезпечують чітке розділення відповідальності та спрощують підтримку і масштабування додатків. Технології локального зберігання даних, включаючи Room Database, CoreData та SQLite, забезпечують стабільну роботу додатків навіть без підключення до інтернету. Android та iOS API для відстеження активності, сповіщень та інтеграції з носимими пристроями розширюють можливості збору даних та взаємодії з користувачем [9].

Аналіз сучасного стану технологій персоналізованого формування корисних звичок виявляє суттєві можливості для інновацій та вдосконалення. Розробка власної системи є обґрунтованою з огляду на кілька ключових факторів.

Технологічна трансформація галузі створює унікальне вікно можливостей для впровадження інноваційних рішень. Перехід до персоналізованих, адаптивних моделей формування звичок відкриває простір для систем, які краще відповідають індивідуальним особливостям користувачів. Існуючі технологічні обмеження можуть бути подолані за допомогою сучасних інструментів, таких як Kotlin та Android Jetpack для клієнтської частини та Firebase або власних серверних рішень для забезпечення аналітики та персоналізації.

Зміни в поведінкових патернах сучасних користувачів також підтверджують необхідність розробки нових рішень. Сучасні користувачі мають підвищені очікування щодо інтерфейсів, соціальних функцій та моделей взаємодії, які часто не повністю задовольняються існуючими додатками. Власна система може бути спроектована з урахуванням цих змін, забезпечуючи більш персоналізований та адаптивний користувацький досвід [10].

Сучасні мобільні технології відкривають значні можливості для технологічних інновацій. Kotlin та Jetpack Compose дозволяють створювати продуктивні, адаптивні та зручні інтерфейси, які можуть забезпечити конкурентну перевагу на ринку. Ці технології дозволяють швидко впроваджувати нові функції та експериментувати з різними підходами до аналізу користувацького прогресу.

Отже, аналіз сучасного стану технологій систем персоналізованого формування корисних звичок свідчить про динамічний розвиток цієї галузі та наявність суттєвих можливостей для інновацій. Технологічні тренди, включаючи

персоналізовані рекомендації на основі штучного інтелекту, інтеграцію з носимими пристроями, гейміфікацію та соціальні елементи, формують нове покоління додатків для формування звичок.

Водночас існують значні технологічні виклики, пов'язані з обробкою та аналізом користувацьких даних, забезпеченням приватності, адаптацією до індивідуальних особливостей та підтримкою мотивації користувачів у довгостроковій перспективі. Ці виклики створюють можливості для розробки нових, більш ефективних рішень.

### 1.2 Порівняльний аналіз аналогів

Fabulous [11] – це мобільний додаток, розроблений на основі досліджень у галузі поведінкової психології, який спрямований на допомогу користувачам у формуванні здорових звичок та покращенні загального способу життя. Додаток використовує науково обґрунтовані підходи для створення персоналізованих програм, що допомагають адаптувати щоденну рутину під індивідуальні потреби користувача.

Основною особливістю Fabulous є його здатність надавати користувачам спеціально підібрані завдання та виклики, які допомагають встановлювати позитивні звички – від поліпшення режиму сну до регулярних фізичних вправ. Додаток не лише пропонує завдання, а й пояснює користувачу наукові основи, чому саме ці дії сприяють поліпшенню здоров’я та благополуччя.

Система активно відстежує прогрес користувача, аналізуючи як суб’єктивні звіти, так і об’єктивні дані, отримані з інтегрованих пристроїв, таких як фітнес-трекери. Це дозволяє Fabulous адаптувати рекомендації в режимі реального часу, враховуючи досягнення користувача та його змінні потреби, що сприяє більш ефективному впровадженню нових звичок.

Завдяки інтерфейсу та елементам гейміфікації, Fabulous мотивує користувачів продовжувати працювати над собою навіть у складні моменти. Система також надає можливість отримання нагород та візуального відображення прогресу, що додатково стимулює бажання розвиватися і підтримувати досягнуті

результати.

Інтерфейс Fabulous наведено на рисунку 1.1.

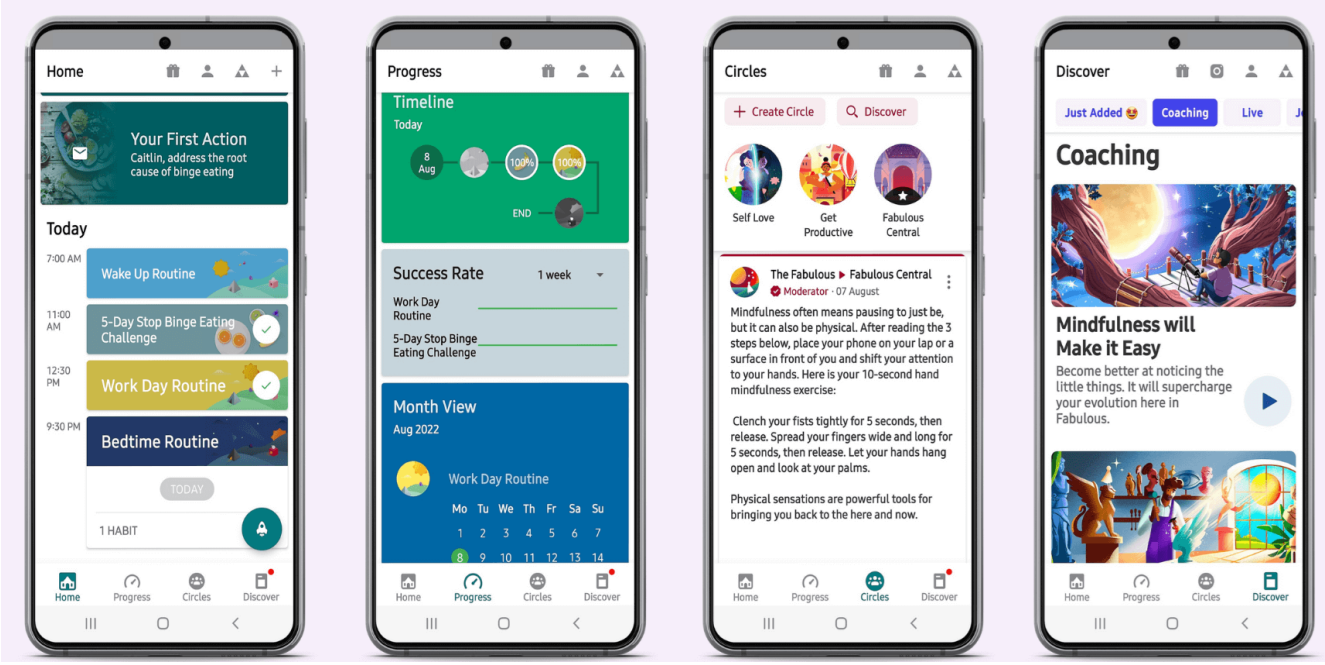


Рисунок 1.1 — Інтерфейс Fabulous

Habitica [12] – це система, що поєднує відстеження щоденних завдань із принципами гейміфікації. Додаток перетворює процес формування звичок на захоплюючу гру, де кожен користувач створює свій віртуальний аватар, який розвивається за рахунок виконання реальних завдань.

Гейміфікація у Habitica полягає в тому, що користувачі отримують бали, рівні та віртуальні нагороди за успішне виконання завдань, що мотивує їх продовжувати працювати над своїми звичками. Завдяки цьому підходу навіть рутинні завдання набувають елементу розваги, а змагання з іншими користувачами стимулюють додаткову активність та відповідальність.

Система динамічно оцінює прогрес користувача через інтегровану систему відстеження виконання завдань. Кожна виконана дія дає можливість підвищити рівень персонажа, що створює відчуття постійного розвитку. Така модель дозволяє користувачам бачити свої досягнення у візуально привабливій формі, що є додатковим стимулом для підтримання мотивації.

Крім індивідуальної роботи над звичками, Habitica створює спільноту користувачів, де можна брати участь у групових викликах і спільних завданнях. Соціальна взаємодія, обмін досвідом та підтримка однодумців роблять систему не тільки інструментом для відстеження прогресу, а й платформою для мотиваційного обміну, що сприяє більш стійким змінам у поведінці.

Інтерфейс Habitica наведено на рисунку 1.2.

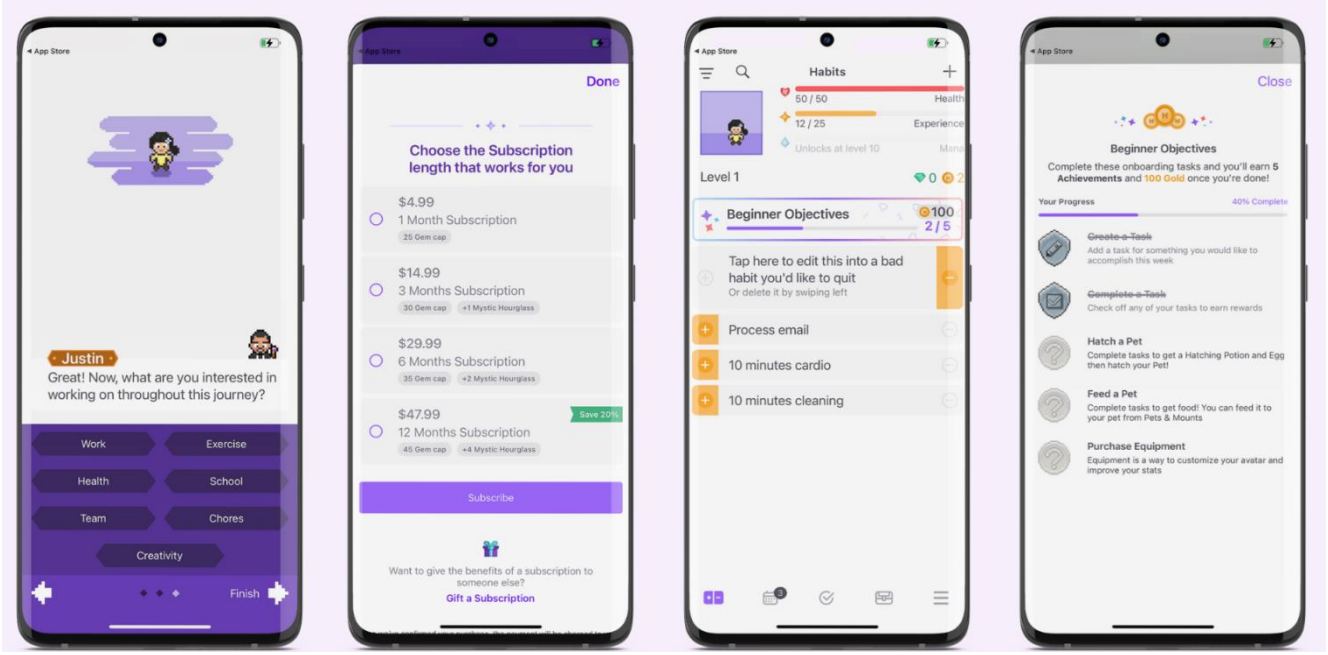


Рисунок 1.2 — Інтерфейс Habitica

Coach.me [13] – це платформа, яка поєднує цифровий трекінг звичок із персональним коучингом, надаючи користувачам можливість отримувати індивідуальні рекомендації від як автоматизованої системи, так і від живих коучів. Такий підхід дозволяє більш гнучко підходити до формування нових звичок, враховуючи особливості кожного користувача.

Платформа дозволяє користувачам вести детальний облік щоденних завдань і досягнень, що допомагає аналізувати власний прогрес. За допомогою вбудованих інструментів Coach.me можна встановлювати цілі, отримувати регулярні нагадування та аналізувати статистику виконання, що дає змогу чітко бачити, де саме потрібне коригування стратегії.

Однією з ключових особливостей Coach.me є динамічна система оцінки прогресу, яка адаптує рекомендації на основі змін у поведінці користувача. Система аналізує дані про виконання завдань у режимі реального часу, що дозволяє своєчасно виявляти відхилення від запланованої траєкторії і пропонувати відповідні коригувальні дії для підтримки мотивації та ефективності.

Крім цифрової частини, Coach.me активно розвиває спільноту, де користувачі можуть обмінюватися досвідом та отримувати підтримку від професійних коучів. Така інтеграція особистісного контакту з автоматизованими рекомендаціями створює комплексний підхід до формування звичок, що дозволяє кожному користувачу знаходити оптимальний шлях до досягнення своїх цілей та покращення якості життя.

Інтерфейс Coach.me наведено на рисунку 1.3.

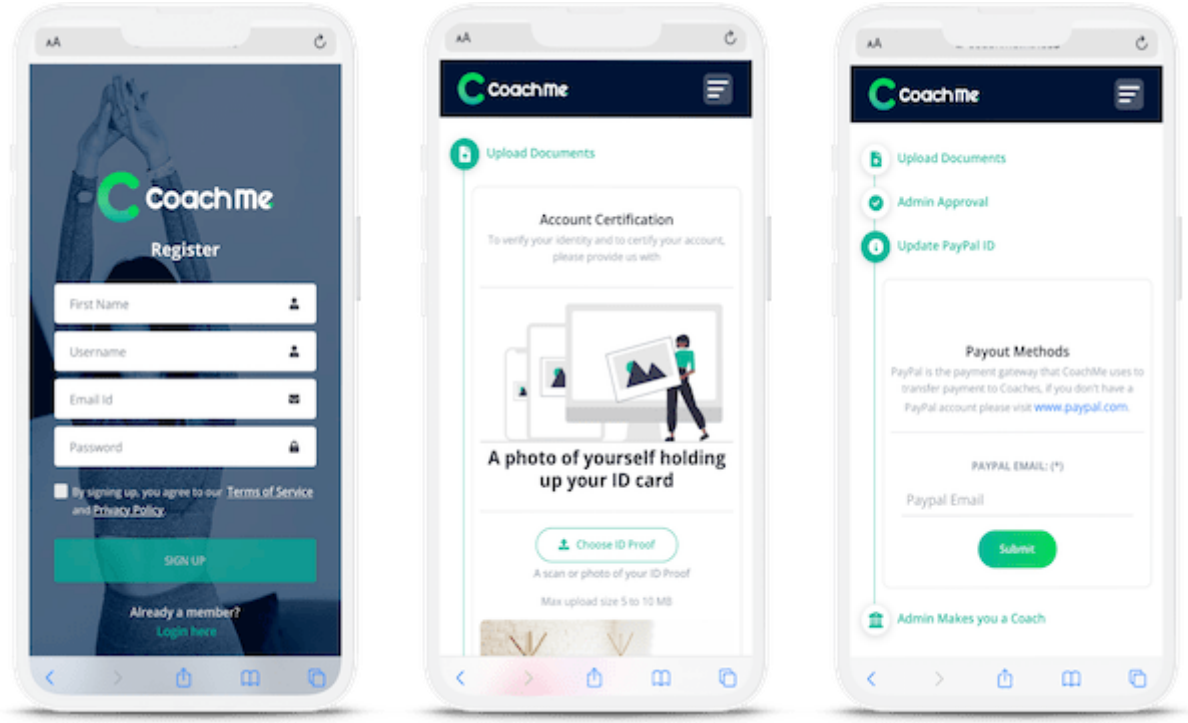


Рисунок 1.3 — Інтерфейс Coach.me

Для порівняння функціональних характеристик розглянутих застосунків, а також їх порівняння з власною розробкою, було сформовано таблицю порівняння аналогів (таблиця 1.1).

Таблиця 1.1 — Порівняльний аналіз аналогів

| Функціональна характеристика | Fabulous | Habitica | Coach.me | Власна розробка |
|---|---|---|---|---|
| Трекер звичок | + | + | + | + |
| Статистика та графіки | + | + | + | + |
| Підрахунок серій виконання звички | + | + | + | + |
| Нагадування | + | + | + | + |
| Щоденні поради | + | - | + | + |
| Персоналізована оцінка прогресу | - | - | - | + |
| Перевірка закріплення звички | - | - | - | + |
| Сумарний бал | 5 | 4 | 5 | 7 |

Таким чином, актуальною є власна розробка, оскільки вона має вищий сумарний бал за Fabulous та Coach.me на 28,5% (100% - 5/7*100% = 28,5%), за Habitica на 42,8% (100% - 4/7*100% = 42,8%).

### 1.3 Аналіз методів розв’язання задачі

Розробка програмних модулів мобільної системи персоналізованого формування корисних звичок з урахуванням користувацького прогресу вимагає детального аналізу існуючих підходів з урахуванням специфіки галузі, психологічних аспектів формування звичок, вимог до адаптивності системи та особливостей взаємодії користувачів з мобільними додатками. Ключовим аспектом при проектуванні такої системи є вибір оптимальної архітектури та методів аналізу користувацького прогресу, які забезпечать персоналізований досвід, високу мотивацію користувачів та ефективне формування стійких звичок.

В контексті розробки мобільної системи персоналізованого формування звичок із використанням Kotlin, можна розглянути декілька архітектурних підходів. Одним із найпоширеніших є Model-View-ViewModel (MVVM) [14], який забезпечує чітке розділення бізнес-логіки від представлення даних. Ця архітектура особливо ефективна при роботі з Android Jetpack компонентами, такими як

LiveData та ViewModels, що дозволяють реагувати на зміни даних та зберігати стан додатку під час змін конфігурації пристрою. Kotlin, як основна мова програмування для Android, повністю підтримує цей підхід, забезпечуючи компактний синтаксис, null-безпеку та функціональні можливості для ефективної обробки даних [15].

Альтернативним підходом є використання Clean Architecture, яка поділяє додаток на шари з чіткими залежностями: домен (бізнес-логіка), дані (репозиторії та джерела даних) та представлення (UI). Цей підхід забезпечує високу тестованість, гнучкість при зміні технологій та чітке розділення відповідальності. В контексті системи формування звичок це особливо важливо для реалізації таких функцій як аналіз прогресу, персоналізовані рекомендації, адаптація складності завдань та інтеграція з різними джерелами даних.

При розробці мобільного додатку з урахуванням користувацького прогресу постає питання ефективного збору та аналізу даних. Для цього можна застосувати патерн «Спостерігач» (Observer), який дозволяє компонентам системи реагувати на зміни в даних користувача. Android Architecture Components, зокрема LiveData, Flow та StateFlow, забезпечують реактивний підхід до обробки даних, що є критично важливим для систем, які мають адаптуватися до змін у поведінці користувача в реальному часі [16].

З точки зору персоналізації та аналізу прогресу користувача, доцільно застосувати алгоритми машинного навчання для класифікації типів користувачів, прогнозування їхньої поведінки та адаптації рекомендацій. Для мобільних додатків можна використовувати локальні моделі TensorFlow Lite або ML Kit, які дозволяють проводити аналіз даних безпосередньо на пристрої, забезпечуючи приватність користувацької інформації та знижуючи залежність від серверної інфраструктури.

Для організації зберігання даних найбільш ефективним є використання Room Database у поєднанні з репозиторіями, що забезпечує абстракцію від конкретного джерела даних та спрощує міграцію між локальним та хмарним зберіганням. Синхронізація даних з хмарними сервісами може бути реалізована

через WorkManager, який забезпечує відкладену та надійну обробку операцій навіть за відсутності підключення до мережі.

Враховуючи поведінкові аспекти формування звичок, особливої уваги потребує система сповіщень та нагадувань. Використання адаптивних алгоритмів для визначення оптимального часу сповіщень на основі історії взаємодії користувача з додатком може суттєво підвищити ефективність формування звичок. Комбінування цих алгоритмів з методами гейміфікації, такими як система досягнень, рівнів та соціальних порівнянь, створює інструмент для підтримки мотивації користувачів.

На основі проведеного аналізу для розробки мобільної системи персоналізованого формування корисних звичок рекомендується використовувати архітектуру MVVM у поєднанні з принципами Clean Architecture та патерном Repository. Додаток буде структуровано за функціональними модулями з чітким розмежуванням відповідальності та використанням Kotlin Coroutines для асинхронних операцій. Android Jetpack компоненти забезпечать надійну роботу з життєвим циклом додатку та реактивне оновлення даних, а Jetpack Compose дозволить створити гнучкий та сучасний користувацький інтерфейс. Така архітектура забезпечить баланс між швидкістю розробки, підтримуваністю коду та можливістю впровадження складних алгоритмів персоналізації та аналізу прогресу користувача.

### 1.4 Постановка задач дослідження

Проаналізувавши існуючі рішення у сфері мобільних систем формування корисних звичок та враховуючи особливості персоналізованого підходу з використанням Kotlin та архітектури MVVM, було визначено наступні задачі, які необхідно виконати в рамках бакалаврської кваліфікаційної роботи:

- розробити модуль управління звичками, що забезпечуватиме створення, редагування, видалення та категоризацію користувацьких звичок з можливістю встановлення періодичності, нагадувань та цілей;

- розробити модуль відстеження прогресу, який забезпечить збір даних про

виконання звичок, візуалізацію прогресу у вигляді графіків та діаграм, а також аналіз успішності формування звичок;

- розробити модуль аналізу користувацьких даних, що дозволятиме проводити глибокий аналіз поведінкових патернів, визначати оптимальний час для виконання звичок та адаптувати систему до індивідуальних особливостей користувача;

- розробити модуль персоналізованих сповіщень, який забезпечить своєчасне нагадування про заплановані звички з урахуванням режиму дня користувача та аналізу найбільш ефективного часу взаємодії;

- розробити модуль статистики та звітності, який забезпечить генерацію детальних звітів про прогрес формування звичок, виявлення патернів поведінки та надання персоналізованих рекомендацій;

- розробити модуль адаптивного користувацького інтерфейсу з використанням Jetpack Compose, який забезпечить зручне відображення контенту та інтуїтивно зрозумілу навігацію;

- провести тестування розроблених модулів на відповідність функціональним вимогам, продуктивність та користувацький досвід.

### 1.5 Висновки

У першому розділі було здійснено комплексний аналіз сучасного стану систем формування звичок, детально розглянуто теоретичні та практичні аспекти цього процесу та систематизовано основні проблеми, з якими стикаються користувачі таких систем. Також проведено ґрунтовне порівняльне дослідження запропонованого рішення з існуючими на ринку платформами, виявлено суттєві конкурентні переваги та інноваційні особливості розроблюваної системи. На основі аналізу чітко сформульовано стратегічні та тактичні завдання, що визначають напрямки подальшої роботи.

# 2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ ПРОГРАМНОГО ЗАСТОСУНКУ

## 2.1 Аналіз вхідних даних системи

Аналіз вхідних даних для програмної системи персоналізованого формування корисних звичок з урахуванням користувацького прогресу потребує визначення основних інформаційних потоків, які формуються в процесі створення, відстеження та аналізу звичок користувача. Дані, що оброблятимуться системою, характеризуються різнорідністю, динамічною природою та високими вимогами до персоналізації, що зумовлює необхідність комплексного підходу до їх організації, зберігання та аналізу.

Для технічної реалізації системи обрано Kotlin у поєднанні з архітектурою MVVM та компонентами Android Jetpack для створення адаптивного та продуктивного мобільного додатку. Архітектурна організація системи містить багаторівневу структуру з чітким розмежуванням відповідальності між компонентами та модульною організацією кодової бази. Такий підхід надасть гнучкості процесу розробки через можливість паралельної роботи над різними модулями та спрощуватиме подальшу підтримку системи через розділення і групування функціональності в окремих частинах системи [17].

Процес розробки мобільної системи персоналізованого формування корисних звичок включає створення п'яти основних функціональних модулів: управління звичками, відстеження прогресу, аналіз користувацьких даних, персоналізовані сповіщення та гейміфікація. Кожен із цих модулів оперує специфічними наборами даних, що потребують відповідних підходів до обробки та візуалізації.

Модуль управління звичками надаватиме функціональність для роботи з інформацією про користувацькі звички, що охоплює параметричні характеристики звичок (назва, опис, категорія, періодичність, час виконання), історію виконання та метрики успішності. Основним призначенням цього модуля є ефективне створення, редагування та відстеження звичок. Дані про звички зберігатимуться

локально в Room Database та синхронізуватимуться з хмарним сховищем за допомогою REST API [18].

Модуль відстеження прогресу опрацьовуватиме дві основні категорії даних, які відрізняються за часовою структурою та характером формування: поточний статус виконання звичок та історичні дані про прогрес. Поточний статус включатиме інформацію про завершені та незавершені звички на поточний день, тиждень та місяць. Історичні дані міститимуть інформацію про тривалі серії успішного виконання звичок, пропущені дні та загальну статистику. Для ефективної роботи з такими даними оптимальним вибором є використання локального сховища Room для зберігання історичних даних та LiveData/Flow для актуалізації інформації в реальному часі [19].

Модуль аналізу користувацьких даних відповідатиме за обробку та інтерпретацію інформації про виконання звичок з метою формування персоналізованих рекомендацій. Призначення цієї частини системи полягає в оперуванні метаданими поведінкових патернів, такими як найбільш продуктивний час доби, взаємозв'язок між різними звичками, фактори, що впливають на успішність виконання, та потенційні перешкоди. Аналіз даних буде реалізовано через використання алгоритмів машинного навчання та статистичних методів, що забезпечить високий рівень персоналізації. Для цього модуля важливо забезпечити ефективну обробку часових рядів та виявлення прихованих закономірностей у даних користувача.

Модуль персоналізованих сповіщень оперуватиме даними про оптимальний час нагадувань, ефективність різних типів мотиваційних повідомлень та особливості взаємодії користувача зі сповіщеннями. Цей модуль забезпечуватиме управління інформацією про заплановані нагадування, історію реакцій на сповіщення та адаптацію стилю комунікації відповідно до преференцій користувача. Для ефективного функціонування системи критично важливим є забезпечення точного тайм-менеджменту та оптимізації частоти сповіщень для запобігання їх ігнорування. Для реалізації цього модуля доцільно використовувати можливості Android Notifications API та WorkManager для планування відкладених

операцій.

Модуль гейміфікації відповідатиме за мотиваційні механізми, що базуються на досягненнях, рівнях прогресу та соціальному порівнянні. Дані цього модуля включатимуть інформацію про доступні та отримані досягнення, поточний рівень користувача, накопичені бали та потенційні винагороди. Використання елементів гейміфікації дозволить підтримувати мотивацію користувача у довгостроковій перспективі та створити додаткові стимули для формування корисних звичок.

У якості засобів для комунікації з серверною частиною доцільно використати RESTful API для синхронізації даних про звички та прогрес, а також Firebase Cloud Messaging для сповіщень та оновлення статусу в реальному часі. Для локального зберігання даних підійде Room Database, яка дозволить ефективно зберігати структуровану інформацію про звички та забезпечить швидкий доступ до неї навіть без підключення до мережі [20].

Отже, аналіз вхідних даних системи персоналізованого формування корисних звичок визначає необхідність розробки п'яти функціональних модулів, кожен з яких працює зі специфічними наборами даних та вимагає відповідних підходів до їх відображення та обробки. Реалізація системи на основі Kotlin з використанням архітектури MVVM забезпечить надійну основу для розробки адаптивного мобільного додатку. Комбінований підхід до організації даних з використанням серверного API для синхронізації основної інформації та локальних сховищ для зберігання і аналізу даних дозволить досягти оптимального балансу між продуктивністю роботи та рівнем персоналізації. Використовуючи такий набір технологій при ретельному плануванні та проєктуванні можна створити програмний продукт, який дозволить ефективно формувати корисні звички з урахуванням індивідуального прогресу користувача.

### 2.2 Аналіз архітектури застосунку

Застосунок розроблено за принципами чистої архітектури (Clean Architecture) з використанням модульного підходу та дотриманням SOLID принципів. Архітектура організована в чотири основні шари, кожен з яких має

чітко визначену відповідальність. Така структура забезпечує легке тестування, розширюваність та підтримку додатку.

Шар інтерфейсу відповідає за відображення даних та взаємодію з користувачем. До цього шару належать екрани HabitListScreen, EditHabitScreen, UserProgressScreen, HabitProgressScreen та багаторазові компоненти HabitItem, StreakIndicator, ProgressOverviewCard.

Особливістю цього шару є його цілковита незалежність від бізнес-логіки. Compose компоненти отримують дані та обробники подій через параметри і не містять власної логіки обробки даних. Це забезпечує чистий розподіл відповідальності та полегшує тестування інтерфейсу.

Шар представлення містить ViewModels та навігаційну логіку. HabitViewModel є центральною точкою цього шару і відповідає за обробку дій користувача, підготовку даних для UI та координацію операцій з нижчими шарами. NavigationManager забезпечує централізоване керування навігацією, уникаючи прямої залежності між компонентами UI.

Цей шар реалізує патерн MVVM (Model-View-ViewModel), де ViewModel виступає посередником між UI та бізнес-логікою. ViewModels використовують StateFlow та SharedFlow для публікації даних та подій до UI в реактивному стилі, забезпечуючи автоматичне оновлення інтерфейсу при зміні даних. Шар домену містить бізнес-логіку додатку незалежно від конкретних реалізацій інтерфейсу чи джерел даних. Він включає класи моделей (Habit, ProgressMetrics, Milestone), сервіси аналізу даних (ProgressAnalyzer) та використовує класи даних для опису бізнес-правил.

Важливою частиною цього шару є механізми аналізу прогресу, такі як алгоритми розрахунку консистентності та генерації персоналізованих рекомендацій. Логіка цього шару не залежить від конкретних технологій, що робить її добре тестованою та потенційно переносимою на інші платформи.

Шар даних відповідає за доступ до всіх джерел даних, включаючи локальну базу даних Room. Він містить репозиторії (HabitRepository, ProgressRepository), які надають абстрактний інтерфейс для доступу до даних, та конкретні реалізації

джерел даних.

Взаємодія між шарами відбувається згідно з принципом залежності від абстракцій. Вищі шари залежать від абстракцій (інтерфейсів) нижчих шарів, але не від їх конкретних реалізацій. Ін'єкція залежностей за допомогою Hilt забезпечує слабкий зв'язок між компонентами та спрощує тестування. Потік даних у додатку є однонаправленим: дані з бази даних через репозиторії потрапляють до ViewModel, який обробляє їх та публікує у StateFlow для відображення в UI. Події користувача, такі як натискання на кнопки, рухаються у зворотному напрямку: від UI до ViewModel, і далі до відповідних компонентів нижчих шарів.

Розроблена архітектура застосунку наведена на рисунку 2.1.

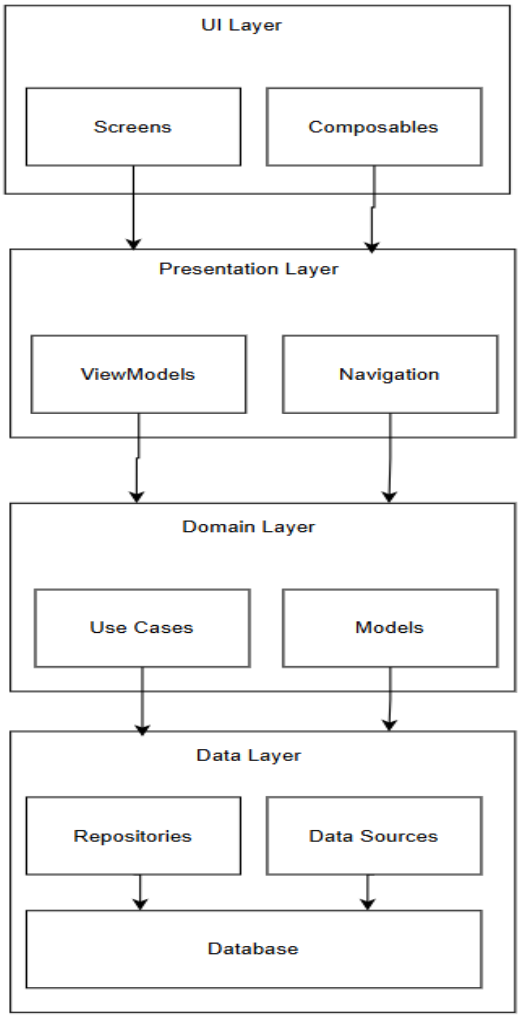


Рисунок 2.1 — Архітектура застосунку

Розроблена архітектура забезпечує чіткий розподіл відповідальності, слабкий зв'язок між компонентами та дотримання принципів SOLID. Модульний підхід сприяє розширюваності та підтримуваності додатку, а використання сучасних технологій Kotlin та Jetpack Compose дозволяє створити ефективне та зручне для користувача рішення. Реактивний підхід до обробки даних та подій забезпечує відмінний користувацький досвід при збереженні чистоти архітектури.

### 2.3 Розробка моделі системи

На основі аналізу функціональних вимог до мобільної системи персоналізованого формування корисних звичок було розроблено модель системи, представлену у вигляді блок-схеми. Ця модель відображає основні компоненти системи та логіку взаємодії між ними, забезпечуючи повне розуміння процесів, що відбуваються в системі (див. рисунок 2.2).

Модель починається з етапу входу користувача у застосунок, що є початковою точкою взаємодії з системою. Після входу користувачу надається вибір між двома основними шляхами: реєстрацією для нових користувачів або авторизацією для вже зареєстрованих. Обидва ці шляхи ведуть до головної сторінки, яка є центральним компонентом користувацького інтерфейсу системи.

З головної сторінки користувач може перейти до функціоналу додавання нової звички. При виборі цієї опції система переходить до екрану створення звички, де користувач може ввести необхідну інформацію про звичку, яку він хоче сформувати. Після заповнення всіх необхідних полів користувач може зберегти звичку, що ініціює процес збереження даних у базі даних системи.

Після збереження звички відбувається оновлення списку звичок, що забезпечує актуальність відображення інформації для користувача. Користувач отримує можливість переглянути деталі збереженої звички, що є важливим етапом для подальшої роботи з нею.

При перегляді деталей звички система пропонує три основні напрямки аналізу: статистику виконання, календар виконання та персоналізовані

рекомендації. Статистика виконання надає користувачу кількісну інформацію про його прогрес, дозволяючи оцінити ефективність формування звички. Календар виконання представляє хронологічний вигляд активності користувача, що допомагає відстежувати регулярність виконання звички. Рекомендації ж базуються на аналізі прогресу користувача та надають персоналізовані поради щодо покращення процесу формування звички.

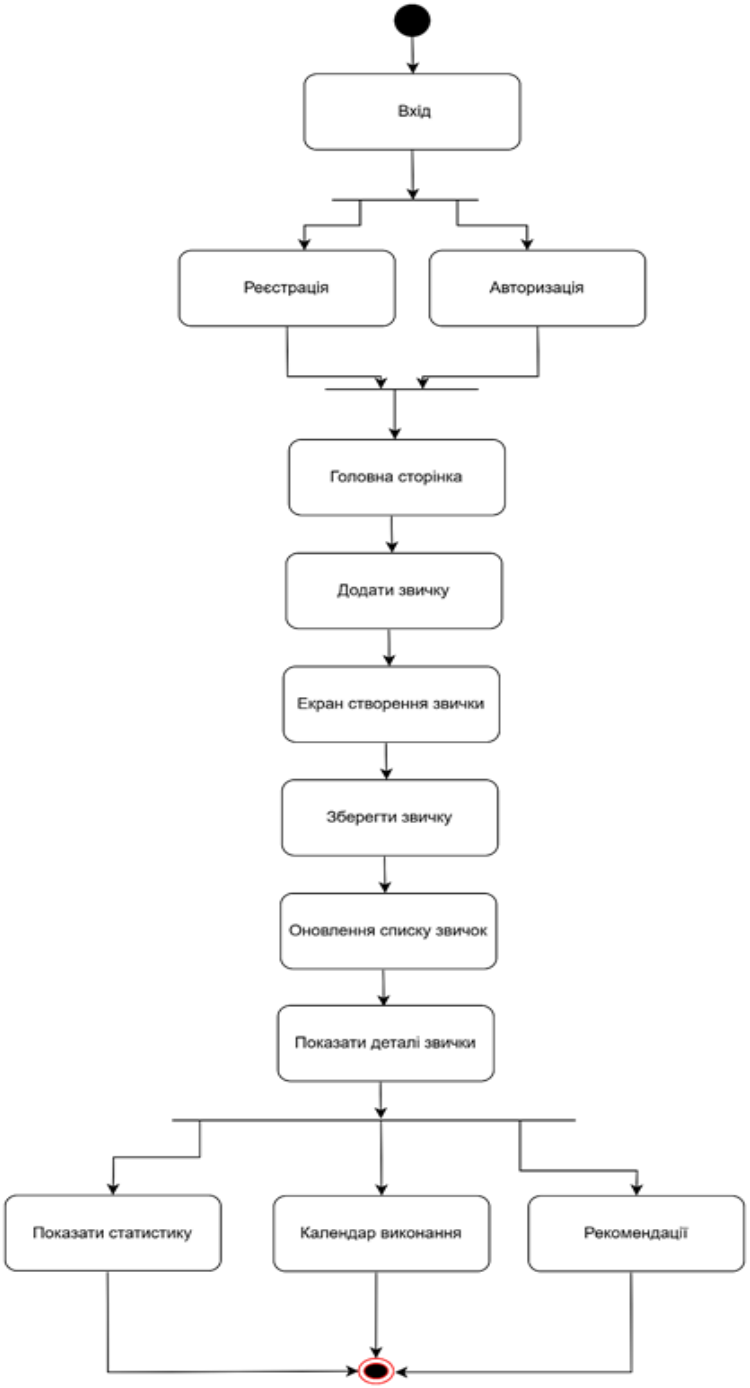


Рисунок 2.2 – Діаграма діяльності програмних засобів

Усі три компоненти аналізу (статистика, календар та рекомендації) завершуються в єдиній кінцевій точці, що символізує завершення процесу аналізу звички. Така структура забезпечує цілісний підхід до відстеження та аналізу прогресу користувача у формуванні корисних звичок.

Розроблена модель системи враховує всі основні функціональні вимоги та забезпечує логічну послідовність дій користувача при роботі з системою. Модель є основою для подальшої розробки програмних модулів та інтерфейсів користувача, забезпечуючи єдине розуміння структури та функціональності системи для всіх учасників процесу розробки.

### 2.3 Розробка методу динамічної персоналізованої оцінки прогресу

Метод динамічної персоналізованої оцінки прогресу аналізує індивідуальні патерни поведінки користувача та генерує адаптивні рекомендації на основі реальних даних використання. Замість універсальних порад, система враховує оптимальний час виконання, фактори, що заважають регулярності, та взаємозв'язки між різними звичками користувача.

Використання цього функціоналу підвищує ефективність формування корисних звичок завдяки точному налаштуванню під особливості кожного користувача.

Послідовність виконання методу:

1. Система отримує дані про виконання звички через інтерфейс користувача.
2. Дані зберігаються в таблиці Completions, фіксуючи факт та час виконання.
3. При запиті персоналізованих рекомендацій викликається метод generatePersonalizedSuggestions().
4. Метод аналізує всі записи виконання для заданої звички та обчислює часові патерни.
5. Система визначає оптимальні часові вікна на основі успішних виконань.
6. Проводиться аналіз пропущених виконань для виявлення типових перешкод.
7. На основі аналізу формуються конкретні рекомендації, адаптовані під

користувача.

8. Рекомендації зберігаються у системі та відображаються користувачу в зрозумілому форматі.

9. Система продовжує моніторинг виконання для подальшого уточнення рекомендацій.

Метод динамічної персоналізованої оцінки прогресу перетворює стандартний трекер звичок у розумного особистого помічника, який здатен розпізнавати індивідуальні особливості користувача та пропонувати релевантні рекомендації. Ефективність цього підходу полягає у постійній адаптації до змін у поведінці користувача, що робить формування звичок більш природним процесом, орієнтованим на індивідуальні потреби та особливості.

### 2.4 Розробка методу інтелектуального аналізу консистентності звичок

Метод інтелектуального аналізу консистентності виходить за межі простого підрахунку виконаних завдань, застосовуючи комплексну математичну модель для оцінки стабільності звички. Метод враховує не лише факт виконання, але й регулярність інтервалів, нещодавні тенденції та довгострокову стабільність відповідно до запланованої частоти.

Завдяки багатофакторному підходу до аналізу консистентності, користувачі отримують більш реалістичну оцінку свого прогресу у формуванні звичок. Це дозволяє ефективніше визначати, чи стала звичка справді частиною життя, а не просто серією випадкових виконань.

Послідовність виконання:

1. Система отримує список всіх виконань звички з бази даних.

2. Метод calculateConsistencyScore() обчислює очікувані дні виконання на основі частоти звички.

3. Розраховується базовий відсоток виконання як відношення кількості фактичних виконань до очікуваних.

4. Система обчислює інтервали між послідовними виконаннями.

5. На основі інтервалів розраховується середнє значення та стандартне

відхилення.

6. Обчислюється коефіцієнт варіації як міра стабільності інтервалів.

7. Система аналізує нещодавні тенденції, надаючи більшу вагу останнім виконанням.

8. Результати трьох факторів (базовий відсоток, стабільність інтервалів, нещодавні тенденції) зважуються для отримання фінальної оцінки консистентності.

9. Фінальна оцінка зберігається в таблиці Progress Metrics та відображається користувачу.

Метод інтелектуального аналізу консистентності забезпечує глибше розуміння реального прогресу у формуванні звичок, виходячи за рамки простих кількісних показників. Цей метод дає користувачам об'єктивну картину їхньої регулярності та допомагає ідентифікувати проблемні аспекти, які потребують уваги. Завдяки комплексній оцінці різних аспектів виконання, користувачі отримують потужний інструмент для моніторингу свого справжнього прогресу у формуванні стійких корисних звичок.

### 2.5 Розробка алгоритмів роботи системи

У рамках розробки мобільної системи персоналізованого формування корисних звичок було розроблено ключові алгоритми, що забезпечують основну функціональність програмного продукту. Основними компонентами системи є генерація персоналізованих рекомендацій та відстеження прогресу користувача.

Було реалізовано два основні алгоритми: алгоритм формування персоналізованих рекомендацій (див. рисунок 2.3) та алгоритм відстеження досягнень користувача (див. рисунок 2.4).

Алгоритм формування персоналізованих рекомендацій призначений для аналізу прогресу користувача та формування відповідних рекомендацій щодо покращення виконання звички. Процес роботи алгоритму починається з отримання ідентифікатора звички (habitId) та даних про неї з репозиторію habitRepository.

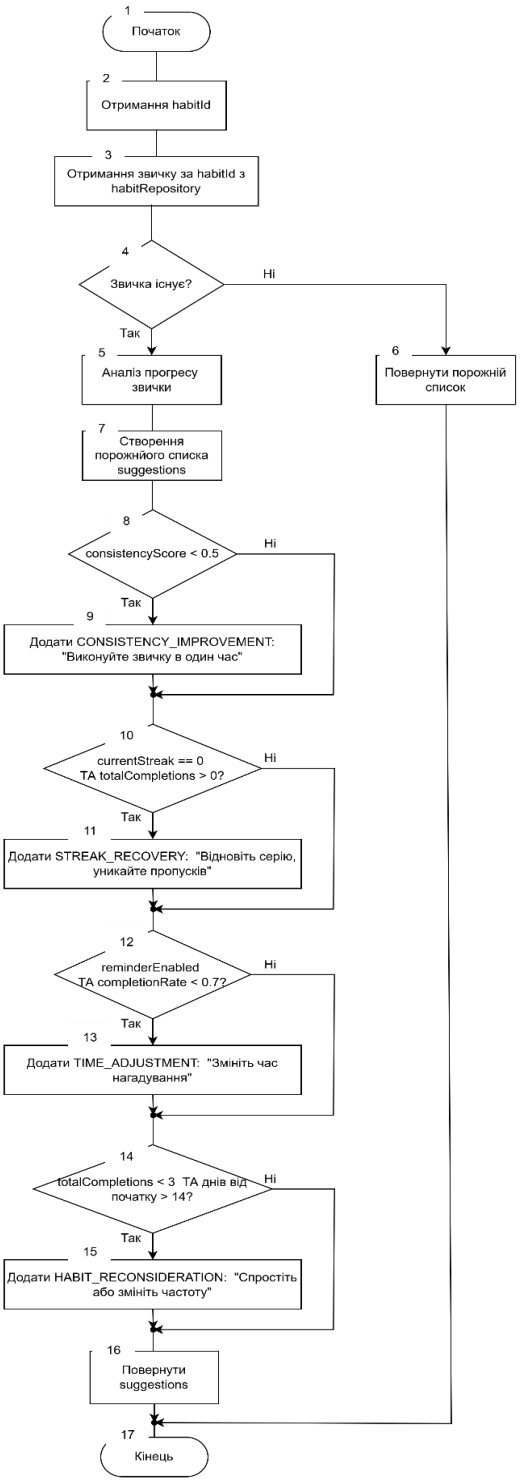


Рисунок 2.3 – Алгоритм формування персоналізованих рекомендацій

Далі відбувається перевірка існування звички – якщо звичка не існує, повертається порожній список рекомендацій, якщо ж звичка існує, відбувається подальший аналіз її прогресу на основі історичних даних. Після цього

створюється порожній список рекомендацій (suggestions), який поступово наповнюється на основі аналізу різних показників. При низькому показнику послідовності (consistencyScore < 0.5) додається рекомендація "CONSISTENCY_IMPROVEMENT: 'Виконуйте звичку в один час'". Якщо поточна серія виконань перервана (currentStreak = 0), але загальна кількість виконань більша нуля (totalCompletions > 0), додається рекомендація "STREAK_RECOVERY: 'Відновіть серію, уникайте пропусків'". При наявності активованих нагадувань та низького показника успішності (remindEnabled ТА completionRate < 0.7) формується рекомендація "TIME_ADJUSTMENT: 'Змініть час нагадування'". Для довгострокових звичок з низькою активністю (totalCompletions < 3 ТА днів від початку > 14) пропонується переглянути доцільність звички через рекомендацію "HABIT_RECONSIDERATION: 'Спробуйте або змініть частоту'". Завершується алгоритм поверненням сформованого списку рекомендацій.

Для ефективного відстеження прогресу користувача розроблено алгоритм, який формує досягнення (milestones) на основі активності користувача. Алгоритм починається з отримання ідентифікатора звички та аналізу її прогресу. Потім з progressRepository отримується інформація про поточні досягнення та створюється порожній список для нових досягнень (newMilestones). В алгоритмі визначено два типи досягнень: за серію послідовних виконань (streakMilestones) із значеннями 3, 7, 14, 30, 60, 90, 180, 365 днів, та за загальну кількість виконань (completionMilestones) із значеннями 10, 25, 50, 100, 250, 500, 1000 виконань. Якщо поточна серія виконань (currentStreak) досягає або перевищує одне із значень зі списку streakMilestones, створюється нове досягнення, яке додається до newMilestones та зберігається в progressRepository. Аналогічно, якщо загальна кількість виконань (totalCompletions) досягає або перевищує одне зі значень зі списку completionMilestones, також створюється відповідне досягнення. Результатом роботи алгоритму є список нових досягнень, які користувач отримав під час поточної перевірки.


Рисунок 2.4 – Алгоритм відстеження досягнень користувача

Обидва алгоритми тісно взаємодіють між собою, забезпечуючи комплексний підхід до формування корисних звичок. Аналіз прогресу користувача

використовується як для формування персоналізованих рекомендацій, так і для відстеження досягнень, що створює єдину екосистему для ефективного формування та підтримки корисних звичок. Розроблені алгоритми реалізовані мовою програмування Kotlin з використанням сучасних патернів програмування та архітектурних принципів, що забезпечує їхню ефективність, масштабованість та підтримуваність.

### 2.6 Висновки

У другому розділі представлено розгорнутий опис архітектурної моделі системи проєкту із застосуванням широкого спектру інструментів UML-моделювання, включаючи діаграми класів, послідовностей, компонентів та розгортання. Окрема увага приділена детальному опису ключових алгоритмів, що забезпечують ефективне функціонування мобільної системи персоналізованого формування звичок, зокрема алгоритмів обробки користувацьких даних, персоналізації рекомендацій та оптимізації циклу зворотного зв'язку.

# 3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ ПЕРСОНАЛІЗОВАНОГО ФОРМУВАННЯ КОРИСНИХ ЗВИЧОК З УРАХУВАННЯМ КОРИСТУВАЦЬКОГО ПРОГРЕСУ

## 3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Розробка мобільної системи персоналізованого формування корисних звичок потребує ретельного вибору технологічних засобів, які забезпечать оптимальну продуктивність, зручність використання та підтримуваність системи. Ефективний вибір інструментів розробки має враховувати як функціональні вимоги до платформи, так і особливості цільової аудиторії та специфіку відстеження звичок.

При проєктуванні мобільної системи було розглянуто декілька варіантів реалізації клієнтської частини, кожен з яких має свої переваги та обмеження. Основними критеріями оцінки стали: продуктивність, підтримка персоналізації, можливості для локального зберігання даних, інтеграція з нативними API смартфонів, користувацький досвід, підтримка спільноти та перспективи розвитку технології [21].

У контексті вибору базової технології для побудови мобільної системи було проаналізовано три основні варіанти: розробка нативного мобільного додатку для Android, створення кросплатформного рішення з використанням Flutter або React Native, та реалізація прогресивного веб-додатку (PWA).

Перший варіант передбачав створення нативного додатку для Android з використанням мови Kotlin та Android SDK. Цей підхід забезпечує повний доступ до всіх функцій пристрою, найвищу продуктивність та найкращу інтеграцію з операційною системою. Нативна розробка дозволяє використовувати всі можливості Android для реалізації сповіщень, фонових процесів та доступу до апаратних сенсорів, що є критичним для системи формування звичок, яка повинна нагадувати користувачу про виконання звичок та відстежувати прогрес. Однак, цей підхід обмежує цільову аудиторію користувачами Android-пристроїв.

Другий варіант включав використання кросплатформних технологій, таких як Flutter або React Native. Цей підхід дозволяє розробити додаток, який працює як на Android, так і на iOS, використовуючи єдину кодову базу. Flutter забезпечує високу продуктивність та нативний зовнішній вигляд інтерфейсу, а React Native дозволяє розробникам використовувати JavaScript та React для створення мобільних додатків. Проте, кросплатформні рішення мають обмеження у доступі до нативних API та можуть відставати у підтримці найновіших функцій операційних систем.

Третій варіант базувався на створенні прогресивного веб-додатку, який доступний через браузер і може бути встановлений на домашній екран пристрою. PWA забезпечує найширше охоплення пристроїв, включаючи не тільки смартфони, але й планшети та комп'ютери [22]. Однак, PWA має значні обмеження щодо доступу до нативних функцій пристрою, особливо для фонової роботи та push-сповіщень, що є критичним для системи формування звичок [23].

Після аналізу варіантів та оцінки їх відповідності вимогам проєкту, було обрано перший варіант з використанням Kotlin та нативної розробки для Android як основної технології для реалізації мобільної системи. Обґрунтуванням цього вибору стали такі фактори:

Kotlin забезпечує сучасний, безпечний та експресивний підхід до розробки Android-додатків, будучи офіційно підтримуваною мовою для Android-розробки з 2017 року. Kotlin пропонує значні переваги порівняно з Java: більш лаконічний синтаксис, null-safety для уникнення помилок NullPointerException, підтримка функціонального програмування та розширення функцій, що дозволяє писати більш чистий та підтримуваний код.

Нативна розробка для Android забезпечує повний доступ до API операційної системи, що є критичним для реалізації таких функцій, як сповіщення про виконання звичок, збір даних з сенсорів для автоматичного відстеження прогресу (наприклад, кроків за допомогою акселерометра) та фонова синхронізація даних.

Для архітектури додатку було обрано шаблон MVVM (Model-View-ViewModel) у поєднанні з принципами чистої архітектури. MVVM було обрано

через його підтримку у Android Jetpack, що забезпечує компоненти ViewModel та LiveData для управління даними, пов'язаними з UI, та їх життєвим циклом. Чиста архітектура забезпечує розділення додатку на шари з чіткими залежностями, що покращує тестованість та підтримуваність коду.

Для побудови інтерфейсу користувача було розглянуто два основні підходи: традиційний підхід з використанням XML-макетів та новий декларативний підхід з використанням Jetpack Compose. Було вирішено використовувати Jetpack Compose як сучасний інструмент для розробки UI, який помітно спрощує створення інтерактивних та анімованих інтерфейсів, що є важливим для забезпечення позитивного досвіду користувача при взаємодії з системою формування звичок. Jetpack Compose також краще інтегрується з Kotlin та дозволяє використовувати його функціональні можливості для опису інтерфейсу[24].

Для локального зберігання даних було обрано Room Persistence Library, яка забезпечує абстракцію над SQLite та спрощує роботу з базою даних. Room надає компіляційну перевірку SQL-запитів, легку інтеграцію з LiveData та Kotlin Coroutines, а також підтримку міграцій бази даних, що важливо для підтримки різних версій додатку.

Для керування асинхронними операціями було вирішено використовувати Kotlin Coroutines та Flow. Coroutines забезпечують просту та ефективну роботу з асинхронним кодом без використання колбеків, а Flow надає реактивний підхід до роботи з потоками даних, що особливо корисно для відстеження прогресу користувача у реальному часі.

Для взаємодії з віддаленим сервером та синхронізації даних між пристроями було обрано Retrofit у поєднанні з OkHttp для виконання HTTP-запитів. Retrofit забезпечує типобезпечний HTTP-клієнт, що спрощує інтеграцію з REST API, а OkHttp надає ефективні механізми кешування та перехоплення запитів.

Для впровадження залежностей було обрано Dagger Hilt, який спрощує налаштування Dagger у Android-додатках та забезпечує стандартизований підхід до DI. Hilt інтегрується з компонентами Android Jetpack та забезпечує автоматичне

керування життєвим циклом залежностей, що спрощує розробку та тестування.

Для реалізації персоналізованих рекомендацій та прогнозування поведінки користувача було вирішено використовувати TensorFlow Lite для машинного навчання на пристрої. Це дозволяє обробляти дані користувача локально, без відправки їх на сервер, що покращує приватність та зменшує залежність від мережевого з'єднання.

Для відстеження аналітики та помилок було обрано Firebase Analytics та Crashlytics, які забезпечують інструменти для аналізу поведінки користувачів та виявлення проблем у додатку.

Таким чином, обраний технологічний стек, що базується на Kotlin та нативній розробці для Android, забезпечує оптимальний баланс між продуктивністю, доступом до системних функцій та зручністю розробки, що відповідає вимогам до мобільної системи персоналізованого формування корисних звичок.

Використання сучасних компонентів Android Jetpack забезпечує надійну основу для розробки, а інтеграція з Firebase надає потужні інструменти для аналітики та моніторингу додатку. Обрана комбінація інструментів дозволяє реалізувати всі заплановані функціональні модулі системи, забезпечуючи високу якість користувацького досвіду та ефективне відстеження прогресу користувача у формуванні корисних звичок.

### 3.2 Розробка бази даних

База даних у застосунку персоналізованого формування корисних звичок виконує фундаментальну роль у забезпеченні повноцінної функціональності системи динамічної оцінки користувацького прогресу. Вона слугує центральним сховищем усієї критично важливої інформації, що дозволяє застосунку ефективно аналізувати поведінкові патерни та генерувати адаптивні рекомендації для кожного користувача [25].

Було розроблено схему бази даних, яка наведена на рисунку 3.1.


Рисунок 3.1 – Структура бази даних для додатку відстеження звичок

Таблиця Habits слугує основним сховищем для визначень та налаштувань звичок. Вона містить усю необхідну інформацію для відображення, планування та відстеження звички.

Поля таблиці Habits наведені у таблиці 3.1.

Таблиця 3.1 — Таблиця Habits

| Поле | Тип | Опис |
|---|---|---|
| id | Long | Первинний ключ, що унікально ідентифікує кожну звичку. Автоінкрементується. |
| title | String | Назва звички, яка відображається користувачу. |
| description | String | Додатковий детальний опис звички. |
| frequency | String | Зберігає шаблон повторення звички (DAILY, WEEKLY, WEEKDAYS, WEEKENDS, SPECIFIC_DAYS, MONTHLY). Зберігається як текстове представлення значення переліку. |
| timeOfDayHour | Integer | Компонент години цільового часу для виконання звички. Використовується для нагадувань. |
| timeOfDayMinute | Integer | Компонент хвилин цільового часу для виконання звички. Використовується для нагадувань. |
| reminderEnabled | Boolean | Вказує, чи повинні надсилатися сповіщення для нагадування користувачу про цю звичку. |
| startDate | Long | Дата початку відстеження цієї звички, зберігається як кількість днів від епохи для ефективних обчислень дат. |
| colorValue | Integer | Значення кольору ARGB, що використовується для візуального представлення звички в інтерфейсі. |

Таблиця Completions відстежує кожен випадок виконання звички, створюючи історичний запис, який дозволяє розраховувати серії, аналізувати регулярність та візуалізувати прогрес.

Поля таблиці Completions наведені у таблиці 3.2.

Таблиця 3.2 — Поля таблиці Completions

| **Поле** | **Тип** | **Опис** |
|---|---|---|
| id | Long | Первинний ключ, що унікально ідентифікує кожен запис виконання. Автоінкрементується. |
| habitId | Long | Зовнішній ключ, що посилається на таблицю Habits для асоціації виконання з конкретною звичкою. |
| completionDate | Long | Дата виконання звички, зберігається як кількість днів від епохи для полегшення запитів та обчислень на основі дат. |

Таблиця Milestones записує значущі досягнення, пов'язані зі звичками, забезпечуючи основу для функцій гейміфікації та визнання прогресу.

Поля таблиці Milestones наведені у таблиці 3.3.

Таблиця 3.3 — Поля таблиці Milestones

| **Поле** | **Тип** | **Опис** |
|---|---|---|
| id | Long | Первинний ключ, що унікально ідентифікує кожне досягнення. Автоінкрементується. |
| habitId | Long | Зовнішній ключ, що посилається на таблицю Habits для асоціації досягнення з конкретною звичкою. |
| type | String | Категоризує досягнення (наприклад, STREAK_MILESTONE, COMPLETION_COUNT, CONSISTENCY_ACHIEVEMENT). Зберігається як текстове представлення значення переліку. |
| achievedDate | Long | Дата досягнення, зберігається як кількість днів від епохи. |
| value | Integer | Числове значення, пов'язане з досягненням, наприклад, довжина серії або кількість виконань, що активували досягнення. |

Таблиця Progress Metrics слугує для оптимізації продуктивності, кешуючи розраховані метрики для кожної звички, уникаючи повторної обробки історичних даних при відображенні інформації про прогрес.

Поля таблиці Progress Metrics наведені у таблиці 3.4.

Таблиця 3.4 — Поля таблиці Progress Metrics

| **Поле** | **Тип** | **Опис** |
|---|---|---|
| habitId | Long | Первинний ключ та зовнішній ключ, що посилається на таблицю Habits. |
| consistencyScore | Float | Значення від 0 до 1, що представляє наскільки регулярно виконується звичка відповідно до запланованої частоти. |
| completionRate | Float | Значення від 0 до 1, що вказує на загальний рівень виконання відносно очікуваних виконань. |
| currentStreak | Integer | Кількість послідовних днів/періодів, протягом яких звичка виконувалася без пропусків. |
| longestStreak | Integer | Рекорд найдовшої серії, досягнутої для цієї звички. |
| totalCompletions | Integer | Загальна кількість виконань звички з початку відстеження. |
| lastUpdated | Long | Дата останнього розрахунку цих метрик, зберігається як кількість днів від епохи, використовується для визначення необхідності перерахунку. |

Ця структура бази даних забезпечує ефективне зберігання та отримання даних про звички, підтримуючи складну аналітику для відстеження прогресу та персоналізованих рекомендацій. Зв'язки між таблицями підтримують цілісність даних, зовнішні ключі забезпечують правильну асоціацію пов'язаних записів.

### 3.3 Розробка модуля генерації порад

Модуль генерації персоналізованих порад є ключовим компонентом розробленої системи, який забезпечує інтелектуальну підтримку користувача у формуванні корисних звичок. Даний модуль реалізований за допомогою функції generatePersonalizedSuggestions, яка приймає ідентифікатор звички як вхідний параметр та повертає список рекомендацій щодо покращення процесу формування конкретної звички.

Основний алгоритм генерації порад базується на аналізі чотирьох ключових аспектів формування звички: консистентність виконання, стан поточної серії виконань, ефективність часу нагадувань та загальна релевантність звички для користувача (див. рисунок 3.1).

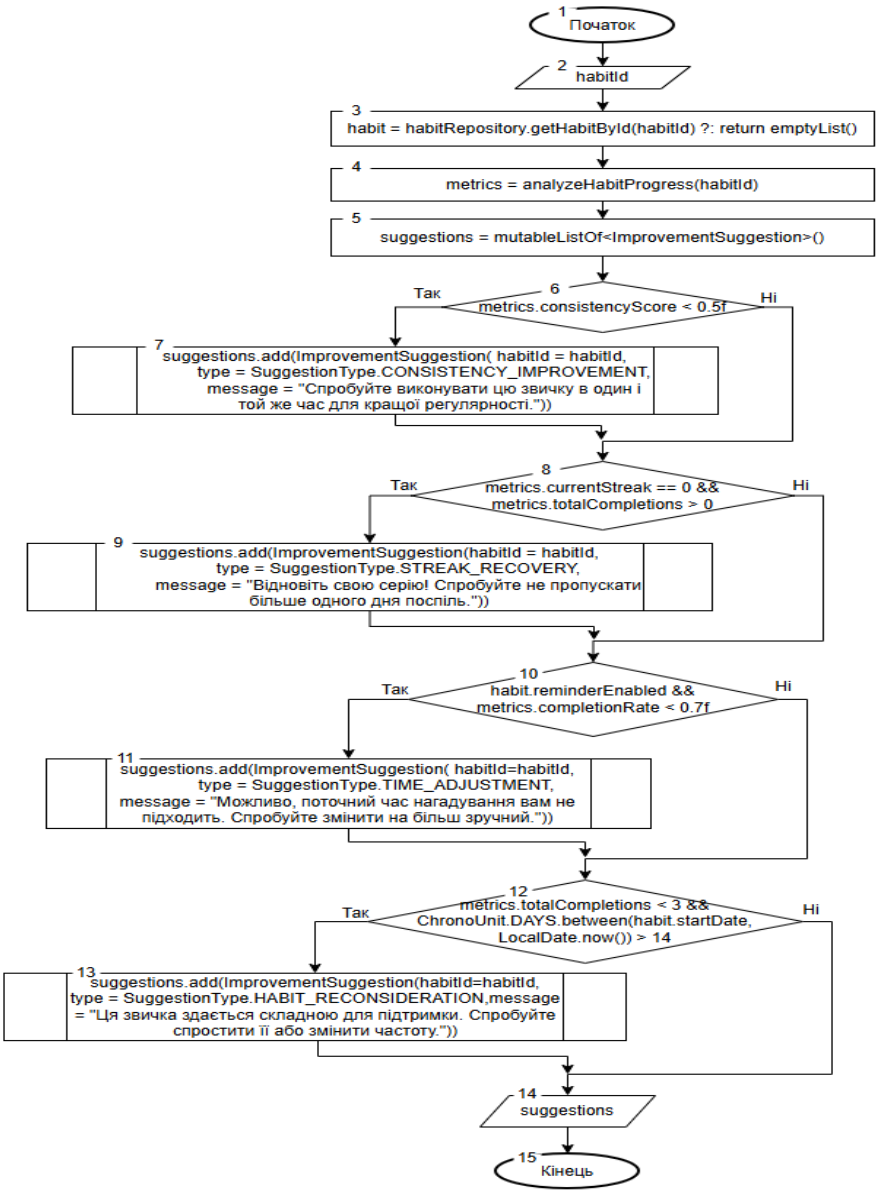


Рисунок 3.1 – Модуль генерації порад

Функція generatePersonalizedSuggestions є асинхронною, що дозволяє виконувати операції з базою даних без блокування основного потоку виконання програми. Це забезпечує плавність інтерфейсу користувача та відповідає сучасним практикам розробки мобільних додатків на Kotlin.

На початку функції відбувається отримання інформації про звичку з репозиторію за допомогою виклику habitRepository.getHabitById(habitId). Якщо звичка не знайдена, функція повертає порожній список, що запобігає виникненню помилок при виконанні подальших операцій. Важливим етапом є аналіз прогресу звички через виклик функції analyzeHabitProgress, яка обчислює різні метрики

ефективності формування звички.

При аналізі консистентності система перевіряє значення метрики consistencyScore. Якщо цей показник нижче порогового значення 0.5, система генерує пораду про необхідність виконання звички в один і той же час для покращення регулярності. Такий підхід базується на психологічних дослідженнях, які показують, що звички краще формуються при фіксованому часі виконання.

Аналіз серії виконань спрямований на мотивацію користувача не переривати послідовність виконання звички. Якщо поточна серія дорівнює нулю (currentStreak == 0), але загальна кількість виконань більше нуля (totalCompletions > 0), система рекомендує відновити серію та не пропускати більше одного дня поспіль. Цей механізм використовує принцип «не розривати ланцюжок» (don't break the chain), популяризований Джеррі Сайнфельдом.

Модуль також аналізує ефективність встановлених нагадувань. Якщо для звички увімкнено нагадування (habit.reminderEnabled), але рівень виконання нижче 70% (completionRate < 0.7f), система пропонує користувачу змінити час нагадування на більш зручний. Цей підхід враховує індивідуальні особливості розпорядку дня користувача та сприяє підвищенню ефективності нагадувань.

Особливу увагу система приділяє переосмисленню звички, якщо вона виявляється складною для користувача. Якщо загальна кількість виконань менше трьох (totalCompletions < 3) за період більше двох тижнів з моменту початку (ChronoUnit.DAYS.between(habit.startDate, LocalDate.now()) > 14), система рекомендує спростити звичку або змінити частоту її виконання. Цей механізм запобігає демотивації користувача через надмірно складні або нереалістичні цілі.

Варто зазначити, що всі рекомендації створюються як об'єкти класу ImprovementSuggestion, який включає ідентифікатор звички, тип рекомендації та текстове повідомлення з конкретною порадою. Типи рекомендацій представлені переліченням SuggestionType, що включає категорії CONSISTENCY_IMPROVEMENT, STREAK_RECOVERY, TIME_ADJUSTMENT та HABIT_RECONSIDERATION.

Архітектурно модуль генерації порад є частиною сервісного шару додатку

та взаємодіє з репозиторієм для отримання даних та аналітичним модулем для обчислення метрик. Такий підхід забезпечує розділення відповідальності та полегшує тестування й модифікацію окремих компонентів системи.

Систему порад можна розширити додатковими типами рекомендацій та алгоритмами аналізу, що дозволить адаптувати систему до різних типів звичок та індивідуальних особливостей користувачів. Наприклад, можна впровадити аналіз кореляції між успішністю формування звички та факторами зовнішнього середовища, такими як погода, день тижня або соціальна активність.

Загалом, розроблений модуль генерації персоналізованих порад є важливою складовою системи формування корисних звичок, що забезпечує індивідуальний підхід до кожного користувача та підвищує ефективність процесу формування звичок шляхом надання конкретних та своєчасних рекомендацій на основі аналізу користувацького прогресу.

### 3.4 Розробка модуля перевірки та створення нових досягнень

Модуль перевірки та створення нових досягнень є важливим компонентом системи гейміфікації, що підвищує мотивацію користувача до регулярного виконання звичок. Цей модуль реалізований через асинхронну функцію checkAndCreateMilestones, яка дозволяє виявити та зафіксувати значущі етапи прогресу у формуванні конкретної звички.

Функція checkAndCreateMilestones приймає ідентифікатор звички та повертає список нових досягнень, які користувач отримав у результаті своєї активності. Асинхронний характер функції, позначений ключовим словом suspend, забезпечує ефективне виконання операцій з базою даних без блокування головного потоку програми, що є важливим аспектом розробки мобільних додатків на Kotlin з використанням корутинів.

На початковому етапі функція отримує метрики прогресу за допомогою виклику analyzeHabitProgress(habitId) та завантажує список існуючих досягнень для конкретної звички з репозиторію. Такий підхід дозволяє уникнути дублювання досягнень та забезпечує актуальність даних при перевірці.

Модуль виконує аналіз двох основних типів досягнень: досягнення за серією безперервних виконань звички (STREAK_MILESTONE) та досягнення за загальною кількістю виконань (COMPLETION_COUNT). Такий поділ дозволяє мотивувати користувача як до регулярності (безперервність), так і до загальної наполегливості (загальна кількість).

Для досягнень, пов'язаних з безперервною серією виконань, визначено вісім порогових значень: 3, 7, 14, 30, 60, 90, 180 та 365 днів. Ці значення вибрані з урахуванням психологічних особливостей формування звичок та представляють символічні етапи на шляху до повноцінного засвоєння звички. Зокрема, дослідження показують, що перші 3-7 днів є критичними для початкової фази, 21-30 днів – для закріплення базового шаблону поведінки, а 60-90 днів – для трансформації нової поведінки у стійку звичку.

Для досягнень за загальною кількістю виконань встановлено сім порогових значень: 10, 25, 50, 100, 250, 500 та 1000. Ці значення демонструють прогрес користувача в довгостроковій перспективі та нагороджують за наполегливість навіть у випадках, коли серія переривається. Логарифмічний характер зростання цих значень узгоджується з психологічними принципами мотивації, коли початкові досягнення отримуються швидше, а подальші потребують більше зусиль.

Алгоритм перевірки досягнень для обох типів має схожу структуру: для кожного порогового значення система порівнює поточні метрики користувача зпороговим значенням і перевіряє, чи не було це досягнення вже зафіксовано раніше. Умова existingMilestones.none { it.type == MilestoneType.STREAK_MILESTONE && it.value == milestone } забезпечує унікальність кожного досягнення та запобігає повторному нарахуванню (див. рисунок 3.2).

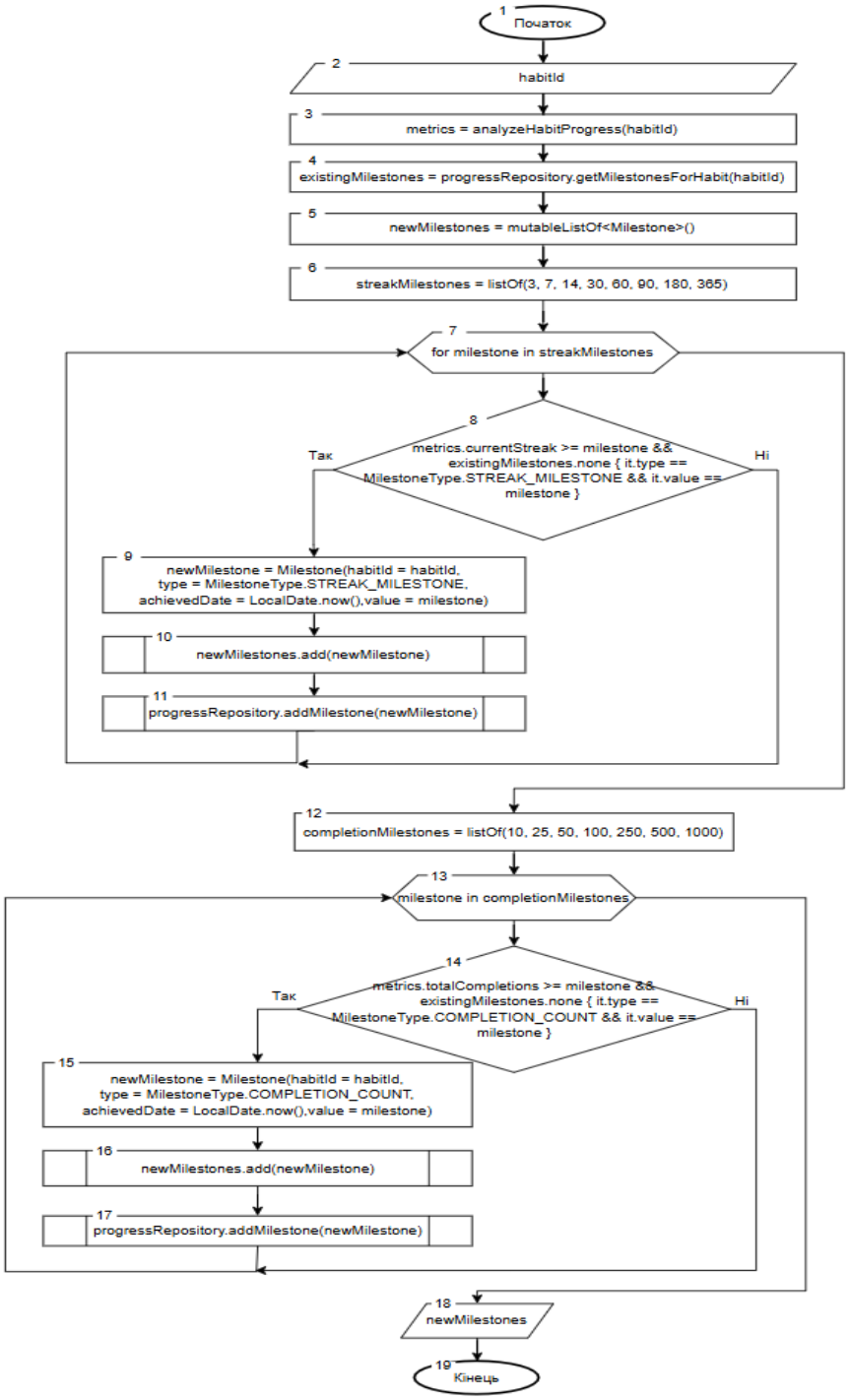


Рисунок 3.2 – Модуль перевірки та створення нових досягнень

Коли система виявляє нове досягнення, вона створює об'єкт класу Milestone, який включає ідентифікатор звички, тип досягнення, дату отримання та значення досягнення. Цей об'єкт додається до списку нових досягнень та зберігається в репозиторії прогресу за допомогою виклику progressRepository.addMilestone(newMilestone).

Архітектурно модуль досягнень інтегрований із загальною системою відстеження прогресу користувача та використовує дані, отримані від аналітичного модуля. Така інтеграція забезпечує узгодженість та актуальність даних при перевірці досягнень. Використання репозиторію для взаємодії з базою даних відповідає принципам чистої архітектури та забезпечує гнучкість при зміні або розширенні системи зберігання даних

У підсумку, розроблений модуль перевірки та створення нових досягнень є ефективним інструментом гейміфікації процесу формування звичок, що допомагає користувачам залишатися мотивованими та відстежувати свій прогрес у досягненні особистих цілей. Завдяки детально продуманій системі порогових значень та типів досягнень, модуль забезпечує регулярний позитивний зворотний зв'язок, який є ключовим елементом у формуванні та підтримці нових корисних звичок.

### 3.5 Розробка інтерфейсу користувача

Інтерфейс мобільного додатку для формування корисних звичок розроблений із дотриманням сучасних принципів UI/UX дизайну та реалізований мовою Kotlin. Він забезпечує інтуїтивно зрозумілу навігацію та задовольняє основні потреби користувачів у контексті формування та відстеження звичок. Додаток спроектовано з урахуванням психологічних аспектів формування звичок, що включає регулярність виконання, нагадування та візуальне відстеження прогресу.

Головний екран додатку представлений мінімалістичним дизайном з темним фоном, що сприяє зменшенню навантаження на зір користувача та економії заряду батареї на пристроях з OLED-дисплеями. У центральній частині екрану розміщено повідомлення українською мовою «Немає звичок. Додайте першу!» та кнопку з іконкою «+», що спонукає користувача до дії. Інтерфейс розроблений у відповідності до концепції «clean design», де кожен елемент має чітке функціональне призначення. Темна кольорова схема додатку не тільки відповідає сучасним трендам дизайну, але й створює контраст між фоном та інтерактивними

елементами, що покращує читабельність і фокусує увагу користувача на ключових компонентах інтерфейсу.

При натисканні на кнопку додавання нової звички користувач переходить на екран створення звички (див. рисунок 3.3).

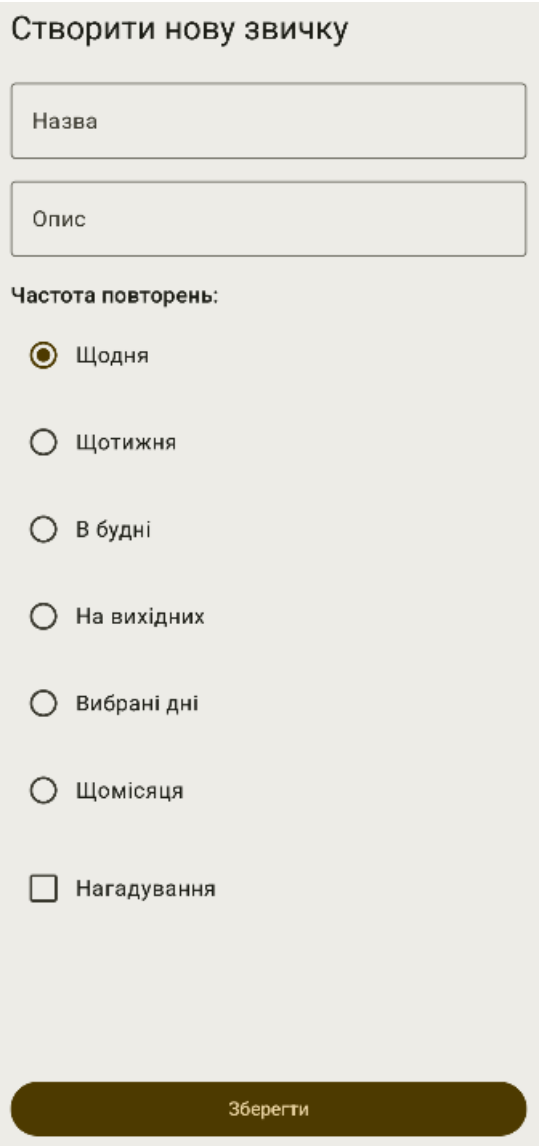


Рисунок 3.3 – Створення звички

Цей екран містить форму з полями для введення назви та опису звички, а також вибору частоти її повторення. Форма організована вертикально, що відповідає природному способу читання та взаємодії з інтерфейсом. Користувачу пропонується декілька опцій частоти повторення: щодня, щотижня, в будні, на вихідних, вибрані дні та щомісяця. Такий широкий вибір забезпечує гнучкість при

налаштуванні різних типів звичок – від щоденних ритуалів до періодичних завдань. Для кожної опції використовується радіокнопка (radio button), що дозволяє обрати лише один варіант частоти. Також передбачена можливість встановлення нагадувань за допомогою чекбоксу внизу форми. Внизу екрану розташована кнопка «Зберегти» з заокругленими краями та контрастним кольором, що підтверджує створення нової звички та забезпечує візуальний акцент на цій ключовій дії.

Всі поля введення реалізовані за допомогою компонентів TextInputLayout з бібліотеки Material Design, що забезпечує анімовані підказки, валідацію введених даних та адаптивну поведінку при фокусуванні. Такий підхід підвищує інтуїтивність інтерфейсу та покращує користувацький досвід.

При виборі опції встановлення нагадування користувачу відображається діалогове вікно вибору часу (див. рисунок 3.4), яке реалізоване за допомогою вбудованих компонентів TimePicker в Android. Інтерфейс дозволяє встановити години та хвилини за допомогою зручного селектора з кнопками навігації (стрілки вгору і вниз) та кнопками підтвердження "OK" та скасування "Скасувати". Діалогове вікно виконане в тому ж стилі, що й основний інтерфейс, забезпечуючи візуальну послідовність та цілісність дизайну. Вибір часу реалізований у 24-годинному форматі, що є стандартом для користувачів в Україні та більшості європейських країн.

Після створення звички вона відображається у списку звичок. Для кожної звички відображається її назва, опис, періодичність та час нагадування. Карточка звички виконана у формі прямокутника з закругленими кутами, що відповідає сучасним тенденціям дизайну інтерфейсів. Також присутня іконка, що візуально ідентифікує звичку, розміщена в правій частині картки. Ця іконка не лише покращує візуальне сприйняття, але й допомагає користувачу швидко ідентифікувати конкретну звичку серед інших. Чекбокс зліва дозволяє відмічати виконання звички безпосередньо зі списку, що сприяє швидкій взаємодії з додатком. Кожна картка звички є окремим інтерактивним елементом, натискання на який відкриває детальний екран конкретної звички. Такий підхід забезпечує

природну та очікувану поведінку інтерфейсу відповідно до принципів матеріального дизайну.

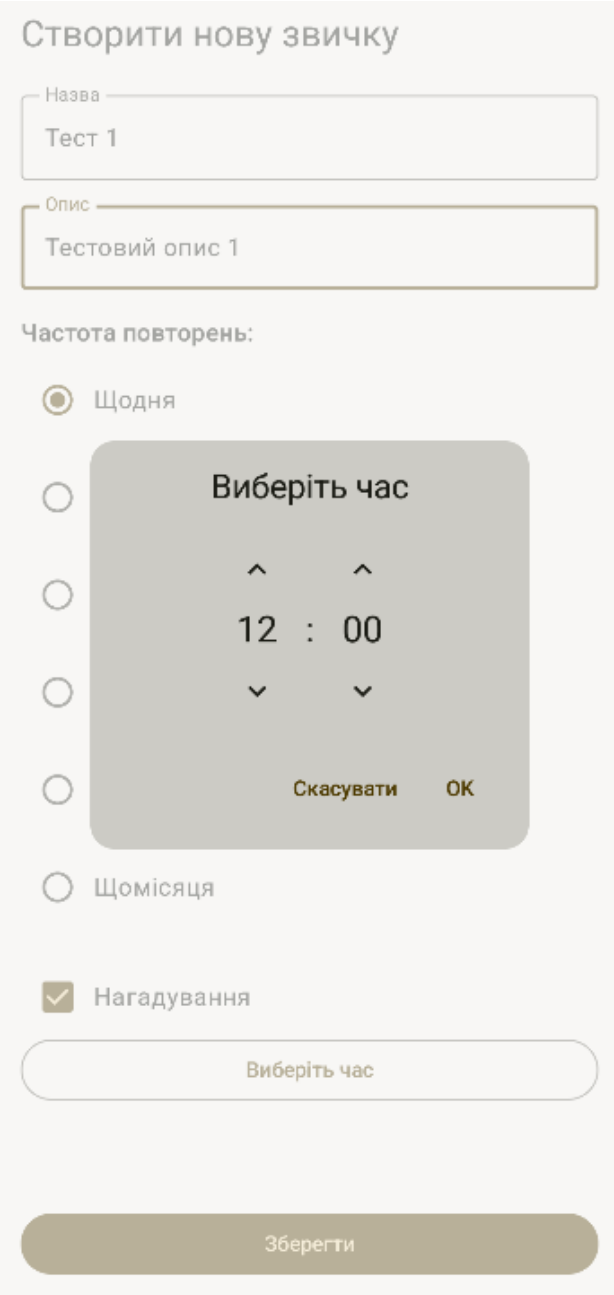


Рисунок 3.4 – Створення нагадування

На головному екрані зі списком звичок також зберігається кнопка додавання нової звички у нижньому правому куті, що забезпечує швидкий доступ до цієї функції незалежно від кількості існуючих звичок. Ця кнопка реалізована як FloatingActionButton з бібліотеки Material Design, що забезпечує ефект підняття

над основним вмістом та анімацію при взаємодії.

Екран прогресу користувача демонструє активність за останній тиждень у вигляді графіка, де кожен день тижня представлений окремим стовпчиком. Дні тижня позначені скороченими назвами (Нд, Сб, Пт, Чт, Ср, Вт, Пн), що полегшує орієнтацію в часі. Цей інтерфейс дозволяє користувачу швидко оцінити свій прогрес та регулярність виконання звичок. Верхня частина екрану містить назву «Ваш прогрес» та кнопку повернення назад у вигляді стрілки, що забезпечує зручну навігацію. Нижче заголовка розміщений блок «Прогрес за звичками», де можна бачити картку звички «Тест 1» з можливістю переходу до детальної інформації. Такий дизайн забезпечує ієрархічну структуру інформації, де користувач може рухатися від загального (прогрес за всіма звичками) до конкретного (детальна інформація про окрему звичку).

Детальний екран конкретної звички (див. рисунок 3.5) містить статистику виконання, що включає показники регулярності, виконання та загальний прогрес, представлені в числовому форматі у відсотках. Прогрес-бар під цими показниками візуалізує загальний стан виконання звички. Кругові діаграми візуалізують поточну та найдовшу серію виконання звички, відображаючи кількість днів у центрі діаграми. Такий підхід до візуалізації даних сприяє кращому розумінню прогресу та створює додаткову мотивацію для користувача.

Календар виконання займає нижню частину екрану та дозволяє відстежувати дні, в які звичка була виконана. Календар представлений у вигляді сітки чисел місяця з можливістю відзначення конкретних дат. Такий інтерфейс дозволяє користувачу бачити послідовність виконання звички та планувати майбутні дії. Дні поточного місяця відображаються в сітці, де користувач може бачити свій прогрес у контексті календарного періоду.


Рисунок 3.5 – Детальний опис звички

Внизу екрану розміщена кнопка для отримання персоналізованих рекомендацій, що підкреслює індивідуальний підхід до формування звичок. Кнопка має контрастний колір та лаконічний текст "Показати персоналізовані рекомендації", що чітко вказує на її призначення. Наявність такої функції демонструє орієнтованість додатку на індивідуальні потреби користувача та використання адаптивних алгоритмів для покращення ефективності формування звичок.

З технічної точки зору, всі екрани інтерфейсу створені з використанням компонентів Material Design, що забезпечує візуальну узгодженість та сучасний вигляд додатку. Верстка інтерфейсу реалізована за допомогою ConstraintLayout, що забезпечує адаптивність до різних розмірів екранів та орієнтацій пристрою. Для відображення списків використовується RecyclerView з кастомними адаптерами, що забезпечує ефективну роботу з даними та плавну анімацію прокрутки.

Навігація між екранами реалізована за допомогою фрагментів та

навігаційних компонентів AndroidX, що забезпечує плавні переходи між екранами та збереження стану додатку при зміні конфігурації пристрою. Для управління навігаційним стеком використовується NavController, що дозволяє реалізувати логічні зв'язки між різними екранами додатку та забезпечити коректну роботу кнопки "назад".

Локалізація інтерфейсу виконана українською мовою, що відповідає цільовій аудиторії додатку. Всі тексти винесені в ресурсний файл strings.xml, що дозволяє легко додавати підтримку інших мов у майбутньому без зміни кодової бази додатку.

Система нотифікацій реалізована за допомогою компонента NotificationManager з використанням каналів сповіщень (Notification Channels) для Android 8.0 (API рівень 26) і вище, що забезпечує відповідність сучасним вимогам операційної системи та дозволяє користувачам налаштовувати сповіщення відповідно до своїх потреб.

Розроблений інтерфейс забезпечує комплексний підхід до формування корисних звичок, поєднуючи функціональність створення та відстеження звичок з елементами гейміфікації та персоналізації, що підвищує мотивацію користувача та ефективність формування звичок. Візуальне представлення прогресу, календар виконання та статистика сприяють формуванню позитивного зворотного зв'язку та підтримують користувача на шляху до формування стійких корисних звичок.

### 3.6 Висновки

У третьому розділі викладено вичерпну інформацію щодо процесу розробки програмних модулів, необхідних для повноцінного функціонування мобільної системи. Наведено огляд сучасних технологій та інструментальних засобів, застосованих під час розробки, з обґрунтуванням їх вибору. Описано внутрішню структуру та принципи взаємодії розроблених модулів управління звичками, відстеження прогресу, аналізу даних, системи сповіщень. Наведено технічні особливості реалізації користувацького інтерфейсу з використанням мови програмування Kotlin та компонентів бібліотеки Jetpack Compose.

# 4 ТЕСТУВАННЯ ПРОГРАМИ

## 4.1 Тестування системи

Тестування мобільної системи персоналізованого формування корисних звичок проводилося з використанням тестового сценарію на пристрої з операційною системою Android. Процес тестування був спрямований на перевірку основних функціональних можливостей системи та валідацію коректності роботи користувацького інтерфейсу.

На початковому етапі тестування інтерфейс системи демонструє темний фон з інформаційним повідомленням «Немає звичок. Додайте першу!» українською мовою та кнопкою з позначкою «+» для додавання нової звички. Візуальне оформлення відповідає сучасним стандартам Material Design, що забезпечує інтуїтивно зрозумілий користувацький досвід (див. рисунок 4.1).

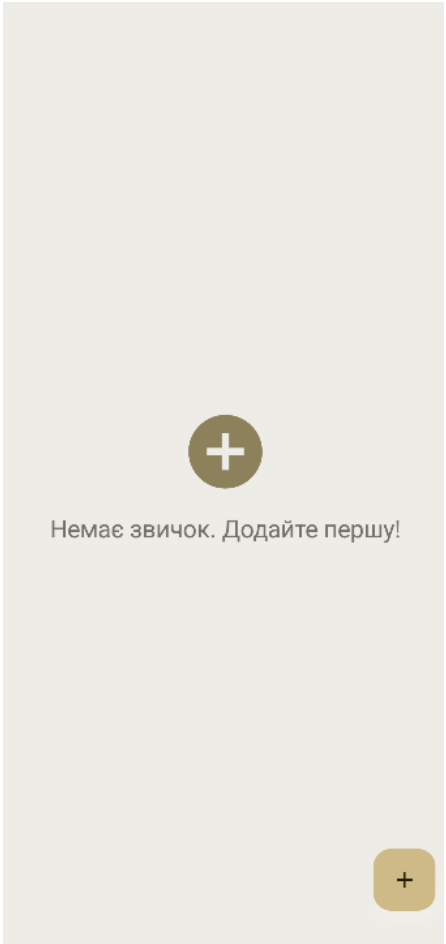


Рисунок 4.1 – Результат тестування входження до системи

При натисканні на кнопку додавання з'являється форма «Створити нову звичку», що містить поля для налаштування параметрів. Форма включає текстові поля для введення назви звички (Тест 1) та її опису (Тестовий опис 1). Користувачу надається можливість вибору частоти повторення звички з набору опцій: "Щодня", «Щотижня», «В будні», «На вихідних», «Вибрані дні» та «Щомісяця». У тестовому сценарії було обрано опцію «Щодня». Додатково система пропонує встановити нагадування, що було активовано шляхом відмітки відповідного прапорця та вибору часу нагадування (див. рисунок 4.2).

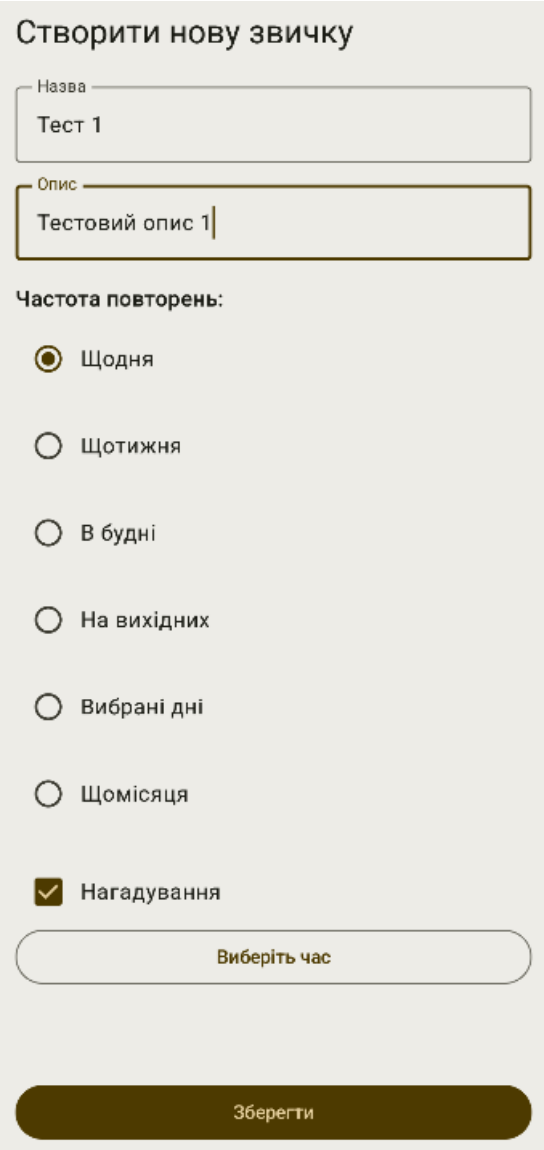


Рисунок 4.2 – Результат тестування додання нової звички

Після створення звички система відображає її у списку з відповідними

параметрами: назва «Тест 1», опис «Тестовий опис 1», частота повторення «Щодня» та встановлений час нагадування «12:00». Ця інформація представлена в компактному форматі в окремій картці з можливістю відмітки виконання звички за допомогою прапорця (див. рисунок 4.3).

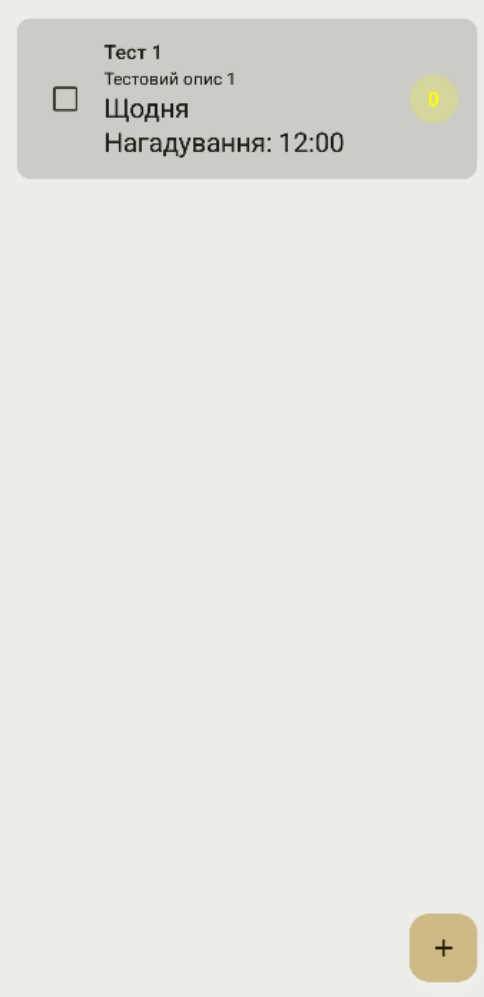


Рисунок 4.3 – Результат тестування створення нової звички

Система також надає доступ до екрану прогресу, де користувач може відстежувати свої досягнення. На цьому екрані відображається загальний прогрес за звичками, включаючи створену тестову звичку «Тест 1». Візуалізація активності за останній тиждень представлена у вигляді графіка по днях тижня (Нд, Сб, Пт, Чт, Ср, Вт, Пн) з показниками активності, що дозволяє користувачу наочно відстежувати свій прогрес (див. рисунок 4.4).

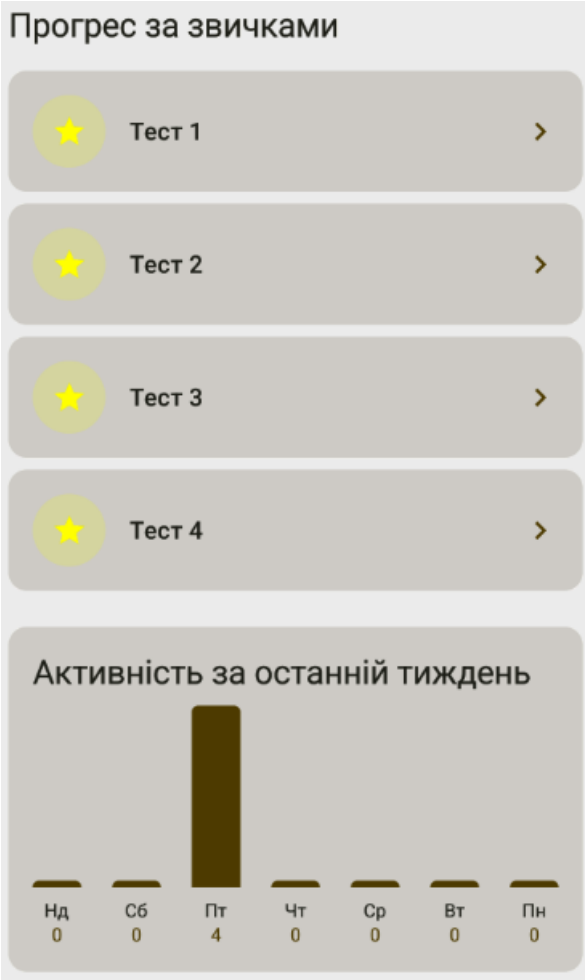


Рисунок 4.4 – Результат тестування виводу прогресу

Додатково було протестовано функцію відмітки виконання звички. При відмітці створеної звички «Тест 1» система коректно зберігає стан виконання та відображає відповідну позначку, що підтверджує працездатність механізму трекінгу прогресу (див. рисунок 4.5).

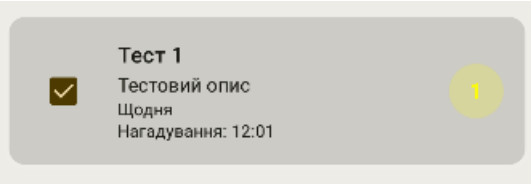


Рисунок 4.5 – Результат тестування відмітки виконання звички

Усі елементи інтерфейсу коректно відображаються та реагують на взаємодію з користувачем, що свідчить про належну імплементацію користувацького інтерфейсу з використанням мови програмування Kotlin та

стандартних компонентів Android. Локалізація інтерфейсу українською мовою реалізована коректно, всі текстові елементи відображаються без помилок кодування.

Тестування підтвердило працездатність основних функціональних модулів системи: створення нових звичок, налаштування параметрів повторення та нагадувань, відстеження прогресу та відмітка виконання. Система ефективно забезпечує персоналізоване формування корисних звичок з урахуванням користувацького прогресу.

### 4.2 Розробка інструкції користувача

Інструкція користувача включає визначення технічних вимог для запуску програмного продукту. Деталі щодо мінімальної та рекомендованої конфігурації розміщено в таблиці 4.1.

Таблиця 4.1 – Вимоги до апаратного забезпечення

| **Характеристика** | **Мінімальні вимоги** | **Рекомендовані вимоги** |
|---|---|---|
| Операційна система | Android 8.0 (Oreo) | Android 10.0 або новіше |
| Процесор | Чотириядерний, 1.5 ГГц | Восьмиядерний, 2.0 ГГц |
| Оперативна пам'ять | 2 ГБ | 4 ГБ або більше |
| Вільне місце у внутрішній пам'яті | 100 МБ | 250 МБ |
| Доступ до Інтернету | Необхідний для синхронізації даних та отримання оновлень | Постійне підключення для повноцінної роботи |
| Доступ до сповіщень | Необхідний | Необхідний |
| Доступ до датчиків | Акселерометр (для відстеження фізичної активності) | Акселерометр, гіроскоп, GPS |

Щоб почати користуватися додатком Healthy Habit App, потрібно спершу

його завантажити з Google Play Store. Після цього слід встановити додаток, дотримуючись інструкцій, що з’являються на екрані. Коли встановлення завершиться, додаток можна одразу відкрити.

Під час першого запуску на екрані з’явиться повідомлення «Немає звичок. Додайте першу!» разом із кнопкою з позначкою «+». Натиснувши її, ви почнете створення нової звички. Для цього необхідно ввести назву звички, дати їй опис і вибрати, як часто вона має повторюватись – щодня, щотижня, у будні, на вихідних, у вибрані дні або щомісяця.

Якщо бажаєте отримувати нагадування про виконання звички, потрібно активувати відповідну опцію. Після цього відкриється вікно, де можна вибрати зручний час нагадування, встановивши години та хвилини. Підтвердження вибору здійснюється кнопкою «ОК», а якщо потрібно скасувати – натискається «Скасувати». Коли вся інформація буде заповнена, слід натиснути кнопку «Зберегти», після чого нова звичка з’явиться у списку на головному екрані.

На головному екрані користувач бачить перелік усіх створених звичок. Для кожної з них видно назву, опис, частоту виконання та встановлений час нагадування. Щоб позначити, що звичка виконана, потрібно просто натиснути на чекбокс ліворуч від її назви. Так звичка буде зарахована як виконана за поточний день. Щоб додати ще одну звичку, знову слід натиснути кнопку “+” у нижньому правому куті екрана, і повторити попередні кроки. Детальнішу інформацію про звичку можна переглянути, натиснувши на її картку у списку – це відкриє окремий екран зі статистикою та аналітикою.

Аналіз власного прогресу здійснюється через спеціальний розділ під назвою «Ваш прогрес». Тут відображається активність за останній тиждень та перелік усіх звичок. Графік активності показує, скільки звичок було виконано кожного дня, де кожен стовпчик відповідає конкретному дню тижня, а його висота – кількості виконаних звичок. Обравши одну із звичок зі списку, можна побачити детальнішу статистику саме по ній.

На екрані звички користувач бачить загальні показники регулярності, рівень виконання та відсотковий прогрес, який візуалізується через прогрес-бар. Окрім

цього, кругові діаграми демонструють поточну та найдовшу серії виконань звички. У центрі кожної діаграми вказується кількість днів у серії. Календар показує, в які дні звичка була виконана, з виділенням відповідних дат у сітці поточного місяця. Якщо ж ви бажаєте отримати поради щодо покращення своїх результатів, достатньо натиснути кнопку «Показати персоналізовані рекомендації», і система проаналізує ваш прогрес та запропонує конкретні поради.

Додаток також має систему досягнень. Вона враховує серії безперервних виконань (наприклад, 3, 7, 14 днів і далі до 365) та загальну кількість виконань (від 10 до 1000). Отримані досягнення можна переглянути на екрані детального аналізу звички, і про нові досягнення користувача повідомляє система.

Для ефективного використання додатку бажано щодня заходити і відмічати виконані звички, а також аналізувати прогрес хоча б раз на тиждень. Нагадування допоможуть не забувати про важливе, тому варто встановити їх на зручний час відповідно до свого розпорядку. Починати краще з однієї-двох звичок, і лише після того, як вони увійдуть у повсякденне життя, додавати нові. Персоналізовані рекомендації варто регулярно переглядати й намагатися впроваджувати запропоновані зміни, аби покращити результати та досягти більшої ефективності.

### 4.3 Висновки

Отже, було проведено тестування створених програмних модулів системи персоналізованого формування корисних звичок з урахуванням користувацького прогресу. Було досліджено дії системи при введенні коректних та некоректних даних. Також, було створено інструкцію користувача з експлуатації програмної системи.

# ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено програмні модулі мобільної системи персоналізованого формування корисних звичок з урахуванням користувацького прогресу. Бакалаврську кваліфікаційну роботу реалізовано згідно методичних вказівок [26].

У ході виконання роботи було здійснено аналіз сучасного стану питання, розглянуто основні аналоги створюваного програмного продукту і проведено порівняння з ними, враховуючи їхні переваги та недоліки. На основі цього порівняння було сформульовано основні завдання дослідження.

Під час аналізу засобів розробки було обґрунтовано вибір мови програмування Kotlin, фреймворку Android для мобільної розробки, та відповідних технологій для реалізації персоналізованої системи формування звичок.

У першому розділі було здійснено комплексний аналіз сучасного стану систем формування звичок, детально розглянуто теоретичні та практичні аспекти цього процесу та систематизовано основні проблеми, з якими стикаються користувачі таких систем. Також проведено ґрунтовне порівняльне дослідження запропонованого рішення з існуючими на ринку платформами, виявлено суттєві конкурентні переваги та інноваційні особливості розроблюваної системи. На основі аналізу чітко сформульовано стратегічні та тактичні завдання, що визначають напрямки подальшої роботи.

У другому розділі представлено розгорнутий опис архітектурної моделі системи проєкту із застосуванням широкого спектру інструментів UML-моделювання, включаючи діаграми класів, послідовностей, компонентів та розгортання. Окрема увага приділена детальному опису ключових алгоритмів, що забезпечують ефективне функціонування мобільної системи персоналізованого формування звичок, зокрема алгоритмів обробки користувацьких даних, персоналізації рекомендацій та оптимізації циклу зворотного зв'язку.

У третьому розділі викладено вичерпну інформацію щодо процесу розробки

програмних модулів, необхідних для повноцінного функціонування мобільної системи. Наведено огляд сучасних технологій та інструментальних засобів, застосованих під час розробки, з обґрунтуванням їх вибору. Описано внутрішню структуру та принципи взаємодії розроблених модулів управління звичками, відстеження прогресу, аналізу даних, системи сповіщень. Наведено технічні особливості реалізації користувацького інтерфейсу з використанням мови програмування Kotlin та компонентів бібліотеки Jetpack Compose.

У четвертому розділі представлено результати всебічного тестування програмних модулів системи, розроблених в рамках бакалаврської кваліфікаційної роботи, із зазначенням методології тестування, тестових сценаріїв та отриманих метрик якості. Розділ також містить докладну, покрокову інструкцію користувача з ілюстраціями та практичними прикладами для ефективного використання мобільної системи персоналізованого формування корисних звичок, що забезпечує максимально зручне освоєння функціоналу користувачами різного рівня технічної підготовки.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Trend hunter [Електронний ресурс] – Режим доступу до ресурсу: https://www.trendhunter.com/trends/habitnow

2. Lally P., Gardner B. Promoting habit formation // Health Psychology Review. – 2013. – Vol. 7, Suppl. 1. – P. S137–S158.

3. Stoyanov S.R., Hides L., Kavanagh D.J., Wilson H., Tjondronegoro D., Mani M. Mobile App Rating Scale: A New Tool for Assessing the Quality of Health Mobile Apps // BMJ Open. – 2016. – Vol. 6. – P. E004585.

4. Nahum-Shani I., Smith S.N., Spring B.J., Collins L.M., Witkiewitz K., Tewari A., Murphy S.A. Just-in-time adaptive interventions (JITAIs) in mobile health: Key components and design principles for ongoing health behavior support // Health Psychology. – 2018. – Vol. 37, No. 9. – P. 1222–1235.

5. Sunyaev A., Dehling T., Taylor P.L., Mandl K.D. Availability and quality of mobile health app privacy policies // Journal of the American Medical Informatics Association. – 2015. – Vol. 22, No. e1. – P. e28–e33.

6. Wood W., Neal D.T. A new look at habits and the habit–goal interface// Psychological Review. – 2007. – Vol. 114, No. 4. – P. 843–863.

7. Гаращук Я.А. Чехместрук Р.Ю. Інтелектуальна програмна система персонального формування корисних звичок на основі динамічної оцінки користувацького прогресу. Modern Scientific Research: Theoretical and Practical Aspects April 14-16, 2025 Riga, Latvia, 2025 p. – С. 46-48.

8. Github [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/igorwojda/android-kotlin-conference-videos

9. Itvdn [Електронний ресурс] – Режим доступу до ресурсу: https://itvdn.com/ua/channel/video/kotlin

10. Habits [Електронний ресурс] – Режим доступу до ресурсу: https://habitscoach.ai/

11. Fabulous [Електронний ресурс] – Режим доступу до ресурсу: https://www.thefabulous.co/

12. Habitica [Електронний ресурс] – Режим доступу до ресурсу: https://habitica.com/

13. Coach me [Електронний ресурс] – Режим доступу до ресурсу: https://www.coach.me/

14. Raul Garcia. iOS Architecture Patterns: MVC, MVP, MVVM, VIPER, and VIP in Swift. New York: Apress, 2023. 416c.

15. Eran Boudjnah. Clean Architecture for Android: Implement Expert-led Design Patterns to Build Scalable, Maintainable, and Testable Android Apps. Noida: BPB Publications, 2022. 262c.

16. Advanced Android App Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://www.kodeco.com/books/advanced-android-app-architecture/v1.0/chapters/4-android-architecture-components

17. Hardik Trivedi. Android application development with Kotlin: Build Your First Android App In No Time. Boida: BPB Publications, 2020. 402c.

18. Mark Masse. REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces. Farnham: O'Reilly Media, 2011. 112c.

19. LiveData overview [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/topic/libraries/architecture/livedata

20. Room database [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/training/data-storage/room

21. What is an API for mobile apps? [Електронний ресурс] – Режим доступу до ресурсу: https://www.koombea.com/blog/api-for-mobile-apps/

22. Що таке прогресивні веб-додатки [Електронний ресурс] – Режим доступу до ресурсу: https://wezom.com.ua/ua/blog/pwa-prilozheniya-web-progressive-app

23. Хошаба О. М.. Методичні вказівки для виконання лабораторних робіт з дисципліни «Програмне забезпечення високопродуктивних обчислень» для студентів спеціальності 121 «Інженерія програмного забезпечення» [Електронний ресурс] – Режим доступу до ресурсу: https://iq.vntu.edu.ua/repository/card.php?id=8220

24. Jetpack Compose [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/compose

25. Методичні вказівки для самостійної роботи студентів під час виконання курсових робіт з дисципліни «Бази даних» для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад.: О. Н. Романюк, О. В. Романюк, А. В. Денисюк. – Вінниця : ВНТУ, 2022. – 71 с.

26. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О. Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 52с.

# ДОДАТКИ

Додаток А – Технічне завдання

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«25» 03 2025 року

**Технічне завдання**
**на бакалаврську кваліфікаційну роботу «Інтелектуальна програмна система персоналізованого формування корисних звичок на основі динамічної оцінки користувацького прогресу»**
**за спеціальністю**
**121 – Інженерія програмного забезпечення**

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. Чехместрук Р.Ю.
«24» 03 2025 року.

Виконала:
студентка гр. 5ПІ-21б Вознюк Я.А.
«24» 03 2025 року.

м. Вінниця – 2025 рік

**1. Найменування та галузь застосування**

Бакалаврська кваліфікаційна робота: «Інтелектуальна програмна система персоналізованого формування корисних звичок на основі динамічної оцінки користувацького прогресу».

Галузь застосування – мобільні технології.

**2. Підстава для розробки.**

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол №18 від 18 лютого 2025 р.

**3. Мета та призначення розробки.**

Метою роботи є підвищення ефективності процесу формування корисних звичок за рахунок використання спеціалізованої мобільної системи, яка дозволяє адаптуватися до індивідуального прогресу користувача.

**4. Вихідні дані для проведення НДР**

1. Eran Boudjnah. Clean Architecture for Android: Implement Expert-led Design Patterns to Build Scalable, Maintainable, and Testable Android Apps. Noida: BPB Publications, 2022. 262c.

2. Advanced Android App Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://www.kodeco.com/books/advanced-android-app-architecture/v1.0/chapters/4-android-architecture-components

3. Mark Masse. REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces. Farnham: O'Reilly Media, 2011. 112c.

4. Jetpack Compose [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/compose

**5. Технічні вимоги**

Вхідні дані — процес персоналізованого формування корисних звичок з урахуванням користувацького прогресу.

Вихідні дані — функціональна система що забезпечує можливість створення, відстеження, аналізу та оптимізації корисних звичок на основі індивідуального прогресу.

**6. Конструктивні вимоги.**

Інтерфейс програми повинен відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

–пояснювальна записка до бакалаврської кваліфікаційної роботи;

–технічне завдання;

–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 розвитку технологій системи персоналізованого формування корисних звичок з урахуванням користувацького прогресу | 25.03.25 - 05.04.25 |
| 2 | Розробка структури та алгоритмів програмного застосунку | 06.04.25 - 18.04.25 |
| 3 | Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення | 19.04.25 - 21.04.25 |
| 4 | Розробка програмних модулів персоналізованого формування корисних звичок з урахуванням користувацького прогресу | 22.04.25 - 17.05.25 |
| 5 | Тестування програми | 18.05.25 - 25.05.25 |
| 6 | Оформлення матеріалів до захисту БКР | 26.05.25 - 30.05.25 |

**10. Порядок контролю та прийняття.**

Усі етапи бакалаврської роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

Додаток Б – Протокол перевірки на наявність текстових запозичень

## ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Інтелектуальна програмна система персоналізованого формування корисних звичок на основі динамічної оцінки користувацького прогресу»

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

Підрозділ кафедра програмного забезпечення, ФІТКІ, 5ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 8,2 %

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 _____ к.т.н., доц. каф. ПЗ Чехместрук Р.Ю.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Вознюк Я.А.
(підпис) (прізвище, ініціали)

# Додаток В – Лістинг програми

```
// AppModule.kt

package com.example.healtyhabitapp

import android.app.NotificationManager
import android.content.Context
import androidx.room.Room
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent
import javax.inject.Singleton

// AppModule.kt
@Module
@InstallIn(SingletonComponent::class)
object AppModule {

    @Provides
    @Singleton
    fun provideHabitDatabase(@ApplicationContext context: Context): HabitDatabase {
        return Room.databaseBuilder(
            context,
            HabitDatabase::class.java,
            "habit_database"
        )
            .fallbackToDestructiveMigration()
            .build()
    }

    @Provides
    @Singleton
    fun provideHabitDao(database: HabitDatabase): HabitDao {
        return database.habitDao()
    }

    @Provides
    @Singleton
    fun provideHabitRepository(
        habitDao: HabitDao
    ): HabitRepository {
        return HabitRepositoryImpl(habitDao)
    }

    @Provides
    @Singleton
    fun provideHabitReminderManager(@ApplicationContext context: Context):
```

```
HabitReminderManager {
        return HabitReminderManager(context)
    }

    @Provides
    fun provideNotificationManager(@ApplicationContext context: Context): NotificationManager {
        return context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
    }

    @Provides
    @Singleton
    fun provideProgressRepository(database: HabitDatabase): ProgressRepository {
        return ProgressRepositoryImpl(database)
    }

    // Також додайте провайдер для ProgressAnalyzer
    @Provides
    @Singleton
    fun provideProgressAnalyzer(
        habitRepository: HabitRepository,
        progressRepository: ProgressRepository
    ): ProgressAnalyzer {
        return ProgressAnalyzer(habitRepository, progressRepository)
    }

    @Provides
    @Singleton
    fun provideNavigationManager(): NavigationManager {
        return NavigationManager()
    }
}


// Converters.kt

package com.example.healtyhabitapp

import android.os.Build
import androidx.annotation.RequiresApi
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.toArgb
import androidx.room.TypeConverter
import org.json.JSONObject
import java.time.LocalDate
import java.time.LocalTime
import java.time.format.DateTimeFormatter

// Converters.kt
class Converters {
    @TypeConverter
    fun fromTimestamp(value: Long?): LocalDate? {
        return value?.let { LocalDate.ofEpochDay(it) }
```

```
}

@TypeConverter
fun dateToTimestamp(date: LocalDate?): Long? {
    return date?.toEpochDay()
}

@RequiresApi(Build.VERSION_CODES.O)
@TypeConverter
fun fromLocalTimeToString(time: LocalTime?): String? {
    return time?.format(DateTimeFormatter.ISO_LOCAL_TIME)
}

@RequiresApi(Build.VERSION_CODES.O)
@TypeConverter
fun fromStringToLocalTime(timeString: String?): LocalTime? {
    return timeString?.let { LocalTime.parse(it , DateTimeFormatter.ISO_LOCAL_TIME) }
}

@TypeConverter
fun fromFrequency(frequency: Frequency): String {
    return frequency.name
}

@TypeConverter
fun toFrequency(name: String): Frequency {
    return Frequency.valueOf(name)
}

@TypeConverter
fun fromColor(color: Color): Int {
    return color.toArgb()
}

@TypeConverter
fun toColor(value: Int): Color {
    return Color(value)
}

@RequiresApi(Build.VERSION_CODES.O)
@TypeConverter
fun fromLocalDateList(dates: List<LocalDate>?): String? {
    return dates?.joinToString(",") { it.toEpochDay().toString() }
}

@RequiresApi(Build.VERSION_CODES.O)
@TypeConverter
fun toLocalDateList(data: String?): List<LocalDate>? {
    return data?.split(",")?.map {
        LocalDate.ofEpochDay(it.toLong())
    }
}
```

```
    @TypeConverter
    fun fromMilestoneType(type: MilestoneType): String {
        return type.name
    }

    @TypeConverter
    fun toMilestoneType(name: String): MilestoneType {
        return MilestoneType.valueOf(name)
    }

    @TypeConverter
    fun fromMilestone(milestone: Milestone): String {
        val json = JSONObject()
        json.put("habitId", milestone.habitId)
        json.put("type", milestone.type.name)
        json.put("achievedDate", milestone.achievedDate.toEpochDay())
        json.put("value", milestone.value)
        return json.toString()
    }

    @TypeConverter
    fun toMilestone(json: String): Milestone {
        val jsonObject = JSONObject(json)
        return Milestone(
            habitId = jsonObject.getLong("habitId"),
            type = MilestoneType.valueOf(jsonObject.getString("type")),
            achievedDate = LocalDate.ofEpochDay(jsonObject.getLong("achievedDate")),
            value = jsonObject.getInt("value")
        )
    }
}


// CreateHabitScreen.kt

package com.example.healtyhabitapp

import android.os.Build
import androidx.annotation.RequiresApi
import androidx.compose.animation.AnimatedVisibility
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
```

```
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.KeyboardArrowDown
import androidx.compose.material.icons.filled.KeyboardArrowUp
import androidx.compose.material3.Button
import androidx.compose.material3.Card
import androidx.compose.material3.Checkbox
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedButton
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.RadioButton
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.compose.ui.window.Dialog
import androidx.hilt.navigation.compose.hiltViewModel
import java.time.DayOfWeek
import java.time.LocalTime

// CreateHabitScreen.kt
@RequiresApi(Build.VERSION_CODES.O)
@Composable
fun CreateHabitScreen(
) {
    var title by remember { mutableStateOf("") }
    var description by remember { mutableStateOf("") }
    var frequency by remember { mutableStateOf(Frequency.DAILY) }
    var timeOfDay by remember { mutableStateOf<LocalTime?>(null) }
    var reminderEnabled by remember { mutableStateOf(false) }
    var selectedDays by remember { mutableStateOf(emptySet<DayOfWeek>()) }
    val habitViewModel: HabitViewModel = hiltViewModel()

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(16.dp)
    ) {
        Text(
            text = "Створити нову звичку",
```

```
    style = MaterialTheme.typography.headlineSmall,
    modifier = Modifier.padding(bottom = 16.dp)
)

OutlinedTextField(
    value = title,
    onValueChange = { title = it },
    label = { Text("Назва") },
    modifier = Modifier.fillMaxWidth()
)

Spacer(modifier = Modifier.height(8.dp))

OutlinedTextField(
    value = description,
    onValueChange = { description = it },
    label = { Text("Опис") },
    modifier = Modifier.fillMaxWidth()
)

Spacer(modifier = Modifier.height(16.dp))

Text("Частота повторень:", style = MaterialTheme.typography.titleMedium)

FrequencySelector(
    selectedFrequency = frequency,
    onFrequencySelected = { frequency = it }
)

if (frequency == Frequency.SPECIFIC_DAYS) {
    WeekDaysSelector(
        selectedDays = selectedDays,
        onDayToggled = { day, selected ->
            selectedDays = if (selected) {
                selectedDays + day
            } else {
                selectedDays - day
            }
        }
    )
}

Spacer(modifier = Modifier.height(16.dp))

Row(
    verticalAlignment = Alignment.CenterVertically,
    modifier = Modifier.fillMaxWidth()
) {
    Checkbox(
        checked = reminderEnabled,
        onCheckedChange = { reminderEnabled = it }
    )
```

```
            Text("Нагадування")
        }

        AnimatedVisibility(visible = reminderEnabled) {
            TimePickerButton(
                selectedTime = timeOfDay,
                onTimeSelected = { timeOfDay = it }
            )
        }

        Spacer(modifier = Modifier.weight(1f))

        Button(
            onClick = {
                if (title.isNotBlank()) {
                    val habit = Habit(
                        title = title,
                        description = description,
                        frequency = frequency,
                        timeOfDay = timeOfDay,
                        reminderEnabled = reminderEnabled && timeOfDay != null
                    )
                    habitViewModel.addHabit(habit)
                    habitViewModel.navigateBack()
                }
            },
            modifier = Modifier.fillMaxWidth()
        ) {
            Text("Зберегти")
        }
    }
}

@Composable
fun FrequencySelector(
    selectedFrequency: Frequency,
    onFrequencySelected: (Frequency) -> Unit
) {
    Column {
        Frequency.values().forEach { frequency ->
            Row(
                verticalAlignment = Alignment.CenterVertically,
                modifier = Modifier
                    .fillMaxWidth()
                    .clickable { onFrequencySelected(frequency) }
                    .padding(vertical = 8.dp)
            ) {
                RadioButton(
                    selected = selectedFrequency == frequency,
                    onClick = { onFrequencySelected(frequency) }
                )
                Text(
```

```
                text = when (frequency) {
                    Frequency.DAILY -> "Щодня"
                    Frequency.WEEKLY -> "Щотижня"
                    Frequency.WEEKDAYS -> "В будні"
                    Frequency.WEEKENDS -> "На вихідних"
                    Frequency.SPECIFIC_DAYS -> "Вибрані дні"
                    Frequency.MONTHLY -> "Щомісяця"
                }
            )
        }
    }
}

@RequiresApi(Build.VERSION_CODES.O)
@Composable
fun WeekDaysSelector(
    selectedDays: Set<DayOfWeek> ,
    onDayToggled: (DayOfWeek , Boolean) -> Unit
) {
    val daysOfWeek = listOf(
        DayOfWeek.MONDAY to "Пн",
        DayOfWeek.TUESDAY to "Вт",
        DayOfWeek.WEDNESDAY to "Ср",
        DayOfWeek.THURSDAY to "Чт",
        DayOfWeek.FRIDAY to "Пт",
        DayOfWeek.SATURDAY to "Сб",
        DayOfWeek.SUNDAY to "Нд"
    )

    Row(
        modifier = Modifier
            .fillMaxWidth()
            .padding(vertical = 8.dp),
        horizontalArrangement = Arrangement.SpaceBetween
    ) {
        daysOfWeek.forEach { (day, label) ->
            DayToggleButton(
                day = day,
                label = label,
                selected = day in selectedDays,
                onToggle = { selected -> onDayToggled(day, selected) }
            )
        }
    }
}

@Composable
fun DayToggleButton(
    day: DayOfWeek ,
    label: String ,
    selected: Boolean ,
```

```
    onToggle: (Boolean) -> Unit
) {
    val backgroundColor = if (selected) {
        MaterialTheme.colorScheme.primary
    } else {
        MaterialTheme.colorScheme.surface
    }

    val textColor = if (selected) {
        MaterialTheme.colorScheme.onPrimary
    } else {
        MaterialTheme.colorScheme.onSurface
    }

    Box(
        modifier = Modifier
            .size(40.dp)
            .clip(CircleShape)
            .background(backgroundColor)
            .clickable { onToggle(!selected) },
        contentAlignment = Alignment.Center
    ) {
        Text(
            text = label,
            color = textColor,
            fontSize = 14.sp
        )
    }
}

@RequiresApi(Build.VERSION_CODES.O)
@Composable
fun TimePickerButton(
    selectedTime: LocalTime? ,
    onTimeSelected: (LocalTime) -> Unit
) {
    var showDialog by remember { mutableStateOf(false) }

    val timeText = selectedTime?.let {
        String.format("%02d:%02d", it.hour, it.minute)
    } ?: "Виберіть час"

    OutlinedButton(
        onClick = { showDialog = true },
        modifier = Modifier.fillMaxWidth()
    ) {
        Text(timeText)
    }

    if (showDialog) {
        TimePickerDialog(
            onDismiss = { showDialog = false },
```

```
            onTimeSelected = {
                onTimeSelected(it)
                showDialog = false
            },
            initialTime = selectedTime ?: LocalTime.of(12, 0)
        )
    }
}

@RequiresApi(Build.VERSION_CODES.O)
@Composable
fun TimePickerDialog(
    onDismiss: () -> Unit ,
    onTimeSelected: (LocalTime) -> Unit ,
    initialTime: LocalTime
) {
    var hour by remember { mutableStateOf(initialTime.hour) }
    var minute by remember { mutableStateOf(initialTime.minute) }

    Dialog(onDismissRequest = onDismiss) {
        Card(
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp),
            shape = RoundedCornerShape(16.dp)
        ) {
            Column(
                modifier = Modifier.padding(16.dp),
                horizontalAlignment = Alignment.CenterHorizontally
            ) {
                Text(
                    text = "Виберіть час",
                    style = MaterialTheme.typography.titleLarge
                )

                Spacer(modifier = Modifier.height(16.dp))

                Row(
                    horizontalArrangement = Arrangement.Center,
                    verticalAlignment = Alignment.CenterVertically,
                    modifier = Modifier.fillMaxWidth()
                ) {
                    // Hour picker
                    NumberPicker(
                        value = hour,
                        onValueChange = { hour = it },
                        range = 0..23
                    )

                    Text(
                        text = ":",
                        style = MaterialTheme.typography.titleLarge,
```

```
                modifier = Modifier.padding(horizontal = 8.dp)
            )

            // Minute picker
            NumberPicker(
                value = minute,
                onValueChange = { minute = it },
                range = 0..59
            )
        }

        Spacer(modifier = Modifier.height(16.dp))

        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.End
        ) {
            TextButton(onClick = onDismiss) {
                Text("Скасувати")
            }
            TextButton(
                onClick = {
                    onTimeSelected(LocalTime.of(hour, minute))
                }
            ) {
                Text("OK")
            }
        }
    }
  }
}

@Composable
fun NumberPicker(
    value: Int,
    onValueChange: (Int) -> Unit,
    range: IntRange
) {
    Column(
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        IconButton(
            onClick = {
                if (value < range.last) onValueChange(value + 1)
                else onValueChange(range.first)
            }
        ) {
            Icon(Icons.Default.KeyboardArrowUp, contentDescription = "Збільшити")
        }

        Text(
```

```
            text = String.format("%02d", value),
            style = MaterialTheme.typography.headlineSmall
        )

        IconButton(
            onClick = {
                if (value > range.first) onValueChange(value - 1)
                else onValueChange(range.last)
            }
        ) {
            Icon(Icons.Default.KeyboardArrowDown, contentDescription = "Зменшити")
        }
    }
}


// EditHabitScreen.kt

package com.example.healtyhabitapp

import androidx.compose.animation.AnimatedVisibility
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.ArrowBack
import androidx.compose.material.icons.filled.BarChart
import androidx.compose.material3.Button
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import java.time.DayOfWeek
import androidx.compose.animation.AnimatedVisibility
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.material.icons.filled.ArrowBack
```

```
import androidx.compose.material.icons.filled.BarChart
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Checkbox
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun EditHabitScreen(habitId: Long) {
    val viewModel:HabitViewModel = hiltViewModel()
    val habits by viewModel.habits.collectAsState()
    val habit = habits.find { it.id == habitId }

    if (habit == null) {
        LaunchedEffect(Unit) {
            viewModel.navigateBack()
        }
        return
    }

    var title by remember { mutableStateOf(habit.title) }
    var description by remember { mutableStateOf(habit.description) }
    var frequency by remember { mutableStateOf(habit.frequency) }
    var timeOfDay by remember { mutableStateOf(habit.timeOfDay) }
    var reminderEnabled by remember { mutableStateOf(habit.reminderEnabled) }
    var selectedDays by remember { mutableStateOf(emptySet<DayOfWeek>()) }

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text("Редагувати звичку") },
                navigationIcon = {
                    IconButton(onClick = { viewModel.navigateBack() }) {
                        Icon(Icons.Default.ArrowBack, contentDescription = "Назад")
                    }
                },
                actions = {
                    IconButton(onClick = { viewModel.navigateToHabitProgress(habitId) }) {
                        Icon(Icons.Default.BarChart, contentDescription = "Переглянути прогрес")
```

```
                }
            }
        )
    }
) { paddingValues ->
    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(paddingValues)
            .padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        OutlinedTextField(
            value = title,
            onValueChange = { title = it },
            label = { Text("Назва") },
            modifier = Modifier.fillMaxWidth()
        )

        OutlinedTextField(
            value = description,
            onValueChange = { description = it },
            label = { Text("Опис") },
            modifier = Modifier.fillMaxWidth()
        )

        Text("Частота повторень:", style = MaterialTheme.typography.titleMedium)

        FrequencySelector(
            selectedFrequency = frequency,
            onFrequencySelected = { frequency = it }
        )

        if (frequency == Frequency.SPECIFIC_DAYS) {
            WeekDaysSelector(
                selectedDays = selectedDays,
                onDayToggled = { day, selected ->
                    selectedDays = if (selected) {
                        selectedDays + day
                    } else {
                        selectedDays - day
                    }
                }
            )
        }

        Row(
            verticalAlignment = Alignment.CenterVertically,
            modifier = Modifier.fillMaxWidth()
        ) {
            Checkbox(
                checked = reminderEnabled,
```

```
                onCheckedChange = { reminderEnabled = it }
            )
            Text("Нагадування")
        }

        AnimatedVisibility(visible = reminderEnabled) {
            TimePickerButton(
                selectedTime = timeOfDay,
                onTimeSelected = { timeOfDay = it }
            )
        }

        Spacer(modifier = Modifier.weight(1f))

        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.spacedBy(8.dp)
        ) {
            Button(
                onClick = { viewModel.navigateBack() },
                modifier = Modifier.weight(1f)
            ) {
                Text("Скасувати")
            }

            Button(
                onClick = {
                    if (title.isNotBlank()) {
                        val updatedHabit = habit.copy(
                            title = title,
                            description = description,
                            frequency = frequency,
                            timeOfDay = timeOfDay,
                            reminderEnabled = reminderEnabled && timeOfDay != null
                        )
                        viewModel.updateHabit(updatedHabit)
                        viewModel.navigateBack()
                    }
                },
                modifier = Modifier.weight(1f)
            ) {
                Text("Оновити")
            }
        }
    }
}
```

```
// Habit.kt

package com.example.healtyhabitapp
```

```
import android.os.Build
import androidx.annotation.RequiresApi
import androidx.compose.ui.graphics.Color
import java.time.DayOfWeek
import java.time.LocalDate
import java.time.LocalTime

// Habit.kt
data class Habit @RequiresApi(Build.VERSION_CODES.O) constructor(
    val id: Long = 0 ,
    val title: String ,
    val description: String = "" ,
    val frequency: Frequency ,
    val timeOfDay: LocalTime? = null ,
    val reminderEnabled: Boolean = false ,
    val startDate: LocalDate = LocalDate.now() ,
    val color: Color = Color.Blue ,
    val completedDates: List<LocalDate> = emptyList()
)

enum class Frequency {
    DAILY,
    WEEKLY,
    WEEKDAYS,
    WEEKENDS,
    SPECIFIC_DAYS,
    MONTHLY
}

data class SpecificDays(val days: Set<DayOfWeek>)


// HabitDao.kt

package com.example.healtyhabitapp

import androidx.room.Dao
import androidx.room.Delete
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import androidx.room.Transaction
import androidx.room.Update
import kotlinx.coroutines.flow.Flow

@Dao
interface HabitDao {
    @Query("SELECT * FROM habits ORDER BY title")
    fun getAllHabits(): Flow<List<HabitEntity>>

    @Query("SELECT * FROM habits WHERE id = :id")
```

```
    suspend fun getHabitById(id: Long): HabitEntity?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertHabit(habit: HabitEntity): Long

    @Update
    suspend fun updateHabit(habit: HabitEntity)

    @Delete
    suspend fun deleteHabit(habit: HabitEntity)

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertCompletion(completion: CompletionEntity)

    @Query("SELECT * FROM completions WHERE habitId = :habitId")
    fun getCompletionsForHabit(habitId: Long): Flow<List<CompletionEntity>>

    @Transaction
    @Query("SELECT * FROM habits WHERE id = :habitId")
    suspend fun getHabitWithCompletions(habitId: Long): HabitWithCompletions?

    @Transaction
    @Query("SELECT * FROM habits")
    suspend fun getAllHabitsWithCompletions(): List<HabitWithCompletions>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertMilestone(milestone: MilestoneEntity): Long

    @Query("SELECT * FROM milestones WHERE habitId = :habitId")
    suspend fun getMilestonesForHabit(habitId: Long): List<MilestoneEntity>

    @Query("SELECT * FROM milestones")
    suspend fun getAllMilestones(): List<MilestoneEntity>

    @Query("SELECT * FROM completions")
    suspend fun getAllCompletions(): List<CompletionEntity>
}


//

package com.example.healtyhabitapp

import androidx.compose.ui.graphics.Color
import androidx.room.Embedded
import androidx.room.Entity
import androidx.room.PrimaryKey
import androidx.room.Relation
import kotlinx.coroutines.flow.first
import kotlinx.coroutines.flow.map
import java.time.DayOfWeek
import java.time.LocalDate
```

```
import java.time.LocalTime
import java.time.temporal.ChronoUnit
import javax.inject.Inject
import javax.inject.Singleton

interface ProgressRepository {
    suspend fun getProgressMetricsForHabit(habitId: Long): ProgressMetrics?
    suspend fun updateProgressMetrics(metrics: ProgressMetrics)
    suspend fun getUserProgressSummary(): UserProgressSummary
    suspend fun addMilestone(milestone: Milestone)
    suspend fun getMilestonesForHabit(habitId: Long): List<Milestone>
    suspend fun getHabitById(id: Long): Habit?
}

@Singleton
class ProgressRepositoryImpl @Inject constructor(
    private val database: HabitDatabase
) : ProgressRepository {

    // Кешування результатів для оптимізації продуктивності
    private val metricsCache = mutableMapOf<Long, ProgressMetrics>()
    private var userProgressCache: UserProgressSummary? = null
    private val milestonesCache = mutableMapOf<Long, List<Milestone>>()

    override suspend fun getProgressMetricsForHabit(habitId: Long): ProgressMetrics? {
        // Перевірка кешу
        metricsCache[habitId]?.let {
            return it
        }

        // Отримання даних з бази даних
        val habitWithCompletions = database.habitDao().getHabitWithCompletions(habitId) ?: return null
        val habit = habitWithCompletions.habit
        val completions = habitWithCompletions.completions.map { it.completionDate }
```

Додаток Г – Ілюстративна частина

# ІЛЮСТРАТИВНА ЧАСТИНА

ІНТЕЛЕКТУАЛЬНА ПРОГРАМНА СИСТЕМА ПЕРСОНАЛІЗОВАНОГО ФОРМУВАННЯ КОРИСНИХ ЗВИЧОК НА ОСНОВІ ДИНАМІЧНОЇ ОЦІНКИ КОРИСТУВАЦЬКОГО ПРОГРЕСУ

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота на тему:
«Інтелектуальна програмна система персоналізованого формування корисних звичок на основі динамічної оцінки користувацького прогресу»

Автор: ст.групи 5ПІ-21б Вознюк Я.А.
Науковий керівник: к.т.н., доцент каф. ПЗ Чехместрук Р.Ю.

Вінниця - 2025

Рисунок Г.1 — Титульний слайд

# Актуальність теми

Сучасні дослідження у галузі поведінкової психології та машинного навчання демонструють потенціал створення адаптивних систем, здатних аналізувати паттерни поведінки користувача, прогнозувати ймовірність виконання завдань та динамічно коригувати стратегії мотивації. Інтеграція методів штучного інтелекту з принципами формування звичок може забезпечити значно вищу ефективність порівняно з існуючими рішеннями. Зокрема, використання алгоритмів машинного навчання дозволяє виявляти складні залежності між різними факторами, що впливають на поведінку користувача, та створювати персоналізовані рекомендації у реальному часі.

Особливого значення набуває можливість динамічної оцінки прогресу користувача, яка передбачає не лише фіксацію факту виконання або невиконання завдань, але й аналіз якості виконання, емоційного стану користувача, зовнішніх обставин та інших контекстуальних даних. Така комплексна оцінка дозволяє системі краще розуміти індивідуальні потреби користувача та адаптувати свою роботу відповідно до його поточного стану і цілей. Водночас важливим є забезпечення приватності та безпеки персональних даних користувачів, що вимагає розробки спеціальних підходів до збору, обробки та зберігання інформації.

Враховуючи недоліки наявних систем, потенціал сучасних технологій для створення по-справжньому персоналізованого досвіду формування звичок, а також зростаючу потребу людей у ефективних інструментах самовдосконалення, актуальною є розробка власної системи.

Рисунок Г.2 — Актуальність теми

# Мета, об’єкт та предмет дослідження

**Мета та завдання дослідження.** Метою бакалаврської кваліфікаційної роботи є підвищення ефективності процесу формування корисних звичок за рахунок використання спеціалізованої мобільної системи, яка дозволяє адаптуватися до індивідуального прогресу користувача.

**Об'єктом дослідження** є процес розробки мобільної системи персоналізованого формування корисних звичок.

**Предметом дослідження** є методи та засоби розробки мобільної системи персоналізованого формування корисних звичок з використанням мови програмування Kotlin та архітектури MVVM.

Рисунок Г.3 — Мета, об’єкт та предмет дослідження

# Завдання дослідження

Основними задачами дослідження є:

- розробити модуль управління звичками, що забезпечуватиме створення, редагування, видалення та категоризацію користувацьких звичок з можливістю встановлення періодичності, нагадувань та цілей;
- розробити модуль відстеження прогресу, який забезпечить збір даних про виконання звичок, візуалізацію прогресу у вигляді графіків та діаграм, а також аналіз успішності формування звичок;
- розробити модуль аналізу користувацьких даних, що дозволятиме проводити глибокий аналіз поведінкових патернів, визначати оптимальний час для виконання звичок та адаптувати систему до індивідуальних особливостей користувача;
- розробити модуль персоналізованих сповіщень, який забезпечить своєчасне нагадування про заплановані звички з урахуванням режиму дня користувача та аналізу найбільш ефективного часу взаємодії;
- розробити модуль статистики та звітності, який забезпечить генерацію детальних звітів про прогрес формування звичок, виявлення патернів поведінки та надання персоналізованих рекомендацій;
- розробити модуль адаптивного користувацького інтерфейсу з використанням Jetpack Compose, який забезпечить зручне відображення контенту та інтуїтивно зрозумілу навігацію;
- провести тестування розроблених модулів на відповідність функціональним вимогам, продуктивність та користувацький досвід.

Рисунок Г.4 — Завдання дослідження

# Наукова новизна

1. Подальшого розвитку отримав метод динамічної персоналізованої оцінки прогресу, який, на відміну від відомих, аналізує індивідуальні патерни поведінки користувача для генерації адаптивних рекомендацій на основі реальних даних використання, враховуючи оптимальний час виконання, фактори що заважають регулярності, та взаємозв'язки між різними звичками, що забезпечує підвищення ефективності формування корисних звичок завдяки точному налаштуванню під особливості кожного користувача.
2.Подальшого розвитку отримав метод інтелектуального аналізу консистентності звичок, який, на відміну від відомих, застосовує комплексну модель для оцінки стабільності звички, враховуючи не лише факт виконання, але й регулярність інтервалів, нещодавні тенденції та довгострокову стабільність відповідно до запланованої частоти, що дозволяє користувачам отримувати об'єктивну оцінку прогресу у формуванні стійких звичок.

Рисунок Г.5 — Наукова новизна

# Практична цінність отриманих результатів

Практична цінність одержаних результатів полягає у можливості використання створеної мобільної системи, що покриватиме різні аспекти формування корисних звичок з урахуванням індивідуального прогресу користувача.

Рисунок Г.6 — Практична цінність отриманих результатів

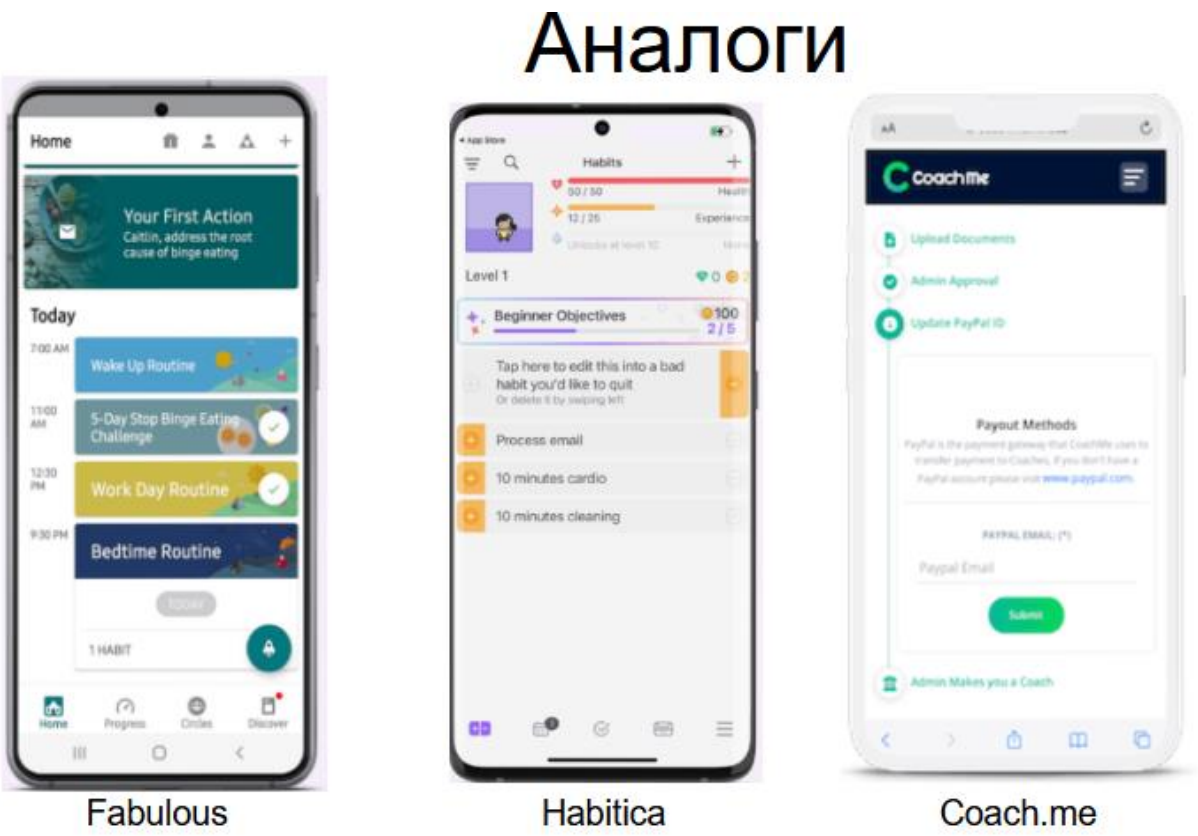


Рисунок Г.7 — Аналоги

# Порівняльний аналіз аналогів

| Функціональна характеристика | Fabulous | Habitica | Coach.me | Власна розробка |
|---|---|---|---|---|
| Трекер звичок | + | + | + | + |
| Статистика та графіки | + | + | + | + |
| Підрахунок серій виконання звички | + | + | + | + |
| Нагадування | + | + | + | + |
| Щоденні поради | + | - | + | + |
| Персоналізована оцінка прогресу | - | - | - | + |
| Перевірка закріплення звички | - | - | - | + |
| Сумарний бал | 5 | 4 | 5 | 7 |

Рисунок Г.8 — Порівняльний аналіз аналогів

## Використані технології

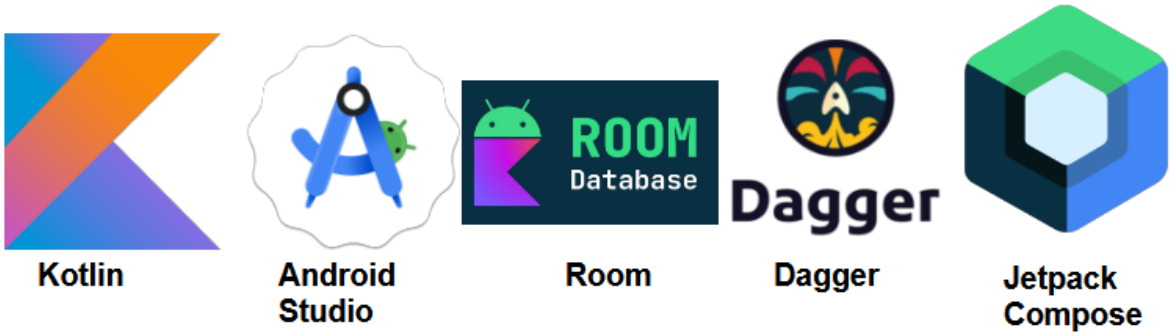


Рисунок Г.9 — Використані технології

## Розробка методу динамічної персоналізованої оцінки прогресу

1. Система отримує дані про виконання звички через інтерфейс користувача.
2. Дані зберігаються в таблиці Completions, фіксуючи факт та час виконання.
3. При запиті персоналізованих рекомендацій викликається метод generatePersonalizedSuggestions().
4. Метод аналізує всі записи виконання для заданої звички та обчислює часові патерни.
5. Система визначає оптимальні часові вікна на основі успішних виконань.
6. Проводиться аналіз пропущених виконань для виявлення типових перешкод.
7. На основі аналізу формуються конкретні рекомендації, адаптовані під користувача.
8. Рекомендації зберігаються у системі та відображаються користувачу в зрозумілому форматі.
9. Система продовжує моніторинг виконання для подальшого уточнення рекомендацій.

Рисунок Г.10 — Розробка методу динамічної персоналізованої оцінки прогресу

## Розробка методу інтелектуального аналізу консистентності звичок

1. Система отримує список всіх виконань звички з бази даних.
2. Метод calculateConsistencyScore() обчислює очікувані дні виконання на основі частоти звички.
3. Розраховується базовий відсоток виконання як відношення кількості фактичних виконань до очікуваних.
4. Система обчислює інтервали між послідовними виконаннями.
5. На основі інтервалів розраховується середнє значення та стандартне відхилення.
6. Обчислюється коефіцієнт варіації як міра стабільності інтервалів.
7. Система аналізує нещодавні тенденції, надаючи більшу вагу останнім виконанням.
8. Результати трьох факторів (базовий відсоток, стабільність інтервалів, нещодавні тенденції) зважуються для отримання фінальної оцінки консистентності.
9. Фінальна оцінка зберігається в таблиці Progress Metrics та відображається користувачу.

Рисунок Г.11 — Розробка методу інтелектуального аналізу консистентності звичок

Розробка архітектури застосунку

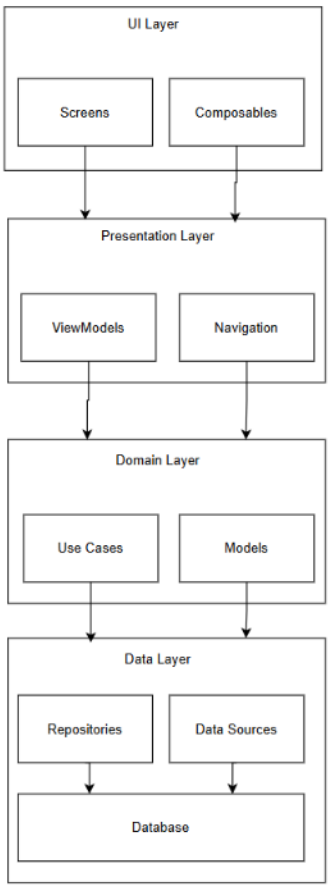


Рисунок Г.12 — Розробка архітектури застосунку

Розробка моделі системи

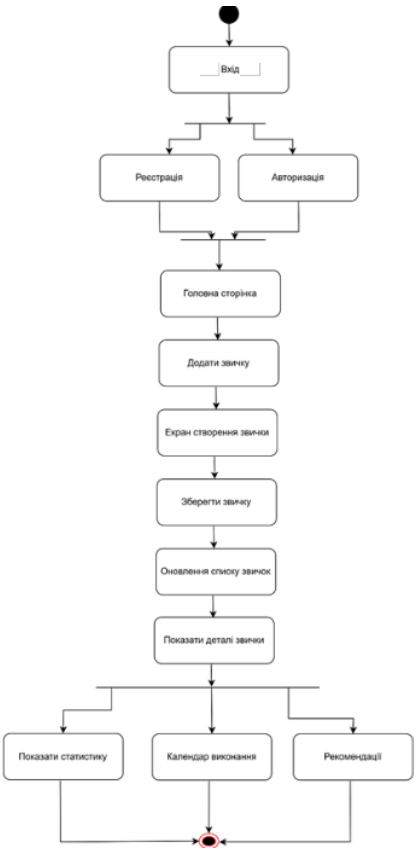


Рисунок Г.13 — Розробка моделі системи

# База даних системи


Рисунок Г.14 — База даних системи

# Тестування системи

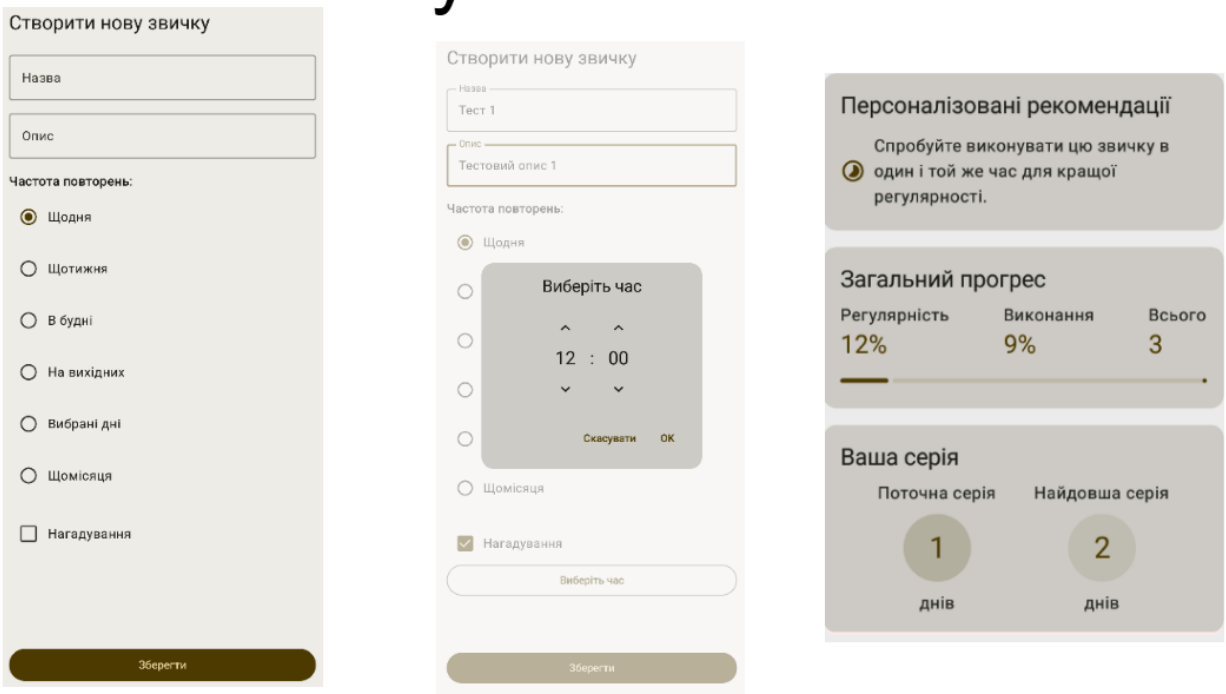


Рисунок Г.15 — Тестування системи (частина 1)

# Тестування системи

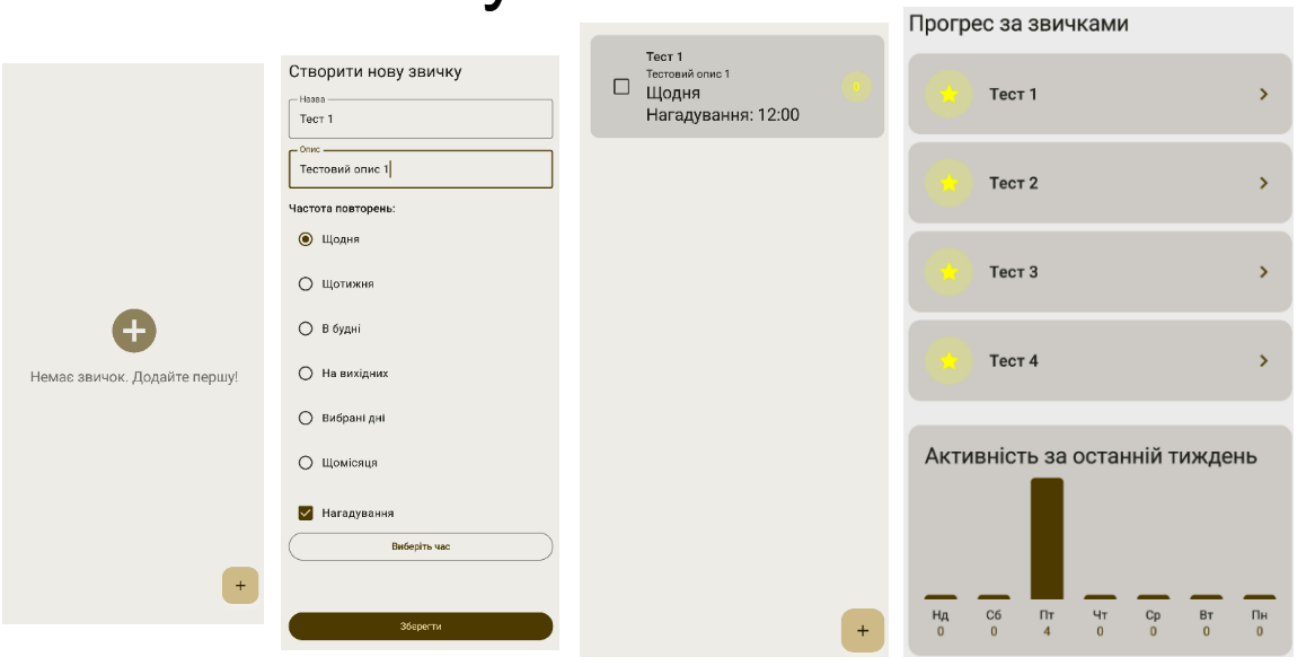


Рисунок Г.16 — Тестування системи (частина 2)

# Висновки

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено програмні модулі мобільної системи персоналізованого формування корисних звичок з урахуванням користувацького прогресу. Бакалаврську кваліфікаційну роботу реалізовано згідно методичних вказівок [20].

У ході виконання роботи було здійснено аналіз сучасного стану питання, розглянуто основні аналоги створюваного програмного продукту і проведено порівняння з ними, враховуючи їхні переваги та недоліки. На основі цього порівняння було сформульовано основні завдання дослідження.

Під час аналізу засобів розробки було обґрунтовано вибір мови програмування Kotlin, фреймворку Android для мобільної розробки, та відповідних технологій для реалізації персоналізованої системи формування звичок.

У першому розділі було здійснено комплексний аналіз сучасного стану систем формування звичок, детально розглянуто теоретичні та практичні аспекти цього процесу та систематизовано основні проблеми, з якими стикаються користувачі таких систем. Також проведено ґрунтовне порівняльне дослідження запропонованого рішення з існуючими на ринку платформами, виявлено суттєві конкурентні переваги та інноваційні особливості розроблюваної системи. На основі аналізу чітко сформульовано стратегічні та тактичні завдання, що визначають напрямки подальшої роботи.

Рисунок Г.17 — Висновки (частина 1)

# Висновки

У другому розділі представлено розгорнутий опис архітектурної моделі системи проєкту із застосуванням широкого спектру інструментів UML-моделювання, включаючи діаграми класів, послідовностей, компонентів та розгортання. Окрема увага приділена детальному опису ключових алгоритмів, що забезпечують ефективне функціонування мобільної системи персоналізованого формування звичок, зокрема алгоритмів обробки користувацьких даних, персоналізації рекомендацій та оптимізації циклу зворотного зв'язку.
У третьому розділі викладено вичерпну інформацію щодо процесу розробки програмних модулів, необхідних для повноцінного функціонування мобільної системи. Наведено огляд сучасних технологій та інструментальних засобів, застосованих під час розробки, з обґрунтуванням їх вибору. Описано внутрішню структуру та принципи взаємодії розроблених модулів управління звичками, відстеження прогресу, аналізу даних, системи сповіщень. Наведено технічні особливості реалізації користувацького інтерфейсу з використанням мови програмування Kotlin та компонентів бібліотеки Jetpack Compose.
У четвертому розділі представлено результати всебічного тестування програмних модулів системи, розроблених в рамках бакалаврської кваліфікаційної роботи, із зазначенням методології тестування, тестових сценаріїв та отриманих метрик якості. Розділ також містить докладну, покрокову інструкцію користувача з ілюстраціями та практичними прикладами для ефективного використання мобільної системи персоналізованого формування корисних звичок, що забезпечує максимально зручне освоєння функціоналу користувачами різного рівня технічної підготовки.

Рисунок Г.18 — Висновки (частина 2)

# Апробація результатів роботи і публікації

**Апробація результатів роботи** . Результати бакалаврської кваліфікаційної роботи доповідались на конференції «Modern Scientific Research: Theoretical and Practical Aspects» у 2025 році.
**Публікації**. Результати роботи були опубліковані в матеріалах конференцій: «Modern Scientific Research: Theoretical and Practical Aspects».

Рисунок Г.19 — Апробація результатів роботи і публікації