НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____Ігор БОЛБОТ	**_____ Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення гейміфікованого мобільного застосунку підтримки фокус-сесії»
Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи _____ **Белла ГОЛУБ**
к.т.н., доцент (підпис)

Виконала _____ **Ангеліна СТРІШЕНЕЦЬ**
(підпис)

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

ЗАВДАННЯ ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Стрішенець Ангеліні Олександрівні
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи
«Програмне забезпечення гейміфікованого мобільного застосунку підтримки фокус-сесії»
затверджена наказом від «19» грудня 2025 р. № 3204 «С»
Термін подання завершеної роботи на кафедру «22» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи:
Створення мобільного застосунку для організації фокус-сесій, збереження активності користувача та підтримки мотивації за допомогою елементів гейміфікації.
Перелік питань, що підлягають дослідженню:

1. Системний аналіз предметної області.
2. Проєктування та розробка інформаційного забезпечення.
3. Проєктування та розробка програмного забезпечення.
4. Рекомендації щодо впровадження та експлуатації системи.

Дата видачі завдання «19» грудня 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Белла ГОЛУБ**

Завдання взяла до виконання _____ (підпис) **Ангеліна СТРІШЕНЕЦЬ**

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

PieceTime – назва розробленого мобільного застосунку для організації фокус-сесій із використанням гейміфікації.

Android – мобільна операційна система, для якої розроблено та протестовано застосунок.

APK – Android Package Kit – формат інсталяційного файлу Android-застосунку.

Cloud Firestore – хмарна документоорієнтована база даних Firebase.

Dart – мова програмування, використана для розробки застосунку у Flutter.

ER – Entity-Relationship – модель зв’язків між сутностями даних.

Firebase – хмарна платформа, що використовується для автентифікації користувачів, збереження даних і синхронізації.

Firebase Authentication – сервіс Firebase для реєстрації, входу та підтвердження електронної пошти користувача.

Flutter – фреймворк для розробки мобільного застосунку з єдиною кодовою базою.

iOS – мобільна операційна система Apple, для якої застосунок може бути адаптований у майбутньому.

Riverpod – інструмент керування станом у Flutter-застосунку.

SharedPreferences – локальне сховище, що використовується для тимчасового збереження даних на пристрої.

UI – User Interface – користувацький інтерфейс.

UML – Unified Modeling Language – мова моделювання програмних систем.

XP – Experience Points – очки досвіду користувача.

ВСТУП

Здатність зосереджуватися на одній справі для багатьох людей стала складним завданням. Під час навчання, роботи або виконання повсякденних справ увага часто перемикається на повідомлення, соціальні мережі, сторонні думки або інші відволікання. Через це навіть прості завдання можуть займати більше часу, ніж очікувалося.

Існує й інша проблема: частина користувачів, навпаки, занадто довго працює без перерв. Спочатку це може створювати враження високої продуктивності, але з часом призводить до втоми, зниження концентрації та складнішого виконання наступних завдань. Тому важливо не тільки підтримувати увагу, а й чергувати роботу з коротким відпочинком.

Одним із практичних підходів є поділ роботи на окремі часові проміжки, у межах яких користувач зосереджується на конкретному завданні, а після цього робить перерву. Однак звичайний таймер не завжди мотивує користувача дотримуватися такого режиму регулярно. Через це виникає потреба у застосунку, який не лише відраховує час, а й показує прогрес, зберігає результати активності та підтримує мотивацію за допомогою елементів гейміфікації.

Метою бакалаврської кваліфікаційної роботи є розробка програмного забезпечення гейміфікованого мобільного застосунку підтримки фокус-сесій, який допомагає користувачеві організовувати власний робочий час, чергувати періоди зосередженої роботи з відпочинком, накопичувати статистику активності та підтримувати мотивацію за допомогою ігрових механік.

Об’єктом дослідження є процес організації зосередженої діяльності користувача під час виконання навчальних, професійних та особистих завдань.

Предметом дослідження є методи, підходи та програмні засоби реалізації мобільного застосунку для підтримки фокус-сесій із використанням механізмів гейміфікації, збору статистики та синхронізації даних користувача.

Для досягнення поставленої мети в роботі необхідно вирішити такі завдання:

1. провести аналіз предметної області та сучасних підходів до організації фокусованої роботи;
2. визначити функціональні та нефункціональні вимоги до програмної системи;
3. спроєктувати інформаційне та програмне забезпечення мобільного застосунку;
4. реалізувати основні функціональні модулі програмного продукту;
5. реалізувати механізми збору статистики та гейміфікації активності користувача;
6. забезпечити синхронізацію даних користувача та підтримку роботи окремих функцій за відсутності мережевого з’єднання;
7. провести тестування розробленого програмного забезпечення.

Під час розробки застосунку використано фреймворк «Flutter», мову програмування «Dart» та сервіси «Firebase» для автентифікації користувачів, збереження даних і синхронізації інформації між пристроями. Також реалізовано локальні сповіщення, підтримку фонової роботи таймера та механізм відкладеного збереження результатів сесій із подальшою синхронізацією після відновлення мережевого з’єднання.

Попередні результати, пов’язані з розробкою мобільного застосунку PieceTime, були представлені у тезах на тему «Програмне забезпечення гейміфікованого мобільного застосунку для підтримки фокус-сесій» у межах VII Всеукраїнської науково-практичної конференції студентів та аспірантів «Теоретичні та прикладні аспекти розробки комп’ютерних систем 2026». У тезах було розглянуто ідею застосунку, його призначення, використання фокус-сесій, елементів гейміфікації, хмарних сервісів «Firebase», локальних сповіщень і фонового виконання таймера.

Бакалаврська кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. У першому розділі розглянуто предметну область, сформовано вимоги до програмної системи, побудовано UML-моделі та виконано огляд існуючих рішень. У другому розділі

спроєктовано інформаційне забезпечення застосунку, описано логічну модель даних, обґрунтовано вибір «Firebase Firestore» та наведено фізичну структуру бази даних. У третьому розділі описано розробку програмного забезпечення, структуру об’єктів, компоненти системи, вибір інструментарію, алгоритми й основні програмні модулі. У четвертому розділі наведено результати тестування, вимоги до апаратного та програмного забезпечення, а також склад інсталяційного пакету мобільного застосунку.

Робота викладена на вісімдесяти трьох сторінках, містить тридцять п’ять рисунків, п’ять таблиць, двадцять п’ять використаних джерел та п’ять додатків.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметна область роботи пов’язана з організацією фокус-сесій – обмежених у часі періодів роботи, які чергуються з короткими перервами [1, 2, 3]. Такий підхід використовується тоді, коли користувачеві потрібно не просто запустити таймер, а підтримувати більш стабільний ритм роботи, а саме не відволікатися занадто часто і водночас не працювати без пауз до перевтоми.

У межах даної роботи фокус-сесія розглядається як послідовність циклів. Один цикл складається з двох фаз – фази фокусу та фази відпочинку. Фаза фокусу призначена для зосередженого виконання завдання, а фаза відпочинку – для короткого відновлення перед наступним циклом. Така структура добре підходить для мобільного застосунку, оскільки її можна описати через прості параметри: тривалість фокусу, тривалість перерви та кількість циклів.

У системі ці параметри об’єднуються в режим. Режим визначає готовий сценарій фокус-сесії: скільки хвилин триває робота, скільки триває перерва і скільки циклів потрібно пройти. У застосунку передбачені готові режими «Класичний», «Глибокий фокус» і «Легкий старт», а також окремий користувацький режим – «Власний», який можна налаштувати під власний режим роботи. Такий підхід зручніший за ручне введення параметрів перед кожною сесією, бо користувач може швидко перейти до потрібного сценарію.

Під час проходження сесії система повинна відстежувати її поточний стан. Для цього важливо знати активний режим, поточну фазу, номер циклу, залишок часу, кількість завершених фокус-блоків і завершених циклів. Ці дані потрібні не лише для відображення таймера на екрані, а й для подальшого збереження результату сесії, розрахунку досвіду та оновлення статистики користувача.

Окремою частиною предметної області є мотивація користувача. Звичайний таймер не завжди утримує інтерес до регулярного використання, тому в системі використовується гейміфікація [7, 8]. Вона базується на нарахуванні досвіду (XP), рівнях, бейджах і серіях активних днів. У роботі термін

«досягнення» використовується для позначення бейджів, які відкриваються після виконання певних умов активності. Такий варіант назви є зрозумілішим для користувача в інтерфейсі застосунку, тоді як у програмній реалізації можуть використовуватися службові назви, пов’язані з бейджами. XP нараховується за завершені хвилини фокусу, завершені цикли та повністю завершену сесію. Така логіка дозволяє винагороджувати користувача не тільки за повне проходження сесії, а й за частковий прогрес.

Для аналізу активності користувача система зберігає історію сесій. Запис сесії містить обраний режим, тривалість фокусу й перерви, заплановану кількість циклів, фактично завершені фокус-блоки, завершені цикли, отриманий XP, ознаку повного завершення сесії, час початку та завершення. На основі цих даних можна будувати статистику: загальну кількість сесій, хвилини фокусу, сумарний XP, кількість циклів, активність за період і серія днів. Серія днів у системі визначається за днями, у які користувач мав хоча б одну сесію, тобто не зберігається окремо вручну, а обчислюється на основі історії активності.

Важливою вимогою для такої системи є стабільність роботи незалежно від стану мережі. Фокус-сесія не повинна втрачатися через відсутність інтернету, тому результат сесії має бути збережений локально, якщо запис у хмарну базу даних тимчасово неможливий. Для цього в системі має бути передбачений механізм відкладеного збереження сесій: якщо хмарне сховище тимчасово недоступне, результат сесії зберігається локально, а після відновлення з’єднання синхронізується з базою даних.

Отже, предметна область охоплює не лише відлік часу, а повний цикл роботи з фокус-сесіями: вибір режиму, проходження фаз фокусу та відпочинку, фіксацію результату, нарахування мотиваційних показників, збереження статистики та синхронізацію даних користувача. Саме ця логіка стала основою для подальшого формування вимог і проєктування програмної системи PieceTime.

1.2 Аналіз вимог до програмної системи

На основі аналізу предметної області було сформовано вимоги до програмної системи. Їх визначення дозволяє окреслити основні можливості застосунку, зрозуміти межі його функціональності та зафіксувати технічні особливості, які потрібно враховувати під час подальшої розробки.

У межах даної роботи вимоги до системи доцільно поділити на дві групи: функціональні та нефункціональні. Функціональні вимоги описують можливості, які застосунок повинен надавати користувачеві. Нефункціональні вимоги визначають технічні характеристики системи, що впливають на стабільність її роботи, зручність використання та надійність збереження даних.

Функціональні вимоги до програмної системи наведено в таблиці 1.

Таблиця 1

Функціональні вимоги до програмної системи

№	Назва вимоги	Опис
1	Автентифікація користувача	Система повинна забезпечувати реєстрацію, вхід до застосунку, підтвердження електронної пошти та відновлення пароля
2	Керування режимами фокус-сесії	Користувач повинен мати можливість обирати готові режими або налаштовувати власний режим із заданою тривалістю фокусу, перерви та кількістю циклів
3	Керування проходженням сесії	Система повинна підтримувати запуск, паузу, продовження, пропуск поточної фази та дострокове завершення сесії
4	Фонова робота та сповіщення	Таймер повинен коректно працювати після згортання застосунку, а система – надсилати сповіщення про зміну фаз або завершення сесії

Таблиця 1 (продовження)

№	Назва вимоги	Опис
5	Збереження результатів сесії	Після завершення або дострокового припинення сесії система повинна зберігати її результат для подальшого обліку активності
6	Гейміфікація активності	Система повинна нараховувати досвід (XP), визначати рівень користувача, відкривати бейджі та вести облік серії активних днів
7	Статистика та аналітика	Користувач повинен мати доступ до загальної статистики активності та перегляду підсумків за вибраний період
8	Налаштування застосунку	Система повинна надавати можливість змінювати мову, тему, параметри сповіщень, вібрації та інші параметри роботи застосунку
9	Автономна робота та синхронізація	Основний функціонал застосунку повинен працювати без підключення до мережі, а накопичені дані мають синхронізуватися після відновлення з’єднання

Наведені функціональні вимоги охоплюють основні сценарії взаємодії користувача із системою: від входу в застосунок і вибору режиму до завершення сесії, збереження її результатів та перегляду накопиченої статистики. Окреме місце у структурі функціоналу займає модуль гейміфікації, який доповнює базову логіку таймера механізмами мотивації та підтримує інтерес користувача до регулярного використання застосунку.

Разом із цим для мобільного застосунку важливими є не лише окремі функції, а й умови їх стабільної роботи. Тому додатково визначено

нефункціональні вимоги, які описують якість роботи системи, зручність використання, надійність збереження даних та підтримку автономної роботи.

Нефункціональні вимоги до програмної системи наведено в таблиці 2.

Таблиця 2

Нефункціональні вимоги до програмної системи

№	Назва вимоги	Опис
1	Зручність використання	Інтерфейс застосунку повинен бути зрозумілим, логічним і зручним для щоденного використання
2	Швидкодія	Основні дії користувача повинні виконуватися без помітних затримок
3	Надійність збереження даних	Дані користувача не повинні втрачатися під час завершення сесії, збою застосунку або тимчасової відсутності мережі
4	Стабільність фонової роботи	Таймер повинен коректно працювати навіть після згортання застосунку або блокування пристрою
5	Автономність	Основні функції системи повинні залишатися доступними без підключення до мережі
6	Безпечність даних	Доступ до персональних даних користувача повинен бути захищений механізмами автентифікації
7	Масштабованість	Архітектура системи повинна дозволяти подальше розширення функціональних можливостей застосунку
8	Підтримка локалізації	Система повинна підтримувати роботу кількома мовами інтерфейсу

Серед нефункціональних вимог для даної системи найбільше значення мають стабільність роботи таймера у фоновому режимі, надійність збереження даних та підтримка автономної роботи. Фокус-сесія є процесом, що триває певний час, тому користувач повинен бути впевненим, що застосунок не втратить прогрес, коректно завершить сесію та збереже результат незалежно від стану мережі або активності самого застосунку на екрані.

Таким чином можна чітко визначити основні модулі системи, їх призначення та ключові особливості роботи. Саме вимоги на цьому етапі задають напрям подальшого проєктування, оскільки дозволяють перейти від загального бачення застосунку до побудови його внутрішньої структури, взаємодії компонентів і моделювання основних процесів системи.

1.3 Моделювання предметної області

Для кращого опису логіки роботи застосунку PieceTime було побудовано кілька UML-моделей, які показують взаємодію користувача із системою, послідовність виконання основних процесів і внутрішню логіку проходження фокус-сесії. Побудова таких моделей дозволяє ще до детального проєктування програмних модулів визначити основні сценарії роботи системи, взаємодію її компонентів та порядок виконання ключових дій.

У межах даної роботи було використано діаграму прецедентів, діаграми послідовності та діаграму активності. Кожна з них описує систему з різних точок зору: дії користувача, взаємодію модулів під час роботи таймера та загальний алгоритм проходження фокус-сесії. Разом ці моделі дають змогу показати не тільки окремі функції застосунку, а й зв’язок між ними: користувач обирає режим, запускає сесію, проходить фази фокусу та відпочинку, після чого система формує результат і оновлює прогрес.

Побудовано діаграму прецедентів, яка описує основні сценарії взаємодії користувача із застосунком. На діаграмі показано, що користувач може авторизуватись, обрати або налаштувати режим сесії, запустити сесію, керувати її проходженням, переглянути статистику та змінити налаштування застосунку.

Така діаграма потрібна для того, щоб визначити межі системи та показати, які дії виконує саме користувач, а які належать до внутрішньої логіки застосунку.

На рис. 1.1 наведено діаграму прецедентів мобільного застосунку PieceTime.

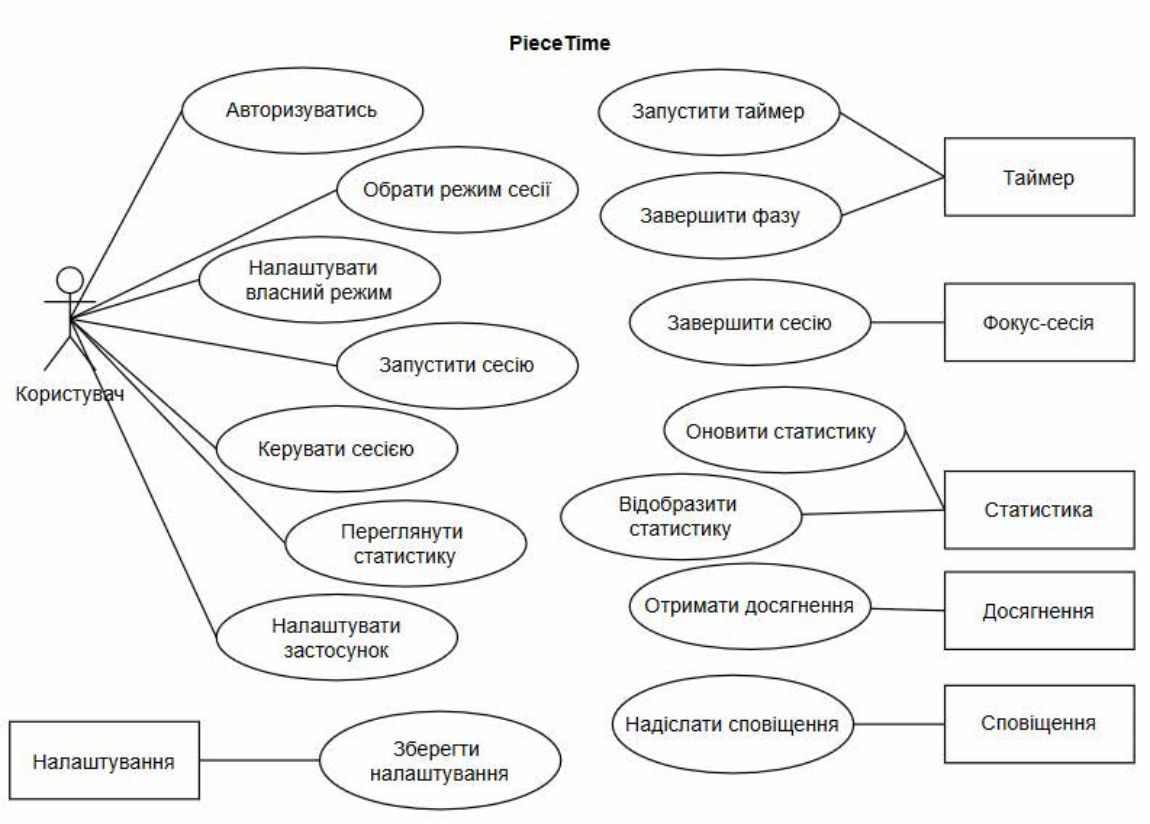

Рис. 1.1 Діаграма прецедентів мобільного застосунку PieceTime

На діаграмі також виділено основні об’єкти предметної області: таймер, фокус-сесію, статистику, досягнення, сповіщення та налаштування. Це показує, що застосунок не обмежується лише відліком часу. Він також зберігає активність користувача, оновлює статистику та підтримує систему прогресу.

Для уточнення окремих процесів було побудовано діаграми послідовності. Вони дозволяють показати не просто перелік можливостей системи, а порядок взаємодії між користувачем і основними частинами застосунку. Першою було змодельовано запуск фокус-сесії, оскільки саме з цього починається основний сценарій використання PieceTime.

У цьому процесі користувач спочатку обирає режим сесії. Після цього фокус-сесія отримує параметри вибраного режиму: тривалість фокусу,

тривалість перерви та кількість циклів. Лише після цього система може сформувати сценарій проходження сесії та перейти до запуску таймера.

На рис. 1.2 наведено діаграму послідовності запуску фокус-сесії.

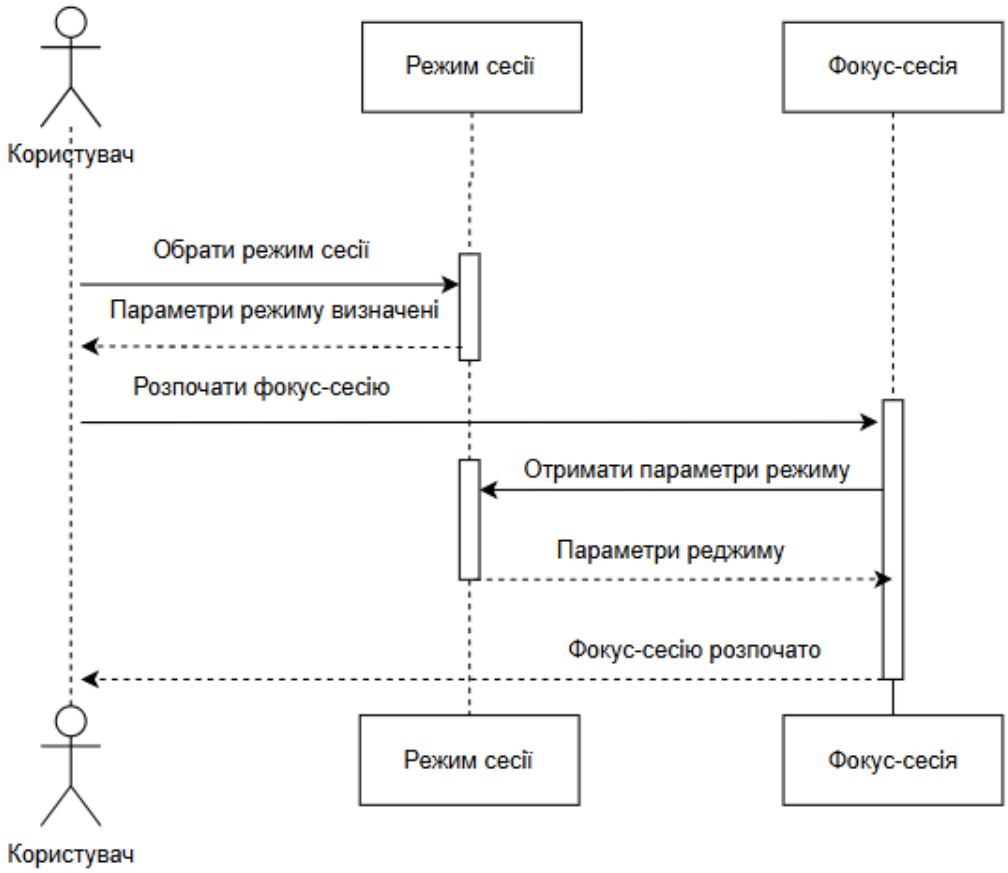

Рис. 1.2 Діаграма послідовності запуску фокус-сесії

Діаграма показує, що фокус-сесія не запускається без попередньо визначених параметрів. Спочатку користувач обирає режим, а вже потім система використовує його налаштування: тривалість фокусу, тривалість перерви та кількість циклів. Такий підхід дозволяє одразу сформувати сценарій сесії та уникнути ситуації, коли таймер запускається без потрібних початкових даних.

Наступна діаграма описує проходження фокус-сесії. Вона не показує внутрішню щосекундну роботу таймера, а відображає сам предметний процес: виконання циклів, проходження фази фокусу та фази перерви. Це важливо, тому що для предметної області головним є не кожне оновлення секундоміра, а зміна станів сесії та поступове проходження запланованих циклів.

У цьому сценарії користувач проходить фазу фокусу, після чого система переводить сесію до фази відпочинку. Після завершення обох фаз цикл вважається виконаним. Якщо в режимі передбачено кілька циклів, така послідовність повторюється до завершення всієї сесії.

На рис. 1.3 наведено діаграму послідовності проходження фокус-сесії.

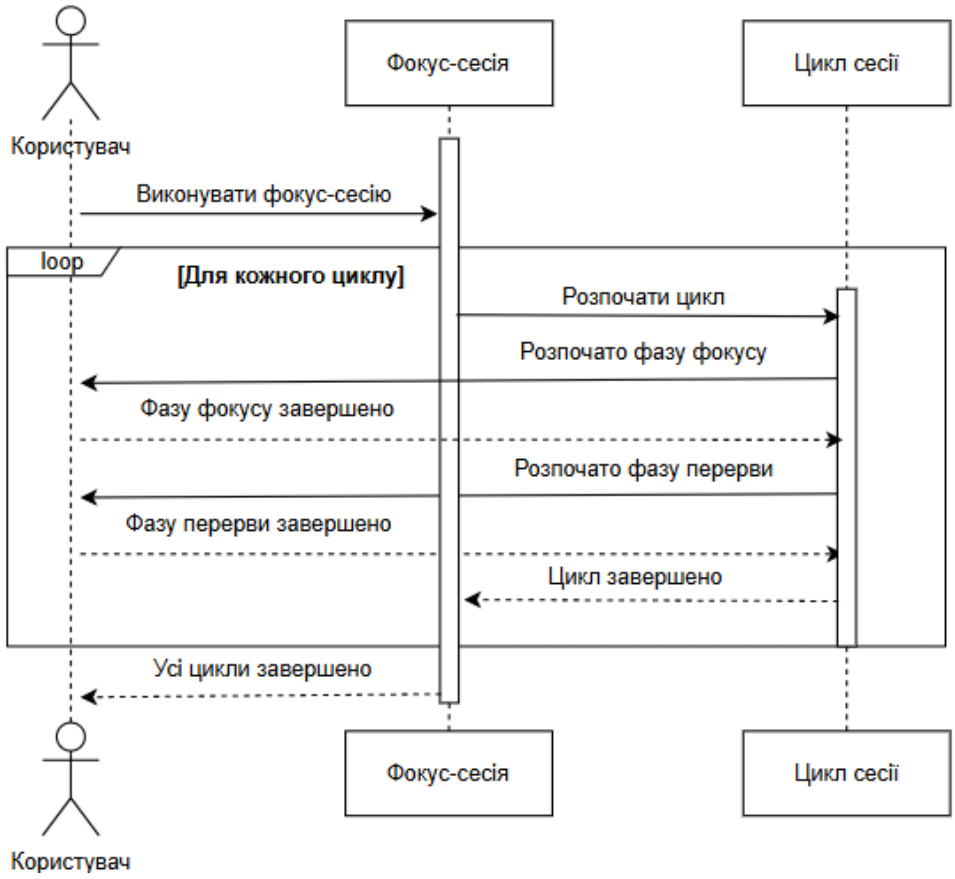

Рис. 1.3 Діаграма послідовності проходження фокус-сесії

У межах кожного циклу користувач спочатку проходить фазу фокусу, після чого переходить до фази перерви. Після завершення обох фаз цикл вважається завершеним. Якщо в сесії передбачено кілька циклів, процес повторюється до проходження всіх запланованих циклів. Така логіка дозволяє відокремити сам процес проходження сесії від етапу її підсумкового збереження.

Також було змодельовано завершення фокус-сесії. Цей процес важливий, бо саме після нього формується результат роботи користувача, оновлюється прогрес і перевіряються умови отримання досягнень. На цьому етапі система вже має дані про фактично завершені фокус-блоки, цикли, отриманий XP та ознаку повного або часткового завершення сесії.

Завершення сесії також пов’язане з оновленням статистики. Результат фокус-сесії надалі використовується для підрахунку загальної кількості сесій, хвилин фокусу, завершених циклів, XP та серії активних днів. Тому цей процес виділено в окрему діаграму послідовності.

На рис. 1.4 наведено діаграму послідовності завершення фокус-сесії.

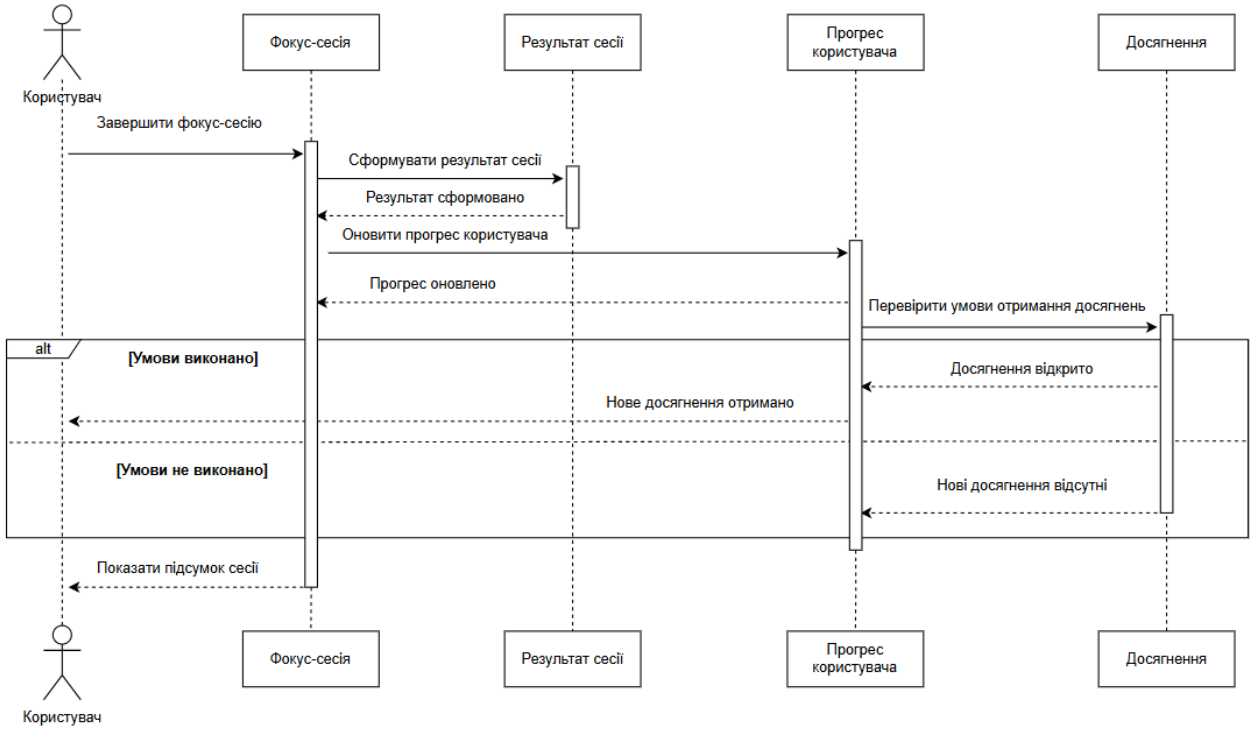

Рис. 1.4 Діаграма послідовності завершення фокус-сесії

Після завершення фокус-сесії формується результат сесії. Далі на його основі оновлюється прогрес користувача. Після цього система перевіряє, чи виконані умови для отримання нового досягнення. Якщо умови виконано, користувач отримує досягнення. Якщо ні, сесія завершується без зміни списку досягнень. Після цього користувачеві показується підсумок фокус-сесії.

Для узагальнення всього процесу було побудовано діаграму активності. Вона показує проходження фокус-сесії від вибору режиму до показу підсумку.

Діаграму активності проходження фокус-сесії наведено у додатку А на рис. А.1.

На діаграмі показано, що користувач спочатку обирає режим сесії. Якщо обрано власний режим, перед запуском задаються його параметри. Якщо обрано готовий режим, система використовує вже визначені налаштування. Далі

проходить фаза фокусу, фаза перерви та завершується цикл сесії. Якщо є наступний цикл, процес повторюється. Після завершення всіх циклів формується результат, оновлюється прогрес користувача та перевіряються умови отримання досягнення.

Побудовані UML-моделі дозволили описати предметну область застосунку на рівні основних дій і процесів. Вони показують, як користувач проходить фокус-сесію, як формується результат і як цей результат впливає на статистику та досягнення. Надалі ці моделі використовуються як основа для проєктування інформаційного забезпечення та програмної структури застосунку.

1.4 Огляд інформаційних джерел та існуючих рішень

Для аналізу існуючих рішень було розглянуто кілька популярних сервісів для організації фокусованої роботи. Основна увага приділялася не кількості функцій чи популярності застосунку, а тому, як у них реалізовано фокус-сесії, систему мотивації, статистику та роботу таймера у фоні.

Одним із найвідоміших застосунків у цій категорії є «Forest». Його основна механіка побудована навколо віртуального дерева, яке росте під час фокус-сесії. Якщо користувач завершує сесію раніше часу або виходить із застосунку, дерево гине. За рахунок цього звичайний таймер сприймається менш «сухо», а користувач бачить візуальний результат своєї концентрації.

Сам застосунок зроблений доволі просто. Інтерфейс не перевантажений, а запуск сесії займає буквально кілька дій. Це добре працює саме для коротких фокус-сесій, коли користувач не хоче витрачати час на додаткові налаштування.

Разом із цим «Forest» [4] має й певні обмеження. Основний акцент тут зроблений саме на візуальній мотивації, тому система статистики реалізована досить просто. Також у застосунку немає окремої системи рівнів, XP чи великої кількості досягнень. Можливості налаштування самих фокус-сесій теж досить обмежені.

Ще одним популярним рішенням є «Focus To-Do» [5]. У цьому застосунку фокус-сесії поєднані зі списками завдань та елементами планування. Користувач

може створювати задачі, запускати «Pomodoro»-сесії та переглядати статистику виконаної роботи.

На відміну від «Forest», тут значно більше уваги приділено саме організації роботи. Застосунок підтримує різні режими таймера, календар активності, нагадування та синхронізацію між пристроями. Через це система виглядає більш функціональною, але водночас і більш перевантаженою. Частина інтерфейсу пов’язана саме з менеджментом задач, тому сам таймер уже не є центральним елементом застосунку.

Також було розглянуто сервіс «Pomofocus» [6]. Це значно простіше рішення, побудоване навколо базової логіки «Pomodoro»-таймера. Користувач може запускати фокус-сесії, змінювати тривалість фаз і переглядати базову статистику активності.

«Pomofocus» має дуже простий інтерфейс. На екрані знаходяться лише основні елементи, потрібні для роботи таймера. За рахунок цього система запускається швидко і не перевантажує користувача зайвими функціями.

Водночас функціональність сервісу досить обмежена. У ньому практично відсутня гейміфікація, немає системи прогресу, досягнень чи розширеної статистики. Сервіс більше підходить саме як мінімалістичний таймер для коротких фокус-сесій.

Для порівняння розглянутих рішень було сформовано таблицю 3.

Таблиця 3

Порівняння існуючих рішень та застосунку PieceTime

Функціональність	**Forest**	**Focus To-Do**	**Pomofocus**	**PieceTime**
Готові режими фокус-сесій	+	+	+	+
Користувацький режим	-	+	+	+
Гейміфікація	+	частково	-	+

Таблиця 3 (продовження)

Функціональність	**Forest**	**Focus To-Do**	**Pomofocus**	**PieceTime**
Система XP та рівнів	-	-	-	+
Система досягнень	-	-	-	+
Серія активних днів	-	частково	-	+
Детальна статистика	частково	+	частково	+
Локальні сповіщення	+	+	+	+
Фонова робота таймера	+	+	залежить від платформи	+
Автономна робота без мережі	частково	частково	+	частково
Синхронізація даних	+	+	-	+

Розглянуті застосунки вирішують задачу організації фокусованої роботи по-різному. У Forest основний акцент зроблено на мотивації користувача, у Focus To-Do – на поєднанні таймера із плануванням задач, а Pomofocus побудований навколо простого підходу до роботи з фокус-сесіями.

Під час аналізу стало помітно, що більшість рішень концентруються лише на окремій частині функціоналу. Одні застосунки мають простий таймер, але слабку систему прогресу. Інші підтримують статистику та планування, але перевантажують інтерфейс або недостатньо працюють із мотивацією користувача.

Під час формування вимог до PieceTime основна увага приділялася поєднанню кількох важливих частин системи одночасно: фокус-сесій, статистики, гейміфікації, автономної роботи та синхронізації даних. За рахунок

цього застосунок не обмежується лише роботою таймера, а дозволяє користувачеві накопичувати прогрес і відстежувати власну активність.

1.5 Постановка завдання

На основі аналізу предметної області, вимог до системи та огляду існуючих рішень було визначено завдання розробки програмного забезпечення гейміфікованого мобільного застосунку PieceTime. Застосунок має допомагати користувачеві організовувати фокус-сесії, чергувати періоди роботи з короткими перервами, зберігати результати активності та підтримувати мотивацію через систему прогресу.

У межах роботи необхідно розробити мобільний застосунок, який дозволяє користувачеві створити обліковий запис, обрати готовий режим фокус-сесії або налаштувати власний режим, запустити таймер та керувати проходженням сесії. Під час роботи сесії система повинна підтримувати паузу, продовження відліку, пропуск поточної фази та дострокове завершення.

Окремою частиною завдання є реалізація збереження результатів фокус-сесій. Після завершення роботи застосунок має фіксувати обраний режим, кількість завершених циклів, кількість завершених блоків фокусу, отриманий досвід, час початку та завершення сесії. Ці дані надалі використовуються для побудови статистики, оновлення прогресу користувача та відображення історії активності.

Також потрібно реалізувати механізми гейміфікації. Система має нараховувати досвід, оновлювати рівень користувача, перевіряти умови отримання досягнень та враховувати серію активних днів. Це потрібно для того, щоб застосунок не обмежувався функцією таймера, а підтримував регулярне використання через зрозумілий для користувача прогрес.

Важливою вимогою є підтримка часткової автономної роботи. Фокус-сесія та її результат не повинні втрачатися через відсутність інтернету. Якщо під час завершення сесії немає доступу до хмарної бази даних, результат має зберігатися локально та синхронізуватися після відновлення з’єднання. При цьому

статистика на окремих екранах може оновлюватися після появи мережі, оскільки частина даних зберігається у хмарному сховищі.

Для виконання поставленого завдання необхідно реалізувати основні частини системи: модуль авторизації, модуль керування режимами, таймер фокус-сесії, систему локальних сповіщень, модуль статистики, систему досягнень, налаштування застосунку, локальне збереження незбережених сесій та синхронізацію з хмарною базою даних.

Таким чином, поставлене завдання передбачає розробку мобільного застосунку, у якому поєднуються таймер фокус-сесій, персональна статистика, система мотивації та механізми збереження даних. Надалі ці вимоги використовуються як основа для проєктування інформаційного та програмного забезпечення системи.

Висновки до розділу 1

У першому розділі було розглянуто предметну область застосунку PieceTime та визначено основну логіку роботи з фокус-сесіями. Фокус-сесію описано не як звичайний таймер, а як послідовність циклів, де кожен цикл складається з фази фокусу та фази відпочинку. Такий підхід дозволив чітко визначити, які дані система повинна зберігати під час проходження сесії: обраний режим, поточну фазу, кількість завершених блоків, циклів, отриманий XP та підсумковий результат.

Також у розділі було сформовано функціональні та нефункціональні вимоги до програмної системи. Окрему увагу приділено стабільній роботі таймера, збереженню результатів сесій, гейміфікації, статистиці, локальним сповіщенням і підтримці роботи без постійного доступу до мережі. Це важливо для такого типу застосунку, оскільки користувач не повинен втрачати прогрес через згортання програми або тимчасову відсутність інтернету.

Для уточнення логіки системи було побудовано UML-моделі, які показують основні сценарії взаємодії користувача із застосунком, запуск фокус-сесії, її проходження та завершення. Аналіз існуючих рішень показав, що популярні застосунки часто роблять акцент лише на окремих частинах

функціоналу: таймері, плануванні задач або простій мотивації. Тому для PieceTime було обґрунтовано поєднання фокус-сесій, статистики, XP, досягнень, серії активних днів, автономного збереження та синхронізації даних.

У результаті першого розділу було сформовано основу для подальшого проєктування системи: визначено проблему, основні вимоги, ключові сценарії роботи та межі функціональності мобільного застосунку.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Під час проєктування застосунку PieceTime було необхідно визначити структуру даних, яка буде використовуватися для роботи таймера, збереження результатів фокус-сесій, налаштувань користувача та системи прогресу. Оскільки застосунок працює із персональною статистикою, досягненнями та історією активності, структура даних повинна забезпечувати зручне збереження й оновлення цих записів без зайвого дублювання інформації.

Для PieceTime важливо, щоб дані користувача були пов’язані між собою логічно. Наприклад, результати фокус-сесій впливають на статистику, XP, рівень і досягнення, а налаштування користувача визначають поведінку таймера, сповіщень та інтерфейсу. Тому модель даних має відображати не тільки окремі сутності, а й те, як вони взаємодіють між собою під час роботи застосунку.

Також під час побудови моделі потрібно було врахувати, що частина даних змінюється часто, а частина використовується як довідкова або налаштувальна. Наприклад, записи фокус-сесій створюються після кожного проходження таймера, статистичні показники оновлюються після збереження результату, а параметри теми, мови або сповіщень змінюються значно рідше. Через це такі дані доцільно розділяти за призначенням, щоб спростити їх подальше оновлення та використання в різних екранах застосунку.

Для цього була сформована логічна модель даних, яка описує основні сутності системи та зв’язки між ними. Під час проєктування враховувалось, що застосунок використовує «Firebase Firestore», тому структура орієнтована не на класичну реляційну базу даних, а на збереження колекцій і пов’язаних записів користувача. Водночас ER-діаграма використовується як зручний спосіб показати логіку даних на концептуальному рівні: які об’єкти є в системі, які дані вони містять і як пов’язані між собою.

Логічну модель даних застосунку PieceTime подано на рис. 2.1.

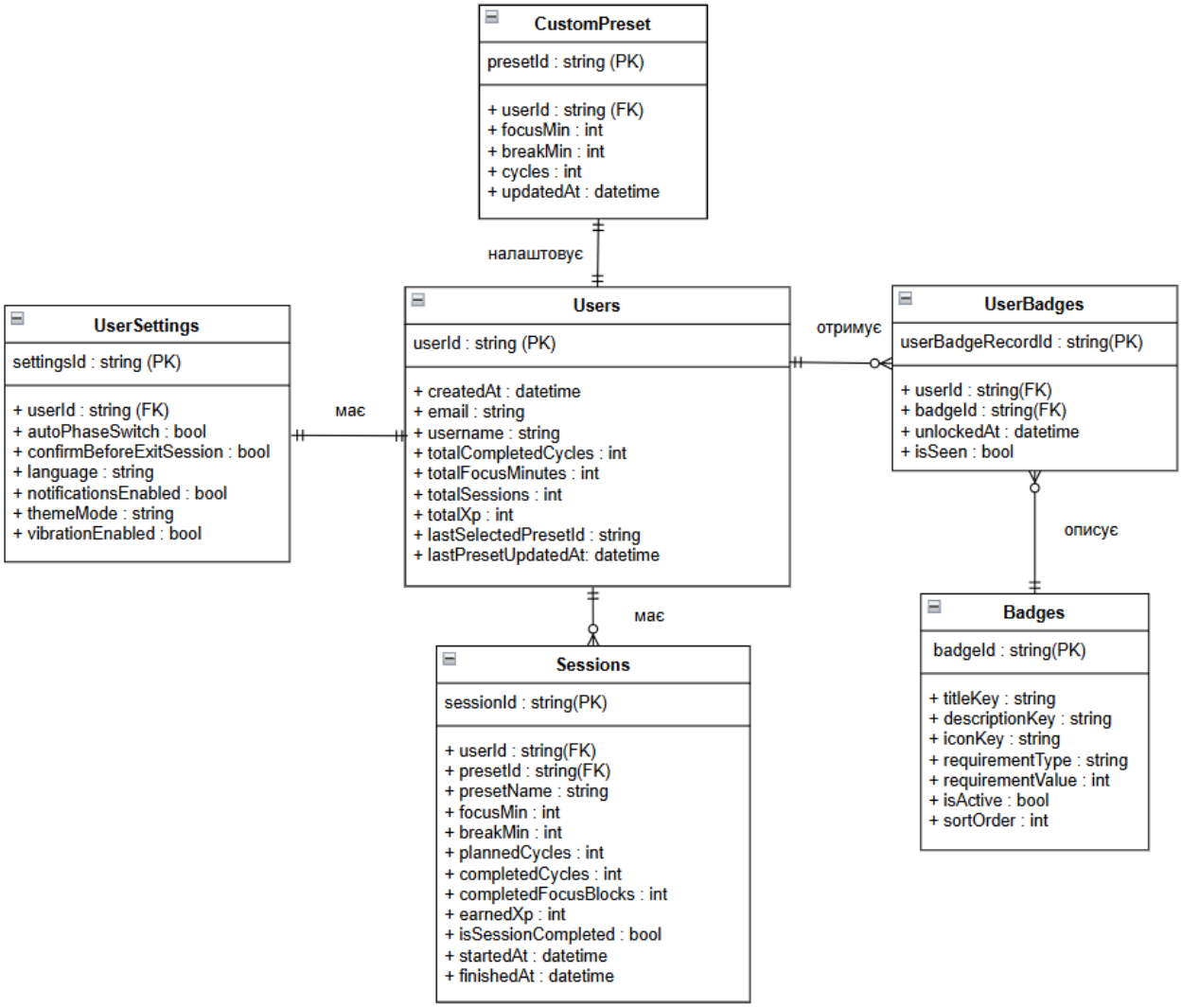

Рис. 2.1 Логічна модель даних застосунку

Центральною сутністю системи є «Users». Вона містить основну інформацію про користувача та його загальний прогрес у системі. Саме тут зберігаються сумарні показники активності: кількість завершених циклів, хвилини фокусу, кількість сесій та накопичений XP. Також у цій сутності зберігається інформація про останній вибраний режим роботи, що дозволяє швидко відновлювати попередній сценарій фокус-сесії без повторного налаштування.

Для збереження результатів роботи використовується сутність «Sessions». Вона містить інформацію про кожну окрему фокус-сесію: обраний режим, тривалість фаз, кількість циклів, отриманий XP, кількість завершених фокус-блоків, а також час початку та завершення сесії. За рахунок цього система може

формувати статистику активності користувача та відображати історію проходження сесій.

Окремо винесено сутність «UserSettings», яка використовується для збереження параметрів застосунку. У ній зберігаються налаштування автоматичного переходу між фазами, підтвердження завершення сесії, мови інтерфейсу, теми застосунку, сповіщень і вібрації. Такий підхід дозволяє відокремити логіку роботи користувача від параметрів інтерфейсу та поведінки системи.

Для підтримки користувацького режиму використовується сутність «CustomPreset». Вона містить параметри власної фокус-сесії користувача: тривалість фокусу, тривалість перерви та кількість циклів. Це дозволяє зберігати індивідуальні налаштування між запусками застосунку та повторно використовувати власний режим без повторного введення параметрів.

Окрему частину моделі даних займає система досягнень. Сутність «Badges» містить опис доступних досягнень, їх тип, умови отримання та параметри відображення. Для збереження прогресу конкретного користувача використовується сутність «UserBadges», яка містить інформацію про отримані досягнення, дату розблокування та стан перегляду нового бейджа.

Між сутностями передбачено зв’язки, які дозволяють об’єднати дані користувача в межах єдиної системи. Один користувач може мати багато сесій та багато отриманих досягнень, але водночас лише один набір налаштувань і один користувацький режим. Така структура дозволяє уникнути дублювання даних та спрощує подальшу роботу із синхронізацією інформації між пристроями.

Сформована логічна модель даних охоплює основні процеси роботи застосунку: проходження фокус-сесій, збереження статистики, налаштування режимів роботи та систему гейміфікації. Побудована структура стала основою для подальшого вибору системи управління даними та розробки інформаційної бази застосунку.

2.2 Вибір системи управління базами даних

Під час розробки застосунку PieceTime необхідно було обрати систему збереження даних, яка б підтримувала роботу з користувацькими акаунтами, історією фокус-сесій, статистикою, налаштуваннями та системою досягнень. Оскільки застосунок орієнтований на щоденне використання та роботу на мобільних пристроях, важливо було забезпечити не лише збереження інформації, а й синхронізацію даних між різними пристроями користувача.

У системі необхідно зберігати результати завершених фокус-сесій, кількість отриманого досвіду, статистику активності, параметри користувацького режиму та інформацію про отримані досягнення. Частина цих даних постійно оновлюється під час роботи застосунку, тому система збереження повинна підтримувати швидке оновлення документів та роботу із вкладеними структурами даних.

Для реалізації інформаційного забезпечення було обрано «Firebase Firestore» [13]. Це документоорієнтована хмарна база даних, яка добре підходить для мобільних застосунків із прив’язкою даних до конкретного користувача. У «Firestore» інформація організовується у вигляді колекцій та документів, що дозволяє логічно групувати дані користувача: сесії, налаштування, прогрес і досягнення.

Вибір «Firestore» пов’язаний із кількома причинами. По-перше, система добре інтегрується з «Flutter» та «Firebase Authentication», який у застосунку використовується для авторизації користувачів. Це дозволяє швидко отримувати доступ до даних конкретного користувача без окремої реалізації серверної частини.

По-друге, «Firestore» підтримує автоматичну синхронізацію даних між пристроями. Якщо користувач увійде у власний акаунт на іншому пристрої, система зможе отримати його статистику, налаштування та історію сесій без додаткового перенесення даних вручну.

Окремо важливою для PieceTime стала підтримка часткової роботи без мережевого з’єднання. У застосунку реалізовано механізм локального

збереження незасинхронізованих фокус-сесій. Якщо під час завершення сесії відсутній доступ до інтернету, результат тимчасово зберігається локально, а після відновлення з’єднання автоматично синхронізується з «Firebase Firestore». Такий підхід дозволяє уникнути втрати прогресу користувача.

Використання класичної реляційної бази даних у межах даного застосунку було б менш зручним, оскільки більшість даних у системі напряму належать конкретному користувачеві та мають вкладену структуру. У випадку «Firestore» ці дані можна організувати значно простіше: через окремі колекції сесій, налаштувань і досягнень користувача.

Таким чином, «Firebase Firestore» дозволив реалізувати хмарне збереження, синхронізацію між пристроями, підтримку авторизації користувачів та часткову автономну роботу застосунку. Обрана система управління даними відповідає особливостям мобільного застосунку PieceTime та структурі його інформаційної моделі.

2.3 Розробка структури інформаційної бази

Після формування логічної моделі даних та вибору «Firebase Firestore» було розроблено структуру інформаційної бази застосунку PieceTime. На цьому етапі визначалося, як саме будуть організовані колекції, документи та зв’язки між даними користувача в межах системи.

Оскільки застосунок працює із персональною інформацією користувача, структура бази будується навколо колекції «users». Кожен користувач має власний документ, у якому зберігаються основні показники прогресу та вкладені колекції із пов’язаними даними. Такий підхід дозволяє ізолювати інформацію різних користувачів та спрощує роботу із синхронізацією даних.

У документі користувача зберігаються загальні статистичні показники: кількість завершених сесій, хвилини фокусування, сумарний XP, кількість завершених циклів, ім’я користувача та службова інформація акаунта. Ці дані використовуються для формування статистики, системи прогресу та відображення інформації у профілі застосунку.

Для збереження історії проходження фокус-сесій використовується вкладена колекція «sessions». Кожен документ у цій колекції описує окрему сесію користувача. У ньому зберігаються параметри режиму, тривалість фаз, кількість циклів, отриманий XP, кількість завершених блоків фокусування, а також час початку та завершення сесії. Така структура дозволяє швидко формувати історію активності та обчислювати статистику користувача.

Налаштування застосунку винесені в окрему колекцію «settings». У ній зберігаються параметри теми оформлення, мови інтерфейсу, автоматичного переходу між фазами, сповіщень, вібрації та підтвердження завершення сесії. Відокремлення налаштувань від статистики та історії сесій спрощує оновлення параметрів без впливу на інші частини системи.

Для підтримки користувацьких режимів роботи використовується колекція «presets». У ній зберігаються параметри власних режимів користувача: тривалість фокус-фази, перерви та кількість циклів. Це дозволяє повторно використовувати створені режими без необхідності повторного введення параметрів під час кожного запуску застосунку.

Система досягнень поділена на дві частини. Глобальна колекція «badges» містить опис усіх доступних досягнень у системі: тип досягнення, умови отримання, ідентифікатори текстів та параметри відображення. Для збереження прогресу конкретного користувача використовується вкладена колекція «user_badges», у якій фіксуються отримані досягнення, дата розблокування та стан перегляду нового бейджа.

Під час побудови структури інформаційної бази основна увага приділялася не складним зв’язкам між таблицями, а зручності роботи мобільного застосунку з даними користувача. Обрана структура дозволяє швидко отримувати необхідну інформацію, окремо оновлювати різні частини даних та підтримувати синхронізацію між пристроями через «Firebase Firestore».

Сформована структура інформаційної бази стала основою для подальшої реалізації фізичної структури сховища даних та організації колекцій у «Firebase Firestore».

2.4 Схема та фізична структура бази даних

Після визначення логічної моделі даних та вибору «Firebase Firestore» було сформовано фізичну структуру інформаційної бази застосунку PieceTime. На цьому етапі логічні сутності були переведені у структуру, яка відповідає способу збереження даних у «Firestore»: колекції, документи та вкладені підколекції.

Оскільки «Firestore» є документоорієнтованою базою даних, структура збереження відрізняється від класичних реляційних СУБД. Дані не поділяються на таблиці із зовнішніми ключами, а групуються навколо окремих документів. Для PieceTime це зручно, тому що більшість інформації напряму належить конкретному користувачеві: історія сесій, статистика, налаштування, власні режими та отримані досягнення.

Центральним елементом фізичної структури стала колекція «users». У ній для кожного облікового запису створюється окремий документ, а пов’язані з ним дані розміщуються у підколекціях. Такий підхід дозволяє ізолювати інформацію різних користувачів, швидко отримувати потрібні записи та не змішувати дані різного призначення в одному місці.

Фізична структура також враховує те, що різні частини застосунку звертаються до різних груп даних. Екран таймера працює з параметрами режиму та результатом поточної сесії, статистика використовує збережені сесії й агреговані показники користувача, а екран налаштувань звертається до окремих параметрів інтерфейсу та поведінки застосунку. Тому розміщення даних у підколекціях дозволяє не завантажувати зайву інформацію там, де вона не потрібна

На рис. 2.2 наведено схему фізичної структури бази даних застосунку PieceTime.

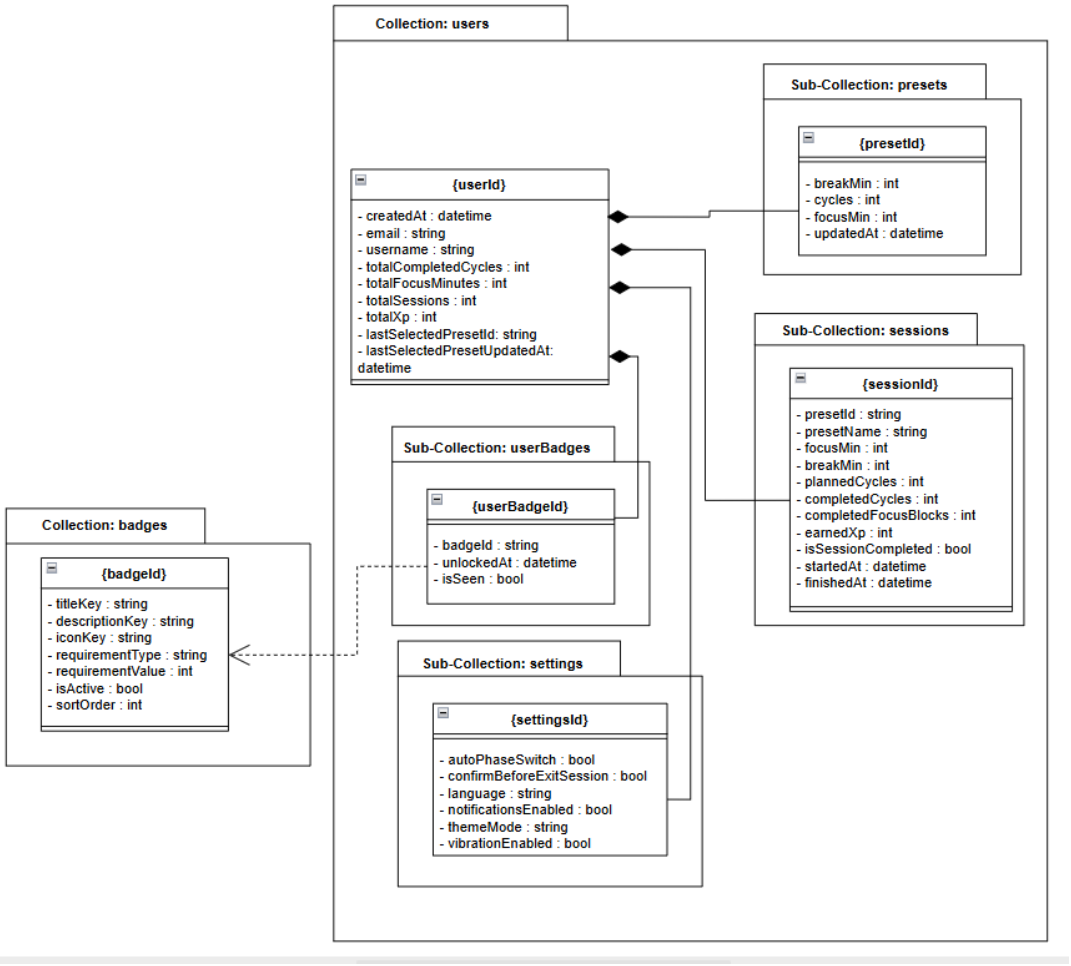

Рис. 2.2 Схема фізичної структури бази даних «Firebase Firestore»

У колекції «users» для кожного користувача створюється окремий документ «{userId}», який містить основну інформацію про профіль та загальну статистику активності. У документі зберігаються електронна пошта, ім’я користувача, дата створення акаунта, сумарна кількість завершених сесій, хвилин фокусу, циклів та накопиченого XP. Також окремо зберігається інформація про останній вибраний режим роботи.

Для збереження історії проходження фокус-сесій використовується підколекція «sessions». Кожен документ «{sessionId}» містить параметри окремої сесії: тип режиму, тривалість фаз, кількість циклів, отриманий досвід, дату початку та завершення сесії, а також інформацію про те, чи була сесія завершена повністю. Така структура дозволяє швидко отримувати історію активності користувача та формувати статистику роботи застосунку.

Користувацькі режими роботи зберігаються у підколекції «presets». У ній містяться параметри власних фокус-сесій: тривалість фокус-фази, перерви та кількість циклів. Це дозволяє користувачеві повторно використовувати власні налаштування без повторного введення параметрів.

Для збереження параметрів застосунку використовується підколекція «settings». У ній містяться налаштування теми оформлення, мови інтерфейсу, сповіщень, автоматичного переходу між фазами та підтвердження завершення сесії. Винесення цих параметрів в окрему структуру дозволяє відокремити логіку інтерфейсу від статистичних даних користувача.

Система досягнень реалізована через окрему глобальну колекцію «badges» та підколекцію «userBadges». Колекція «badges» містить опис усіх доступних досягнень у системі: умови отримання, тип досягнення, іконку та порядок відображення. У підколекції «userBadges» зберігається інформація про досягнення конкретного користувача, дата їх відкриття та стан перегляду нового бейджа.

Побудована фізична структура бази даних дозволяє організувати збереження інформації відповідно до особливостей мобільного застосунку та структури «Firebase Firestore». Такий підхід спрощує синхронізацію даних між пристроями, зменшує дублювання інформації та забезпечує швидкий доступ до даних користувача під час роботи застосунку.

Висновки до розділу 2

У другому розділі було спроєктовано інформаційне забезпечення застосунку PieceTime. Основну увагу приділено структурі даних, які потрібні для роботи фокус-сесій, статистики, налаштувань, користувацьких режимів і системи досягнень.

Було сформовано логічну модель даних, у якій центральною сутністю є користувач. Саме з ним пов’язані сесії, налаштування, власний режим, загальні статистичні показники та отримані досягнення. Така структура відповідає реальній логіці застосунку, оскільки всі основні дані PieceTime прив’язані до конкретного акаунта користувача.

Для збереження даних було обрано «Firebase Firestore». Це рішення підходить для мобільного застосунку, тому що дозволяє організувати дані у вигляді колекцій і документів, підтримує синхронізацію між пристроями та добре поєднується з «Firebase Authentication». Окремо було враховано потребу в автономній роботі: якщо результат фокус-сесії не може бути одразу записаний у хмарну базу, він тимчасово зберігається локально та синхронізується після відновлення з’єднання.

У розділі також було описано фізичну структуру бази даних «Firestore». Дані користувача організовано навколо колекції «users», а пов’язані записи винесено в окремі підколекції: «sessions», «settings», «presets» і «userBadges». Такий підхід дозволяє не змішувати різні типи даних, швидко отримувати потрібну інформацію та окремо оновлювати статистику, налаштування або історію сесій.

Отже, у другому розділі було визначено, як саме застосунок зберігає та організовує дані. Сформована структура інформаційної бази стала основою для реалізації синхронізації, статистики, системи прогресу та відкладеного збереження результатів фокус-сесій.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Абстракції предметної області

Під час проєктування програмного забезпечення застосунку PieceTime необхідно було визначити основні абстракції предметної області, які описують логіку проходження фокус-сесій, керування таймером, систему досягнень та взаємодію користувача із застосунком. Виділення таких абстракцій дозволяє ще до детальної реалізації програмних модулів визначити ключові об’єкти системи, їх відповідальність та основні дії, які вони виконують.

На відміну від UML-моделей предметної області, побудованих у першому розділі, даний етап уже орієнтований не лише на сценарії роботи користувача, а й на структуру майбутнього програмного забезпечення. Основна увага приділяється тим сутностям, які надалі будуть використовуватись під час реалізації логіки застосунку, роботи таймера, керування сесіями та формування прогресу користувача.

На рис. 3.1 наведено UML-модель основних абстракцій предметної області застосунку PieceTime.

Рис. 3.1 UML-модель основних абстракцій предметної області застосунку PieceTime

Центральною абстракцією системи є «Фокус-сесія». Саме вона об’єднує логіку проходження циклів фокусування та перерв, зберігає поточний стан сесії, вибраний режим роботи та інформацію про активну фазу. У межах цієї абстракції реалізуються дії запуску сесії, завершення фази, дострокового завершення та переходу між етапами роботи таймера.

Для опису параметрів роботи використовується абстракція «Режим». Вона визначає тривалість фази фокусу, тривалість перерви та кількість циклів. У застосунку підтримуються як готові режими роботи, так і власний користувацький режим, який можна окремо налаштовувати та повторно використовувати під час запуску нових сесій.

Окремо виділено абстракцію «Фаза». Вона використовується для опису окремого етапу проходження сесії. У системі існують фази фокусу та перерви, кожна з яких має власну тривалість і стан виконання. Поділ сесії на окремі фази дозволяє коректно реалізувати циклічну роботу таймера та автоматичний перехід між етапами.

Робота з часовими параметрами винесена в абстракцію «Таймер». Вона відповідає за запуск відліку часу, паузу, продовження роботи та оновлення залишку часу під час активної сесії. Окреме виділення таймера дозволяє відокремити логіку обробки часу від загальної логіки проходження фокус-сесії.

Абстракція «Користувач» описує дані, пов’язані з активністю користувача у системі: кількість завершених сесій, накопичений досвід, загальний час фокусування та отримані досягнення. Саме з цією абстракцією пов’язане оновлення прогресу після завершення чергової фокус-сесії.

Для підтримки елементів гейміфікації у системі використовується абстракція «Досягнення». Вона визначає умови отримання окремих бейджів та дозволяє перевіряти прогрес користувача після завершення сесій. Такий підхід дозволяє зробити систему мотивації незалежною від основної логіки таймера.

Окремо у моделі представлено абстракцію «Сповіщення». Вона не є основною сутністю фокус-сесії, але підтримує її проходження з боку користувацької взаємодії. Сповіщення використовуються для повідомлення про

завершення фази або сесії, а також враховують параметри звуку та вібрації. У реалізації ця логіка винесена в окремий сервіс, тому її доцільно показати як окрему допоміжну абстракцію.

Побудована модель абстракцій дозволила визначити основні складові програмної системи ще до етапу детального моделювання структури класів та взаємодії об’єктів. Це спростило подальше проєктування програмних модулів та допомогло розділити відповідальність між окремими частинами системи.

3.2 Моделювання структури та взаємодії об’єктів

Після визначення основних абстракцій предметної області наступним етапом стало моделювання структури об’єктів застосунку та взаємодії між ними. На цьому етапі абстрактні сутності були деталізовані у вигляді класів із визначенням їх властивостей, основних операцій та зв’язків.

Для мобільного застосунку PieceTime важливо було не лише визначити окремі об’єкти, а й правильно організувати їх взаємодію між собою. Логіка фокус-сесії охоплює роботу таймера, проходження фаз, оновлення статистики користувача, перевірку досягнень, збереження налаштувань та підтримку сповіщень. Через це структура системи була побудована навколо центрального об’єкта фокус-сесії, який об’єднує інші компоненти.

Під час побудови моделі потрібно було врахувати, що не всі об’єкти мають однакову роль у системі. Частина з них описує основний процес проходження сесії, наприклад фокус-сесія, режим, фаза та таймер. Інші об’єкти відповідають за результат роботи користувача: статистику, XP, рівень і досягнення. Окремо виділяються допоміжні елементи, які впливають на поведінку застосунку, зокрема налаштування та сповіщення. Такий поділ дозволяє не змішувати логіку таймера з логікою збереження прогресу або параметрами інтерфейсу.

На рис. 3.2 наведено діаграму класів, що ілюструє структуру та взаємодію об’єктів застосунку PieceTime.

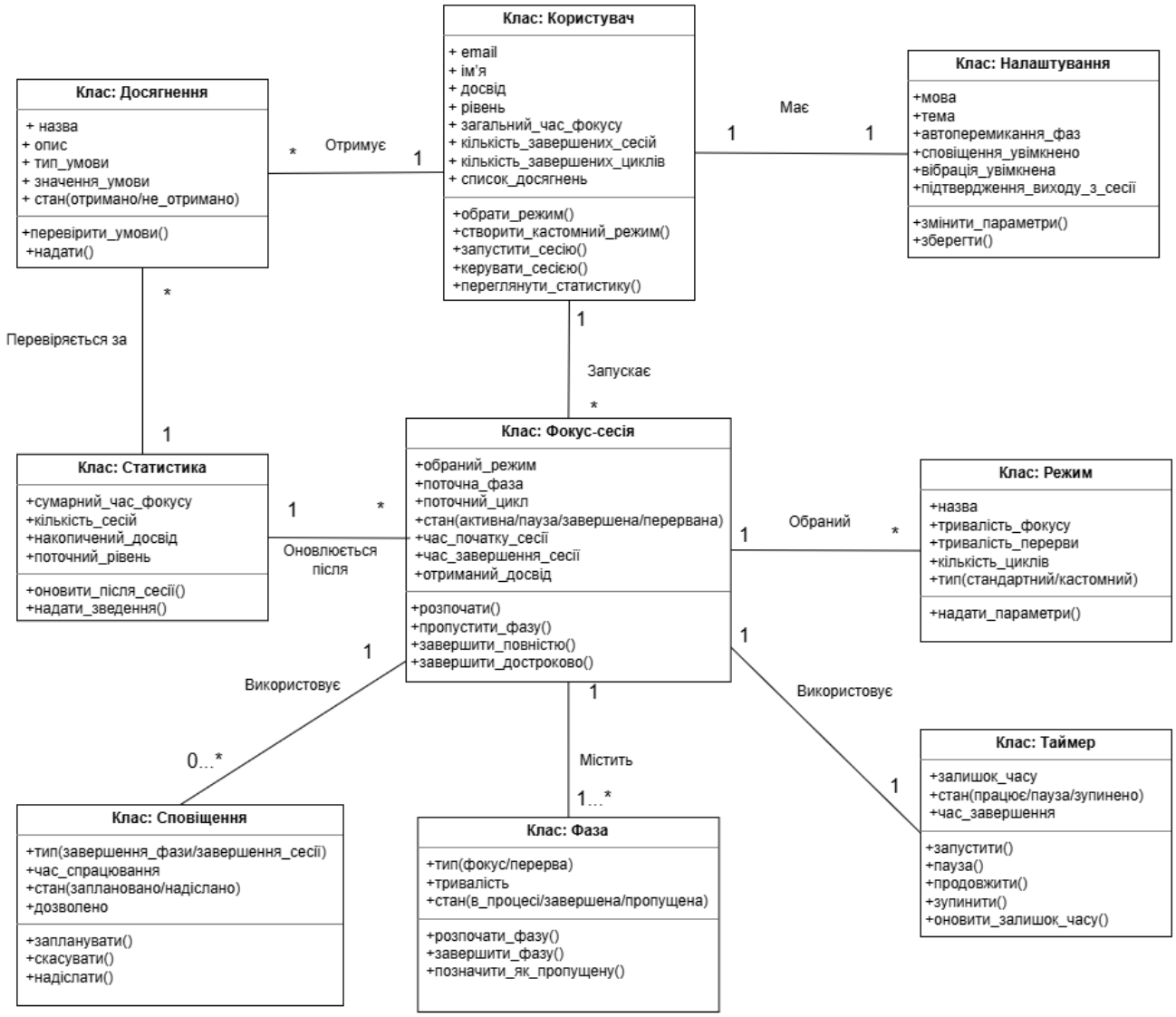

Рис. 3.2 Діаграма класів, що ілюструє структуру та взаємодію об’єктів застосунку PieceTime

У центрі діаграми розташований клас «Фокус-сесія». Саме він відповідає за проходження користувачем циклів фокусування та перерв. Об’єкт сесії зберігає інформацію про обраний режим, поточну фазу, номер циклу, стан проходження та час початку і завершення роботи. Окремо передбачено методи запуску, пропуску фаз, повного завершення або дострокового припинення сесії.

Клас «Користувач» пов’язаний із фокус-сесіями через асоціацію запуску сесії. Один користувач може запускати багато сесій, а також накопичувати досвід, завершені цикли та отримані досягнення. У межах моделі користувач також взаємодіє з налаштуваннями застосунку та статистикою активності.

Для керування параметрами проходження використовується клас «Режим». Він містить тривалість фокусу, тривалість перерви, кількість циклів та тип режиму. У системі підтримуються як готові режими, так і власні користувацькі налаштування. Одна фокус-сесія використовує один режим, але той самий режим може повторно застосовуватись у різних сесіях.

Проходження сесії складається з окремих фаз, тому між класами «Фокус-сесія» та «Фаза» передбачено зв’язок включення. Кожна фаза містить власний тип, тривалість і стан виконання. Це дозволяє окремо керувати фазами фокусування та перерви, а також реалізувати механізми пропуску або автоматичного переходу між ними.

Для відліку часу використовується клас «Таймер». Він відповідає за запуск, паузу, продовження та оновлення залишку часу. У структурі системи таймер не існує окремо від фокус-сесії, а використовується нею під час проходження циклів.

Окремо на діаграмі представлено клас «Статистика». Його призначенням є накопичення сумарних показників активності користувача: загального часу фокусування, кількості сесій, рівня та досвіду. Після завершення кожної сесії статистика оновлюється відповідно до отриманого результату.

Система досягнень реалізована через клас «Досягнення». Він містить умови отримання нагороди, її тип та поточний стан. Після оновлення статистики система перевіряє виконання умов і, за необхідності, відкриває нові досягнення користувача.

Для підтримки взаємодії із користувачем під час проходження сесії у структурі також передбачено клас «Сповіщення». Він використовується для планування та надсилання повідомлень про завершення фаз або всієї сесії. Наявність окремого класу для сповіщень дозволяє відокремити логіку таймера від механізму взаємодії з системними повідомленнями мобільного пристрою.

Клас «Налаштування» відповідає за параметри роботи застосунку. У ньому зберігаються мовні параметри, тема інтерфейсу, стан сповіщень, вібрації та

автоматичного переходу між фазами. Такий поділ дозволяє ізолювати користувацькі налаштування від логіки проходження фокус-сесій.

Побудована діаграма класів дозволила сформувати загальну структуру прикладного програмного забезпечення та визначити відповідальність окремих об’єктів системи. Це стало основою для подальшої організації програмних модулів, реалізації логіки взаємодії між компонентами та побудови архітектури застосунку PieceTime.

3.3 Організаційна структура програмного забезпечення

Після визначення основних класів та взаємодій між ними необхідно організувати програмне забезпечення у вигляді окремих логічних частин. Такий поділ дозволяє зменшити залежність між компонентами системи, спростити підтримку коду та зробити структуру застосунку більш зрозумілою під час подальшої розробки.

Для застосунку PieceTime програмна структура була побудована з урахуванням особливостей «Flutter» та багаторівневої організації мобільних застосунків. Основна логіка системи не змішується з інтерфейсом користувача або роботою з «Firebase». Кожен рівень відповідає лише за власну частину функціоналу.

Такий поділ є важливим для PieceTime, оскільки застосунок одночасно працює з кількома різними процесами: авторизацією користувача, таймером, локальними сповіщеннями, фоновою роботою, статистикою, досягненнями, налаштуваннями та синхронізацією даних. Якщо ці частини не розділяти, зміна одного модуля може впливати на інші частини системи. Тому програмна структура була організована так, щоб інтерфейс, бізнес-логіка, доступ до даних і системні сервіси мали окрему відповідальність.

На рис. 3.3 наведено організаційну структуру програмного забезпечення застосунку PieceTime.

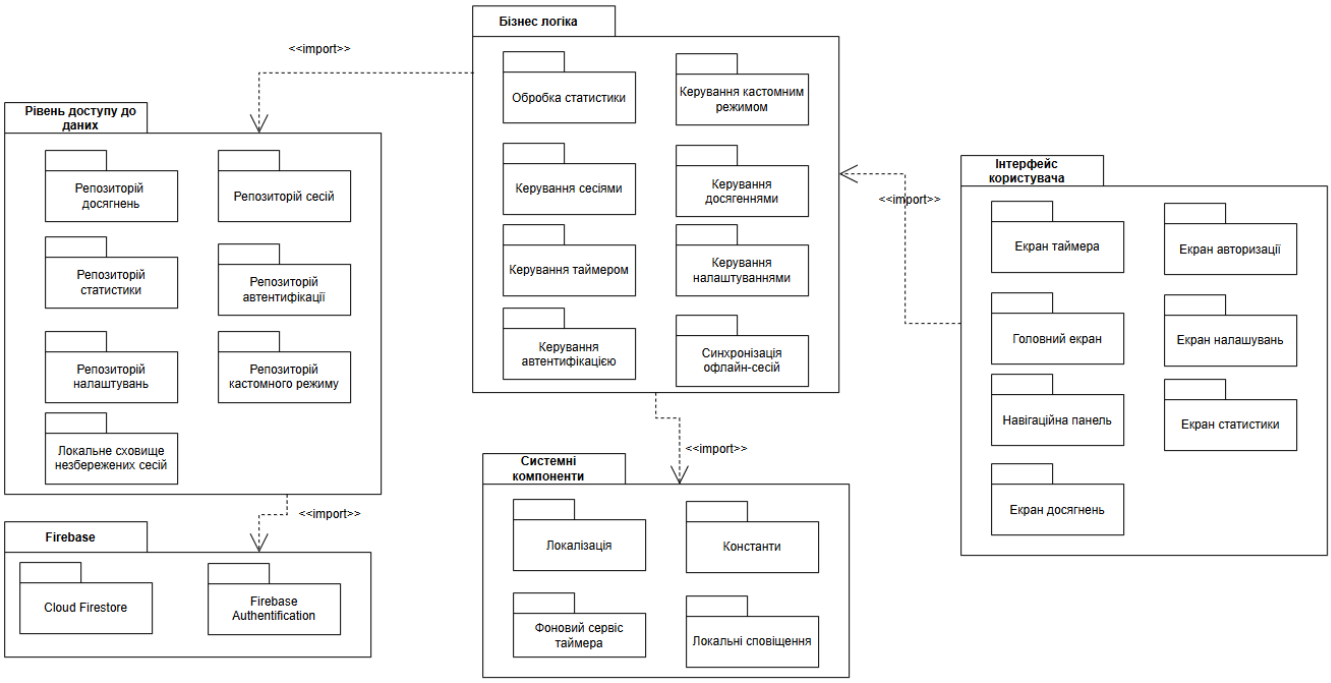

Рис. 3.3 Організаційна структура програмного забезпечення застосунку PieceTime

У структурі системи окремо виділено рівень доступу до даних, бізнес-логіку, інтерфейс користувача, системні компоненти та «Firebase» як зовнішній сервіс збереження даних і автентифікації.

Рівень доступу до даних відповідає за роботу з репозиторіями, локальним збереженням інформації та отриманням даних із «Firebase». Саме через цей рівень виконується доступ до статистики користувача, історії фокус-сесій, досягнень, налаштувань та кастомних режимів. Окремо було виділено локальне сховище незбережених сесій, яке використовується для підтримки офлайн-режиму та подальшої синхронізації після відновлення з’єднання.

Бізнес-логіка містить основні механізми роботи застосунку. Тут реалізовується керування сесіями, таймером, досягненнями, налаштуваннями та автентифікацією користувача. Окремо виділено модуль обробки статистики, який відповідає за оновлення накопичених показників після завершення фокус-сесії. Також у цьому рівні реалізовано синхронізацію офлайн-сесій та підтримку кастомних режимів роботи.

Інтерфейс користувача містить окремі екрани застосунку та навігаційні елементи. До цього рівня входять головний екран, екран таймера, статистики, досягнень, налаштувань та авторизації. Такий поділ дозволяє ізолювати UI від внутрішньої логіки системи та спростити зміну інтерфейсу без впливу на інші частини застосунку.

Системні компоненти містять допоміжний функціонал, необхідний для стабільної роботи застосунку. До них належать локалізація, константи, локальні сповіщення та фоновий сервіс таймера. Окреме винесення таких компонентів дозволяє не перевантажувати бізнес-логіку службовими механізмами.

«Firebase» використовується як зовнішній backend-сервіс. «Cloud Firestore» застосовується для збереження статистики, сесій, досягнень та інших даних користувача, а «Firebase Authentication» забезпечує механізм авторизації та ідентифікації користувачів.

Побудована структура дозволяє розділити відповідальність між компонентами системи, спростити підтримку застосунку та забезпечити більш зрозумілу організацію програмного коду під час реалізації програмних модулів.

3.4 Компонентна структура системи

Після побудови організаційної структури програмного забезпечення необхідно визначити взаємодію основних програмних компонентів системи. Для цього було побудовано діаграму компонентів, яка дозволяє показати залежності між окремими модулями застосунку, зовнішніми сервісами та системними механізмами.

Компонентна структура PieceTime була сформована таким чином, щоб основні частини системи могли працювати незалежно одна від одної. Це дозволяє спростити підтримку проєкту, ізолювати окремий функціонал та зменшити кількість прямих залежностей між модулями.

На рис. 3.4 наведено діаграму компонентів застосунку PieceTime.

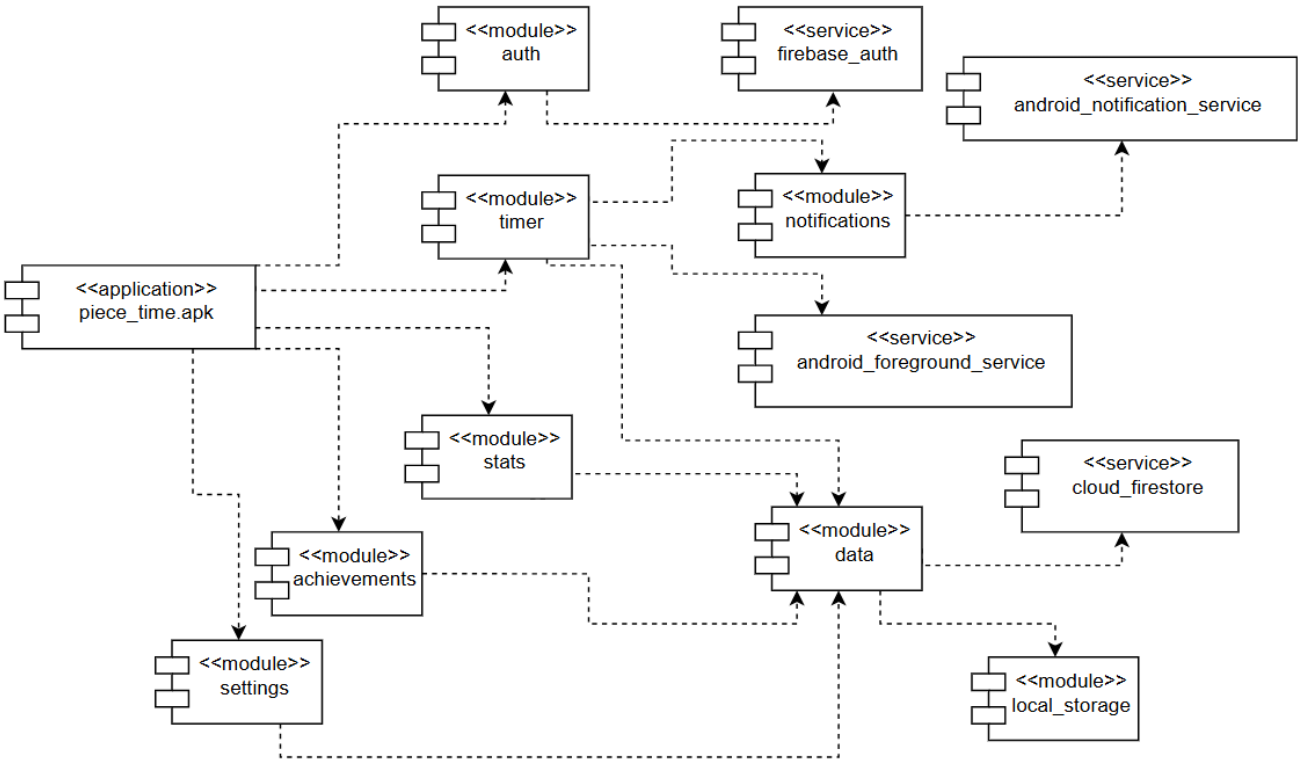

Рис. 3.4 Діаграма компонентів застосунку PieceTime

У центрі структури знаходиться мобільний застосунок «piece_time.apk», який об’єднує всі внутрішні модулі системи та координує їхню взаємодію. Основний функціонал застосунку був розділений на окремі компоненти відповідно до їх призначення.

Компонент «auth» відповідає за авторизацію користувача та взаємодію з сервісом «firebase_auth». Через цей модуль виконується реєстрація, вхід користувача та отримання інформації про поточний акаунт.

Компонент «timer» реалізує основну логіку роботи фокус-сесії. Саме він керує переходами між фазами фокусу та перерви, взаємодіє з модулем сповіщень та передає дані до інших частин системи після завершення сесії.

Модуль «notifications» використовується для створення локальних сповіщень та роботи з «Android»-сервісами. Для цього компонент взаємодіє з «android_notification_service» та «android_foreground_service». Такий підхід дозволяє підтримувати роботу таймера навіть після згортання застосунку або блокування екрана пристрою.

Компонент «stats» відповідає за формування статистики користувача, накопичення досвіду, підрахунок завершених циклів та оновлення загального прогресу після завершення фокус-сесій.

Модуль «achievements» реалізує перевірку умов отримання досягнень та їх відкриття після оновлення прогресу користувача. Він взаємодіє з компонентом «data», через який отримуються необхідні дані про статистику та попередні результати сесій.

Компонент «settings» використовується для роботи з налаштуваннями застосунку: темою оформлення, мовою інтерфейсу, локальними сповіщеннями, автоматичним перемиканням фаз та іншими параметрами.

Окремо виділено компонент «data», який виконує роль центрального рівня доступу до даних. Через нього відбувається взаємодія з «cloud_firestore» та «local_storage». Це дозволяє ізолювати бізнес-логіку від конкретного способу збереження інформації та підтримувати як онлайн-, так і офлайн-режим роботи застосунку.

Компонент «local_storage» використовується для локального збереження незавершених або не синхронізованих сесій. Після відновлення підключення ці дані передаються до «cloud_firestore» та синхронізуються з акаунтом користувача.

Побудована компонентна структура дозволяє розділити функціонал застосунку на незалежні частини та забезпечує більш гнучку організацію програмного забезпечення. Такий підхід спрощує підтримку системи, тестування окремих модулів та подальше розширення функціоналу застосунку.

3.5 Вибір інструментарію для створення прикладного програмного забезпечення

Для реалізації мобільного застосунку PieceTime було обрано набір інструментів, які дозволяють підтримувати роботу таймера, збереження прогресу користувача, синхронізацію даних між пристроями та побудову адаптивного інтерфейсу. Під час вибору технологій основна увага приділялась

не тільки швидкості розробки, а й можливості стабільно підтримувати роботу фокус-сесій у реальних умовах використання мобільного застосунку.

Основним середовищем розробки було обрано «Flutter» [9]. Даний фреймворк дозволяє створювати мобільні застосунки з єдиною кодовою базою та забезпечує побудову інтерфейсу через систему віджетів. Для PieceTime це було важливо через велику кількість екранів і станів інтерфейсу: таймер, статистика, налаштування, авторизація, досягнення та екрани вибору режимів. «Flutter» також дозволяє гнучко керувати анімаціями, перебудовою екранів та адаптацією інтерфейсу під різні розміри пристроїв. Під час побудови інтерфейсу також враховувалися підходи «Material Design 3», зокрема використання зрозумілої навігації, карток, кнопок і тем оформлення [23].

Мовою програмування було обрано «Dart» [10], оскільки вона є основною мовою «Flutter» та добре підходить для побудови реактивного інтерфейсу. «Dart» використовується для реалізації логіки таймера, керування станами застосунку, обробки статистики та взаємодії з «Firebase». Для підтримки української та англійської мов інтерфейсу в застосунку використано механізми локалізації «Flutter» [22].

Для організації авторизації користувачів було використано «Firebase Authentication» [11]. Даний сервіс дозволяє реалізувати реєстрацію, вхід у систему та підтримку облікових записів без необхідності створення власного серверного модуля автентифікації. Це дало можливість зосередитись на логіці самої системи фокус-сесій, а не на окремій реалізації механізму авторизації.

Для збереження даних застосунку було використано «Cloud Firestore» [12]. Вибір документо-орієнтованої бази даних пов’язаний із структурою даних PieceTime. Користувач має власні сесії, налаштування, статистику та досягнення, які логічно групуються у вкладені колекції. «Firestore» також забезпечує синхронізацію між пристроями та дозволяє працювати із даними у реальному часі.

Для керування станом застосунку було використано «Riverpod» [14]. Даний інструмент використовується для централізованого керування таймером,

статистикою, авторизацією, налаштуваннями та іншими частинами системи. Це дозволяє уникати хаотичної передачі даних між екранами та спрощує оновлення інтерфейсу при зміні стану застосунку.

Окрему увагу під час розробки було приділено підтримці роботи застосунку без стабільного підключення до мережі. Для цього використовується локальне збереження даних через «SharedPreferences» [15]. Незбережені фокус-сесії тимчасово записуються у локальне сховище, а після відновлення підключення синхронізуються з «Firestore». Такий підхід дозволяє не втрачати прогрес користувача навіть при відсутності інтернету.

Для сповіщень у застосунку використано бібліотеку «flutter_local_notifications» [16]. Вона потрібна для того, щоб користувач отримував повідомлення про завершення фази фокусу, перерви або всієї сесії. Окремо було використано механізм фонового сервісу через «flutter_foreground_task» [17]. Завдяки цьому таймер може продовжувати роботу навіть тоді, коли користувач згорнув застосунок або заблокував екран. Для PieceTime це важливо, бо фокус-сесія має тривати без прив’язки до того, чи відкритий застосунок на екрані в даний момент.

Середовищем розробки було обрано «Android Studio» [20]. Воно використовувалось для написання коду, тестування інтерфейсу, емуляції «Android»-пристроїв, аналізу журналів системи та створення APK-збірок застосунку.

Для збереження історії змін проєкту використовувався «GitHub» [21]. Це дозволяло фіксувати окремі етапи розробки, зберігати різні версії коду та за потреби повертатися до попереднього стану проєкту. Використання системи контролю версій було важливим під час поступового додавання функціоналу таймера, статистики, досягнень та синхронізації даних.

Обраний набір інструментів дозволив реалізувати PieceTime як мобільний застосунок із підтримкою авторизації, синхронізації даних, офлайн-режиму, локальних сповіщень та стабільної роботи таймера у фоновому режимі. Крім

цього, використані технології добре поєднуються між собою та спрощують подальше розширення функціоналу системи.

3.6 Алгоритмізація та програмування програмних модулів

Етап алгоритмізації у застосунку PieceTime використовується для опису основної логіки роботи фокус-сесій, переходів між фазами таймера, обробки циклів, нарахування XP, синхронізації даних та збереження прогресу користувача. Тут важливо не тільки показати окремі функції, а й пояснити послідовність дій між модулями системи. Від цієї логіки залежить стабільність таймера, коректність статистики, робота досягнень та поведінка застосунку без підключення до мережі.

У цьому підрозділі алгоритми розглядаються не як окремі ізольовані дії, а як послідовність станів, через які проходить застосунок під час роботи користувача. Для PieceTime це особливо важливо, оскільки таймер, статистика, XP, досягнення та синхронізація даних залежать від одного результату фокус-сесії. Якщо один із цих етапів буде виконано неправильно, система може показати некоректний підсумок, не зарахувати прогрес або втратити результат під час відсутності мережі.

Тому основна увага під час реалізації була приділена не тільки запуску таймера, а й контролю переходів між станами. Сесія має чітко визначений початок, поточну фазу, номер циклу, накопичений прогрес і момент завершення. Такий підхід дозволяє уникнути ситуацій, коли інтерфейс показує один стан, а внутрішня логіка вже перейшла до іншого етапу.

Першим ключовим алгоритмом є запуск фокус-сесії. Алгоритм запуску фокус-сесії наведено у додатку Б на рис. Б.1.

На діаграмі показано процес підготовки сесії перед запуском таймера. Після відкриття головного екрана застосунок отримує список доступних режимів роботи. Користувач може обрати готовий пресет або налаштувати власний режим вручну. У випадку кастомного режиму додатково зберігаються параметри користувацької конфігурації.

Після вибору режиму параметри передаються до модуля таймера, де створюється початковий стан фокус-сесії. Далі система встановлює першу фазу фокусу, відкриває екран таймера та запускає відлік часу. Такий підхід дозволяє ізолювати етап підготовки сесії від основної логіки таймера та спрощує подальше керування станами.

Наступним важливим алгоритмом є проходження фокус-сесії, оскільки саме він визначає основний цикл роботи застосунку.

Алгоритм проходження фокус-сесії наведено у додатку Б на рис. Б.2.

Після запуску сесії система отримує її поточний стан, визначає активну фазу та запускає таймер. Під час роботи відбувається постійне оновлення залишку часу на екрані таймера. Коли час поточної фази завершується, система перевіряє тип активної фази.

Якщо завершується фаза фокусу, система фіксує завершений фокус-блок та переходить до фази перерви. Якщо ж завершується фаза перерви, це означає завершення повного циклу. У такому випадку фіксується завершений цикл та виконується перевірка кількості циклів, що залишились.

Під час проходження фокус-сесії система повинна розрізняти завершення окремої фази та завершення всього циклу. Фаза фокусу впливає на кількість завершених фокус-блоків і подальше нарахування XP, а фаза відпочинку завершує цикл та переводить сесію до наступного повторення. Через це перехід між фазами не можна зводити лише до зміни тексту на екрані. Він також змінює внутрішні лічильники, стан таймера та дані, які потім використовуються для формування підсумку.

У реалізації важливо було врахувати й крайні випадки. Наприклад, користувач може пропустити поточну фазу або завершити сесію раніше. У такому випадку система не повинна зараховувати ті частини прогресу, які фактично не були виконані. Тому результат сесії формується не тільки на основі початкового режиму, а й на основі реального проходження: скільки фокус-блоків і циклів було завершено на момент зупинки.

Якщо всі цикли завершені, дані передаються до алгоритму завершення сесії. Інакше система переходить до наступного циклу та повторно запускає фазу фокусу. Така організація дозволяє уникнути дублювання логіки таймера та підтримує циклічну структуру роботи застосунку.

Алгоритм завершення та збереження фокус-сесії наведено у додатку Б на рис. Б.3.

Після завершення останнього циклу система отримує дані завершеної сесії та формує підсумковий результат. На цьому етапі виконується остаточний розрахунок XP на основі кількості завершених фокус-блоків, циклів та факту повного завершення сесії.

Під час проходження фокус-сесії система не виконує остаточне нарахування XP. На цьому етапі фіксуються проміжні результати: кількість завершених фокус-блоків, кількість завершених циклів і стан завершення сесії. Ці дані потрібні для того, щоб після завершення роботи правильно сформувати підсумок активності користувача.

Після завершення сесії система має визначити, який прогрес потрібно зарахувати користувачеві. Для цього враховуються не лише повністю завершені сесії, а й частково виконана робота. Якщо користувач пройшов частину фокус-блоків або окремі цикли, цей результат також повинен бути врахований у підсумку. Завдяки цьому прогрес користувача не втрачається навіть тоді, коли сесію було завершено достроково.

Остаточний розрахунок XP виконується після формування результату сесії. Отримане значення використовується для оновлення загального прогресу користувача, статистики активності та подальшої перевірки умов отримання досягнень.

Нарахування XP виконує не тільки роль числового підсумку, а й пов’язує таймер із модулем гейміфікації. Система враховує завершені хвилини фокусу, завершені цикли та повністю завершену сесію. Завдяки цьому користувач отримує винагороду не лише за ідеальне проходження всього режиму, а й за

частковий прогрес. Це важливо для реального використання застосунку, бо не кожна сесія завершується за планом.

Після оновлення XP система може перевіряти умови відкриття досягнень. У цій частині логіка гейміфікації залежить від уже збережених або розрахованих показників: загальної кількості сесій, хвилин фокусу, циклів, досвіду та серії активних днів. Тому перевірка досягнень виконується після формування результату сесії, а не перед ним. Це дозволяє враховувати саме актуальний стан користувача після завершення роботи таймера.

Після розрахунку XP створюється запис фокус-сесії. Далі система перевіряє наявність підключення до мережі. Якщо інтернет доступний, дані зберігаються у «Cloud Firestore», оновлюється статистика користувача та виконується перевірка досягнень. Якщо умови відкриття нового бейджа виконані, застосунок відображає відповідне повідомлення користувачу.

У випадку відсутності мережі система не припиняє роботу та не втрачає дані сесії. Запис зберігається у локальній черзі та позначається для подальшої синхронізації. Такий механізм був реалізований для підтримки офлайн-режиму та стабільної роботи застосунку навіть при нестабільному інтернет-з’єднанні.

Механізм збереження результату побудований так, щоб завершення сесії не залежало повністю від наявності мережі. Для користувача сесія вважається завершеною одразу після формування підсумку, а вже після цього система визначає спосіб збереження. Якщо з’єднання доступне, результат передається до хмарної бази даних і використовується для оновлення статистичних показників. Якщо з’єднання відсутнє, запис не відкидається, а тимчасово зберігається локально.

Такий підхід важливий саме для мобільного застосунку. Користувач може завершити фокус-сесію в дорозі, у приміщенні зі слабким зв’язком або після тимчасового вимкнення інтернету. У цих умовах застосунок не повинен втрачати результат. Локальна черга незбережених сесій дозволяє відкласти синхронізацію до моменту, коли з’єднання буде відновлено, і при цьому не змушує користувача повторно запускати або вручну зберігати сесію.

Окрім побудови блок-схем, на етапі програмування виконувалась реалізація окремих програмних модулів. Особлива увага приділялась розділенню логіки між модулями таймера, статистики, досягнень, локального збереження та синхронізації даних. Це дозволило уникнути надмірної залежності між компонентами та спростило подальшу підтримку проєкту.

Описані алгоритми стали основою для реалізації програмних модулів застосунку PieceTime. Після визначення послідовності роботи системи було важливо перенести цю логіку в код так, щоб окремі частини застосунку не дублювали відповідальність одна одної. Саме тому логіку налаштування режимів, запуску таймера, переходів між фазами, збереження результатів, розрахунку XP та визначення рівня користувача було розділено між окремими модулями.

Далі розглянуто фрагменти програмної реалізації, які відповідають за ключові сценарії роботи застосунку. Основна увага приділяється не всім файлам проєкту, а тим частинам коду, які безпосередньо показують, як параметри фокус-сесії передаються між екранами, як змінюється стан таймера, як формується результат сесії та як цей результат впливає на прогрес користувача.

Першим варто розглянути реалізацію користувацького режиму. Цей режим потрібний для того, щоб користувач міг самостійно задати тривалість фокусу, тривалість перерви та кількість циклів. На відміну від готових режимів, його параметри не є фіксованими, тому після зміни вони мають бути передані назад у логіку застосунку та використані під час запуску фокус-сесії. Фрагмент реалізації налаштування власного режиму наведено на рис. 3.5.

```
class _CustomPresetSheetState extends State<CustomPresetSheet> {
  late int focus;
  late int rest;
  late int cycles;
  @override
  void initState() {
    super.initState();
    focus = widget.initial.focusMin;
    rest = widget.initial.breakMin;
    cycles = widget.initial.cycles;
  }
  void _savePreset() {
    Navigator.of(context).pop<FocusPreset>(
      FocusPreset(
        id: widget.initial.id,
        name: widget.initial.name,
        focusMin: focus,
        breakMin: rest,
        cycles: cycles,
      ),
    ); } }
```

Рис. 3.5 Фрагмент реалізації налаштування власного режиму

У наведеному фрагменті показано, що екран налаштування власного режиму отримує початкові параметри з уже наявного об’єкта «FocusPreset». Після зміни значень користувачем формується новий об’єкт режиму з оновленою тривалістю фокусу, перерви та кількістю циклів. Далі цей об’єкт передається назад на головний екран і може бути використаний контролером таймера під час запуску сесії.

Після вибору режиму фокус-сесії система переходить до запуску таймера. На цьому етапі важливо не тільки почати відлік часу, а й зберегти момент старту сесії, запланувати локальне сповіщення та запустити фоновий сервіс. Це потрібно для того, щоб таймер продовжував працювати після згортання застосунку або блокування екрана пристрою.

Фрагмент реалізації запуску фокус-сесії та фонового таймера наведено у додатку В у лістингу В.1.

У лістингу В.1 метод «start()» спочатку перевіряє, чи таймер уже не запущений. Далі фіксується час початку сесії та розраховується момент завершення поточної фази. Після цього стан таймера оновлюється через «copyWith», а для щосекундного відліку створюється «Timer.periodic».

Окремо в методі формується текст локального сповіщення. Якщо сповіщення увімкнені в налаштуваннях користувача, система планує повідомлення про завершення фази. Наприкінці запускається «focusFgServiceProvider», який підтримує роботу таймера у фоновому режимі. Завдяки цьому фокус-сесія не прив’язана лише до відкритого екрана застосунку і може продовжуватися після згортання програми.

Після запуску таймера система повинна коректно змінювати стан фокус-сесії залежно від того, яка фаза завершилась. Для цього в контролері таймера використовується окремий метод, який обробляє завершення поточної фази. Він визначає, чи потрібно перейти до перерви, почати новий цикл або завершити всю сесію.

Фрагмент реалізації керування станом фокус-сесії наведено у додатку В у лістингу В.2.

У лістингу В.2 метод «_completePhase()» отримує параметр «rewardUser», який показує, чи потрібно зараховувати завершення фази до прогресу користувача. Це важливо, тому що фаза може завершитися природно після закінчення часу або бути пропущеною вручну.

Якщо поточною фазою є фокус, система переходить до фази перерви та, за потреби, збільшує кількість завершених фокус-блоків. Якщо завершується фаза перерви, система фіксує завершений цикл і перевіряє, чи був він останнім. У разі завершення всіх циклів сесія позначається як завершена. Якщо цикли ще залишаються, система переходить до наступного циклу та знову встановлює фазу фокусу.

Після завершення сесії застосунок має не тільки показати підсумок користувачу, а й зберегти результат у системі. Для цього формується об’єкт «FocusSessionRecord», у якому фіксуються параметри режиму, кількість завершених фокус-блоків, циклів, отриманий XP та час проходження сесії. Після збереження результату система перевіряє, чи відкрилися нові досягнення.

Фрагмент реалізації збереження результату сесії та перевірки досягнень наведено у додатку В у лістингу В.3.

У лістингу В.3 спочатку перевіряється, чи сесія вже не була збережена раніше та чи має вона хоча б якийсь прогрес. Це захищає систему від повторного запису одного й того самого результату. Далі отримується поточний користувач, час початку сесії та розрахований XP.

Після цього створюється об’єкт «FocusSessionRecord», який містить підсумкові дані сесії. Саме цей об’єкт передається до репозиторію історії сесій для збереження. Після успішного збереження викликається перевірка досягнень користувача. Якщо нові бейджі не відкриті, метод повертає порожній список. Якщо умови виконані, повертається список нових досягнень, які потім можуть бути показані користувачу.

Після формування результату сесії застосунок має визначити, скільки XP отримав користувач. Для цього використовується окремий модуль «TimerXpCalculator». Винесення цієї логіки в окремий клас спрощує підтримку системи прогресу, оскільки правила нарахування досвіду не змішуються з кодом екрана таймера або збереженням сесії.

Розрахунок XP виконується на основі фактично завершеного прогресу користувача. Система окремо враховує кількість завершених фокус-блоків, завершених циклів і факт повного проходження сесії. Завдяки цьому користувач отримує досвід не тільки за повністю завершену сесію, а й за частково виконану роботу. Фрагмент реалізації розрахунку XP наведено на рис. 3.6.

```
static int earned({
  required FocusPreset preset,
  required int completedFocusBlocks,
  required int completedCycles,
  required bool isSessionCompleted,
}) {
  final focusXp =
      completedFocusBlocks * preset.focusMin * Constants.xpPerFocusMinute;

  final cycleBonus =
      completedCycles * Constants.xpPerCompletedCycle;

  final sessionBonus =
      isSessionCompleted ? Constants.xpPerCompletedSession : 0;

  return focusXp + cycleBonus + sessionBonus;
}
```

Рис. 3.6 Фрагмент реалізації розрахунку XP

У наведеному фрагменті метод «earned()» розраховує загальну кількість XP, яку користувач отримує після проходження сесії. Розрахунок складається з трьох частин: досвіду за завершені хвилини фокусу, бонусу за завершені цикли та додаткового бонусу за повне завершення сесії.

Кількість XP за фокус визначається множенням кількості завершених фокус-блоків на тривалість фокусу в обраному режимі. Окремо додається бонус за кожен завершений цикл. Якщо сесію пройдено повністю, користувач отримує додатковий бонус. Завдяки цьому система враховує не тільки повне завершення сесії, а й частковий прогрес користувача.

На основі накопиченого XP у застосунку також визначається рівень користувача. XP, отриманий після фокус-сесій, додається до загального прогресу, а далі система перевіряє, якому рівню відповідає поточна кількість досвіду. Це дозволяє показувати користувачеві не тільки окремий результат сесії, а й довгостроковий прогрес у застосунку.

Для визначення кількості XP, необхідної для проходження рівня, використовується формула:

$$XP(level) = 100 + (level - 1)^2 * 50, \quad (1)$$

де level – поточний рівень користувача;

XP(level) – кількість досвіду, яку потрібно набрати для переходу з цього рівня на наступний.

За формулою (1) для першого рівня потрібно 100 XP, для другого – 150 XP, для третього – 300 XP, для четвертого – 550 XP. Тобто з кожним наступним рівнем кількість необхідного досвіду зростає. Завдяки цьому перші рівні користувач отримує швидше, а подальший прогрес потребує більш регулярного проходження фокус-сесій.

Фрагмент реалізації визначення рівня користувача за кількістю XP наведено у додатку В у лістингу В.4.

У лістингу В.4 функція «_xpRequiredForLevel()» визначає, скільки XP потрібно для проходження конкретного рівня. Значення не є сталим, оскільки з кожним наступним рівнем вимога до кількості досвіду збільшується. Це потрібно для того, щоб система прогресу не завершувалась занадто швидко після кількох перших сесій.

Функція «_calculateLevel()» визначає поточний рівень користувача на основі загальної кількості накопиченого XP. Розрахунок починається з першого рівня. Далі система перевіряє, чи достатньо користувач має досвіду для переходу на наступний рівень. Якщо XP вистачає, необхідна кількість віднімається, а рівень збільшується. Цикл повторюється доти, поки залишку XP вже недостатньо для нового переходу.

Окремо реалізовано функції «_xpForLevelStart()» та «_xpForNextLevel()». Вони використовуються для відображення прогресу всередині поточного рівня. Перша функція визначає, скільки XP потрібно було набрати до початку поточного рівня, а друга – скільки XP потрібно для досягнення наступного рівня. Завдяки цьому інтерфейс може показувати не тільки номер рівня, а й заповнення шкали прогресу між поточним і наступним рівнем.

Висновки до розділу 3

У третьому розділі було розглянуто розробку програмного забезпечення PieceTime. Спочатку було визначено основні абстракції предметної області: фокус-сесію, режим, фазу, таймер, користувача, статистику, досягнення, сповіщення та налаштування. Це дозволило розділити логіку системи на зрозумілі частини та визначити відповідальність кожного елемента.

Далі було побудовано структуру взаємодії об’єктів. Центральним елементом системи стала фокус-сесія, оскільки саме вона об’єднує обраний режим, поточну фазу, номер циклу, стан таймера та результат проходження. Окремо було виділено таймер, статистику, досягнення й налаштування, щоб не змішувати різні частини логіки в одному модулі.

Програмна структура застосунку була організована за рівнями: доступ до даних, бізнес-логіка, інтерфейс користувача, системні компоненти та зовнішні сервіси «Firebase». Такий поділ спрощує підтримку коду, зменшує залежність між модулями та дозволяє окремо змінювати інтерфейс, логіку таймера або роботу з даними. Компонентна структура також показала, як між собою взаємодіють модулі авторизації, таймера, статистики, досягнень, налаштувань, локального сховища та хмарної бази даних.

Для реалізації застосунку було обрано «Flutter», «Dart», «Firebase Authentication», «Cloud Firestore», «Riverpod», «SharedPreferences», локальні сповіщення та фоновий сервіс таймера. Цей набір інструментів дозволив реалізувати мобільний застосунок із авторизацією, збереженням даних, синхронізацією, підтримкою офлайн-сценаріїв, фоновою роботою таймера та адаптивним інтерфейсом.

Окремо в розділі було описано ключові алгоритми роботи системи: налаштування власного режиму, запуск фокус-сесії, керування переходами між фазами, збереження результату сесії, перевірку досягнень і розрахунок XP. Також було розглянуто формулу визначення кількості XP, необхідної для проходження рівнів. Вона використовується для поступового збільшення вимог

до прогресу користувача: перші рівні досягаються швидше, а наступні потребують більшої кількості завершених фокус-сесій.

У результаті третього розділу було описано основну програмну реалізацію PieceTime: від архітектури та компонентів до алгоритмів таймера, збереження сесій, нарахування XP, визначення рівня користувача та роботи гейміфікації.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування програмного забезпечення є етапом перевірки роботи системи після її реалізації. Його основне завдання полягає в тому, щоб переконатися, що застосунок виконує передбачені функції, правильно реагує на дії користувача та не втрачає дані під час виконання основних сценаріїв. Для мобільного застосунку важливо перевіряти не тільки окремі екрани, а й повний шлях користувача: від створення облікового запису до запуску основної функції, отримання результату та перегляду збереженого прогресу.

Тестування може проводитися різними способами. Модульне тестування перевіряє окремі функції або класи, інтеграційне – взаємодію між частинами системи, а функціональне – роботу застосунку з погляду користувацьких сценаріїв. Для PieceTime найбільш доцільним є функціональне тестування, оскільки основна цінність застосунку полягає у коректній роботі повного процесу: виборі режиму, проходженні фокус-сесії, завершенні таймера, нарахуванні XP, збереженні результату та відкритті досягнень.

У межах роботи було обрано ручне функціональне тестування. Під час перевірки виконувалися основні дії в інтерфейсі: реєстрація користувача, підтвердження електронної пошти, налаштування власного режиму, запуск готового та власного режимів, перехід між фазами фокусу й перерви, завершення сесії, нарахування XP та перевірка відкритих досягнень.

Тестування проводилося на «Android»-пристрої після встановлення APK-збірки. Для перевірки використовувався тестовий обліковий запис, підключення до «Firebase Authentication» та «Cloud Firestore», а також дозволи «Android», необхідні для роботи сповіщень. Після першого запуску застосунок запитує дозвіл на надсилання сповіщень і використання точних будильників та нагадувань. Без цих дозволів таймер може працювати, але користувач не завжди отримає системне повідомлення про завершення фази або сесії.

Першим було перевірено сценарій реєстрації користувача. Цей етап важливий, оскільки PieceTime зберігає статистику, XP, досягнення та історію фокус-сесій окремо для кожного облікового запису. Користувач вводить ім’я, електронну пошту, пароль і підтвердження пароля, після чого система створює новий акаунт. Екран реєстрації користувача наведено на рис. 4.1.

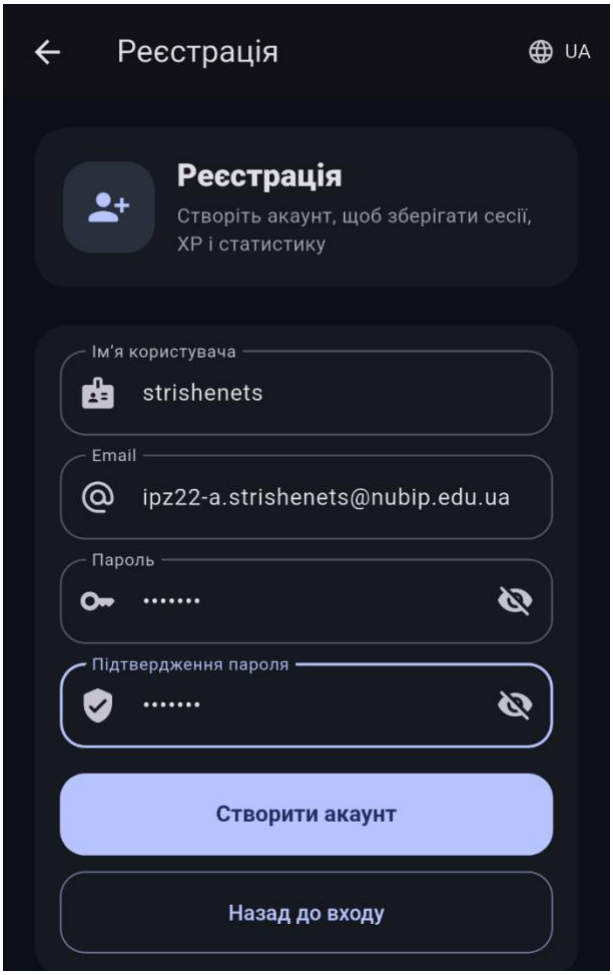

Рис. 4.1 Екран реєстрації користувача, що ілюструє створення нового облікового запису

Після створення акаунта застосунок не переводить користувача одразу до основного екрана. Спочатку система просить підтвердити електронну пошту. Така перевірка потрібна для того, щоб користувач працював із дійсною адресою, а обліковий запис міг коректно використовуватися для подальшого входу, відновлення доступу та синхронізації даних.

Екран підтвердження електронної пошти наведено на рис. 4.2.

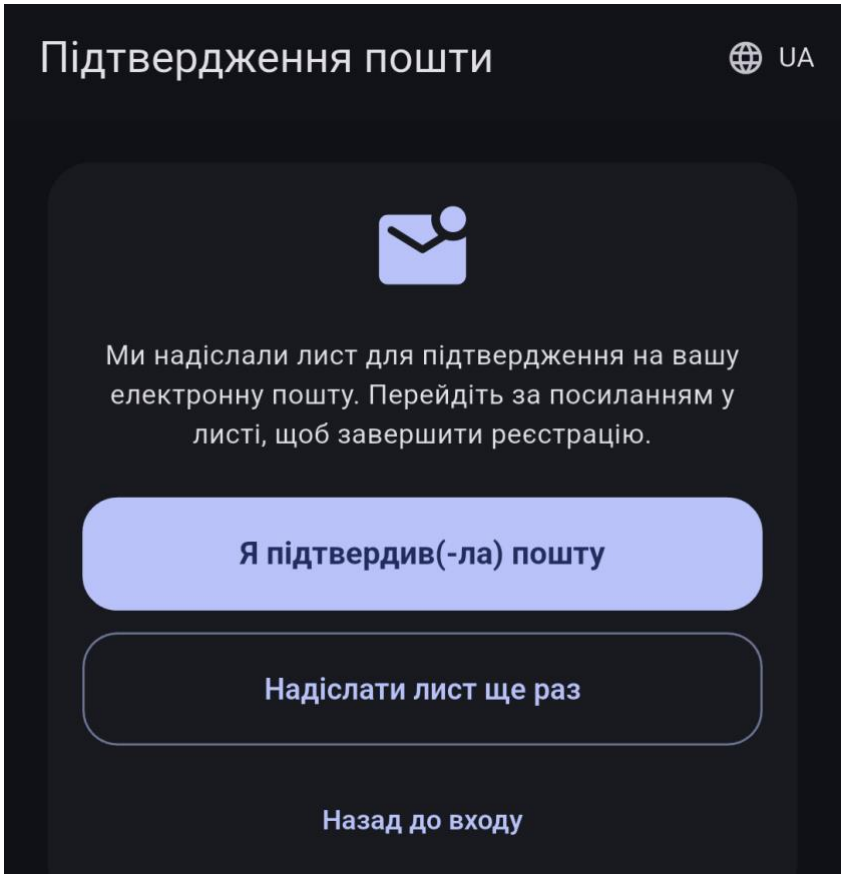

Рис. 4.2 Екран підтвердження електронної пошти, що ілюструє перевірку облікового запису

На екрані підтвердження пошти користувач бачить повідомлення про те, що лист для підтвердження було надіслано на вказану електронну адресу. Після цього користувач має самостійно відкрити поштову скриньку та перейти за посиланням у листі. Якщо натиснути кнопку перевірки в застосунку до підтвердження адреси, система не пропускає користувача далі та повідомляє, що пошта ще не підтверджена.

Після реєстрації на електронну адресу користувача надходить лист із посиланням для підтвердження облікового запису. Наявність такого листа підтверджує, що застосунок коректно взаємодіє з механізмом автентифікації та надсилає запит на підтвердження пошти через «Firebase Authentication». Приклад листа підтвердження наведено на рис. 4.3.

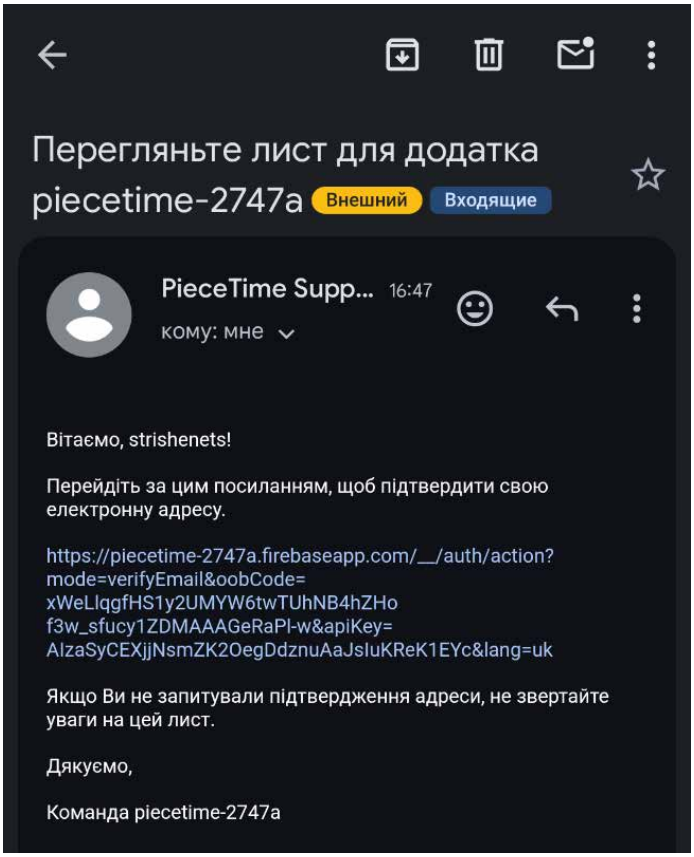

Рис. 4.3 Лист підтвердження електронної пошти для підтвердження електронної адреси

Після переходу за посиланням у листі електронна адреса отримує статус підтвердженої. Далі користувач повертається до застосунку та натискає кнопку перевірки. Якщо обліковий запис уже підтверджено, система допускає користувача до головного екрана PieceTime. Таким чином було перевірено повний сценарій реєстрації: створення акаунта, надсилання листа, перевірку статусу підтвердження та перехід до основної частини застосунку після успішної активації.

Після підтвердження облікового запису було перевірено перехід до головного екрана застосунку. На цьому екрані користувач отримує доступ до основної функції PieceTime – запуску фокус-сесії. Після першого входу в застосунку автоматично відображається режим, встановлений за замовчуванням. Він дозволяє користувачеві одразу почати роботу без обов’язкового попереднього налаштування. Також на головному екрані доступне нижнє меню

навігації, за допомогою якого користувач може перейти до розділів «Головна», «Статистика» та «Налаштування».

Головний екран застосунку з режимом, встановленим за замовчуванням, наведено в додатку Г на рис. Г.1.

Під час перевірки головного екрана зверталася увага не лише на відображення вибраного режиму, а й на загальну логіку взаємодії з інтерфейсом. Було перевірено, що після входу користувач бачить актуальні параметри сесії, може одразу запустити таймер або змінити режим перед початком роботи. Це важливо, оскільки головний екран є початковою точкою основного сценарію використання застосунку.

Окремо було перевірено роботу нижнього меню навігації. Перехід між вкладками не скидає стан головного екрана та дозволяє швидко перейти до перегляду статистики або налаштувань. Така структура навігації робить застосунок зручнішим для щоденного використання, оскільки основні розділи залишаються доступними без додаткових проміжних екранів.

На головному екрані користувач може запустити сесію з поточними параметрами або відкрити список доступних режимів. Під час тестування було перевірено, що кнопки головного екрана коректно реагують на дії користувача: запуск сесії відкриває екран таймера, а вибір режиму відкриває список доступних сценаріїв роботи.

У списку режимів користувач може обрати один із готових варіантів або перейти до налаштування власного режиму. Застосунок підтримує класичний режим, глибокий фокус, легкий старт, власний режим і тестовий режим. Готові режими мають заздалегідь визначені параметри, а власний режим дозволяє самостійно змінити тривалість фокусу, перерви та кількість циклів.

Список доступних режимів фокус-сесії наведено на рис. 4.4.

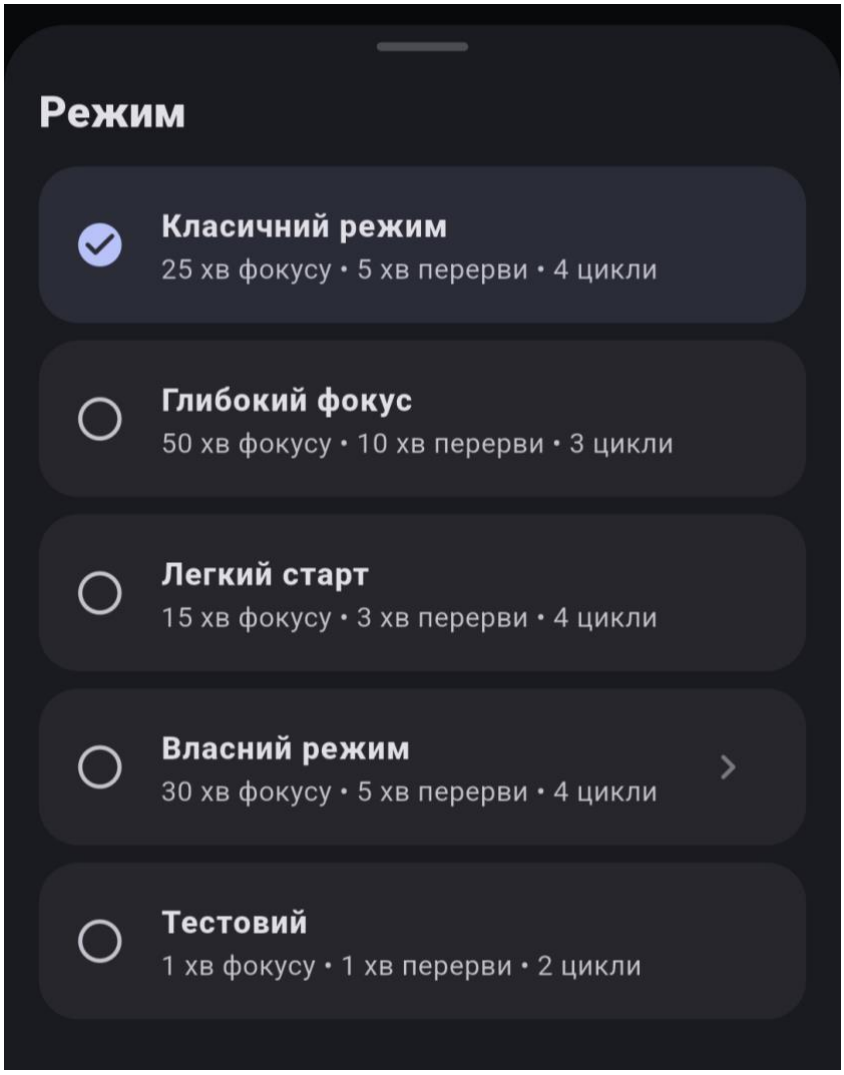

Рис. 4.4 Список режимів фокус-сесії, що ілюструє вибір сценарію роботи таймера

Окремо було перевірено налаштування власного режиму. Цей сценарій важливий, оскільки користувач не завжди працює за готовими шаблонами і може самостійно визначати зручну тривалість фокусу, перерви та кількість циклів. Для зміни параметрів у застосунку використовується нижнє модальне вікно, яке не переводить користувача на окремий екран і дозволяє швидко змінити налаштування без втрати контексту головного екрана.

Екран налаштування власного режиму наведено на рис. 4.5.

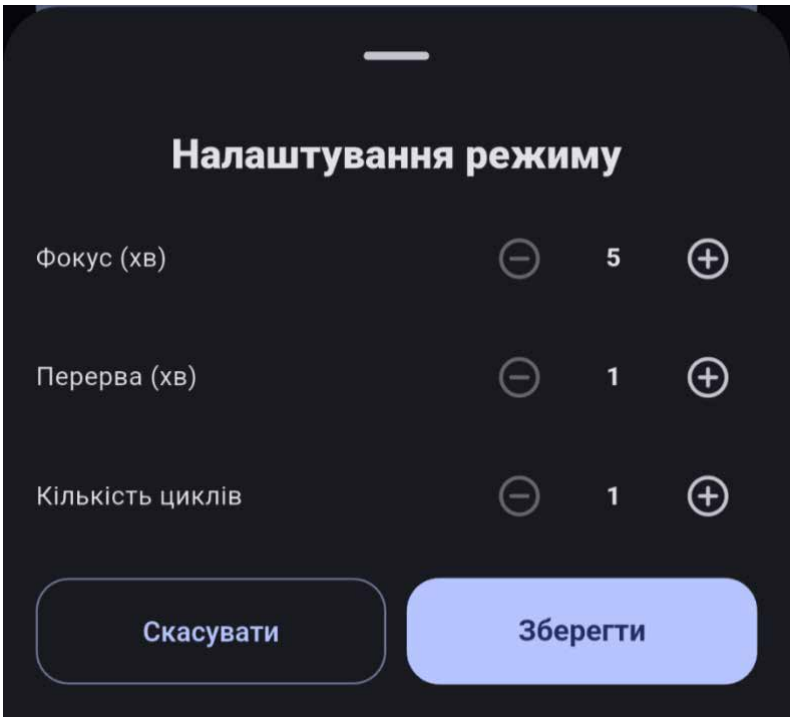

Рис. 4.5 Екран налаштування власного режиму, що ілюструє зміну параметрів фокус-сесії

Під час тестування було перевірено зміну тривалості фокусу, тривалості перерви та кількості циклів. Після натискання кнопки збереження оновлені параметри застосовуються до власного режиму та відображаються на головному екрані. Оскільки режим прив’язаний до облікового запису користувача, збережені параметри можуть використовуватися повторно під час наступних запусків застосунку та після входу на іншому пристрої.

Після налаштування власного режиму було перевірено запуск фокус-сесії. Для тестування використовувався скорочений власний режим, щоб швидко пройти повний цикл роботи таймера та перевірити всі основні стани сесії. Після натискання кнопки запуску застосунок відкриває екран таймера, встановлює фазу фокусу та починає відлік часу.

Екран фокус-сесії під час фази фокусу наведено на рис. Г.2.

На екрані таймера відображається поточна фаза, номер циклу, назва режиму, залишок часу та основні кнопки керування. Під час тестування було перевірено, що таймер запускається з параметрами власного режиму, коректно

відображає залишок часу та дозволяє поставити сесію на паузу, пропустити фазу або зупинити проходження.

Окремо було перевірено дострокове завершення фокус-сесії. Якщо користувач натискає кнопку зупинки, застосунок не завершує сесію одразу, а спочатку показує попередження. У цьому вікні відображається вже отриманий XP і можливий результат у разі продовження сесії. Такий підхід зменшує ризик випадкового завершення таймера та втрати поточного прогресу.

Попередження про дострокове завершення сесії наведено на рис. 4.6.

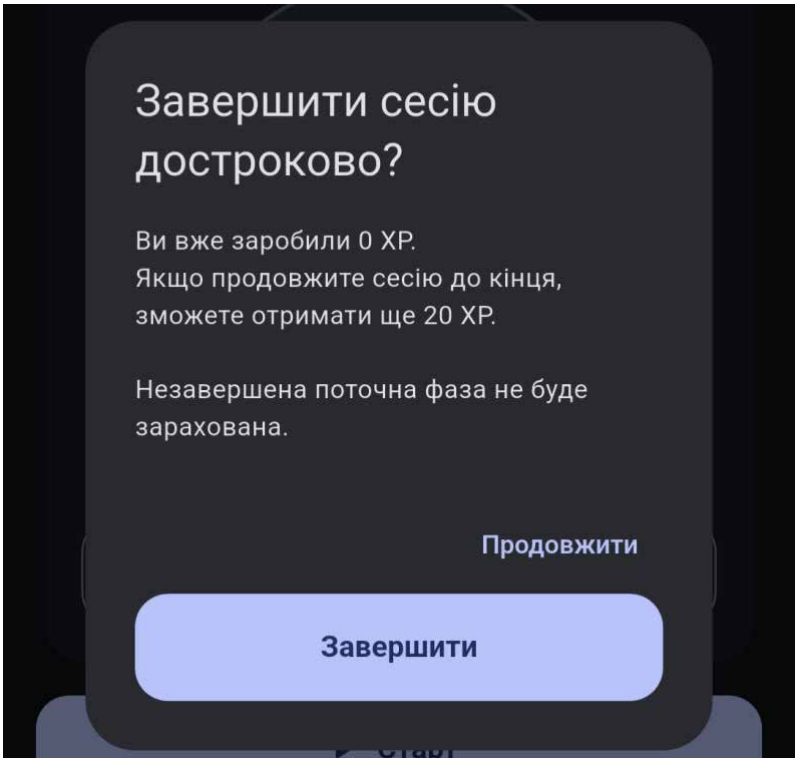

Рис. 4.6 Попередження про дострокове завершення фокус-сесії

Після завершення фази фокусу система переходить до фази перерви. На цьому етапі змінюється назва активної фази, текстова підказка та залишається інформація про поточний цикл. Це підтверджує, що застосунок не просто відраховує час, а змінює стан сесії відповідно до логіки циклу.

Екран фокус-сесії під час фази перерви наведено на рис. Г.3.

Після завершення останньої фази застосунок формує підсумок сесії. У вікні результату користувач бачить, що сесію завершено, скільки XP було отримано за останній етап і загальну кількість XP за сесію. Під час цієї перевірки

також було зафіксовано системне сповіщення про завершення сесії, що підтверджує роботу механізму локальних повідомлень.

Вікно завершення фокус-сесії наведено на рис. 4.7.

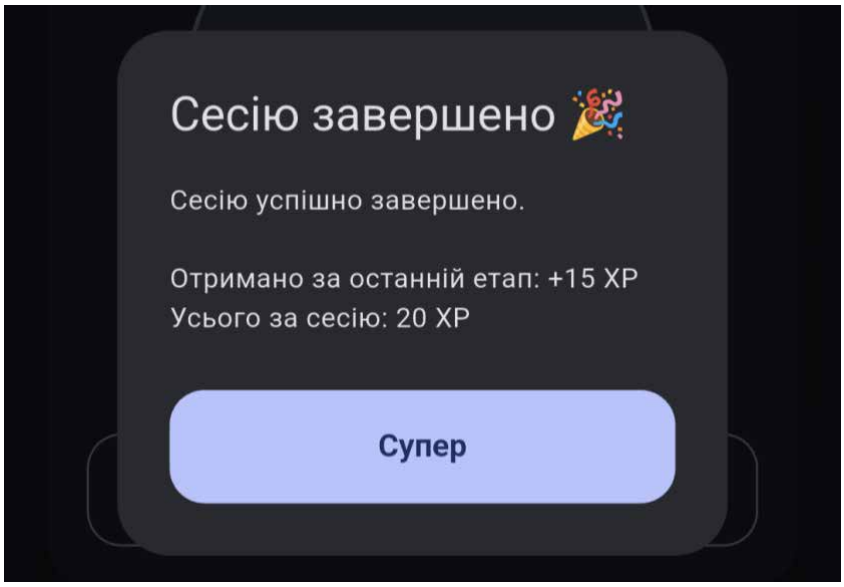

Рис. 4.7 Вікно завершення фокус-сесії, що ілюструє нарахування XP

Після завершення фокус-сесії система не обмежується показом підсумкового повідомлення. Результат сесії передається до механізму збереження, де фіксуються параметри режиму, кількість завершених фокус-блоків, завершені цикли, отриманий XP та час проходження сесії. Ці дані надалі використовуються для оновлення статистики, аналітики та загального прогресу користувача.

Після збереження результату система додатково перевіряє умови отримання досягнень. Якщо користувач виконав необхідні умови, застосунок показує окреме повідомлення зі списком нових досягнень. У тестовому сценарії після завершення сесії було відкрито досягнення, пов’язані з першою завершеною сесією та першим завершеним циклом.

Вікно отримання досягнень наведено на рис. 4.8.

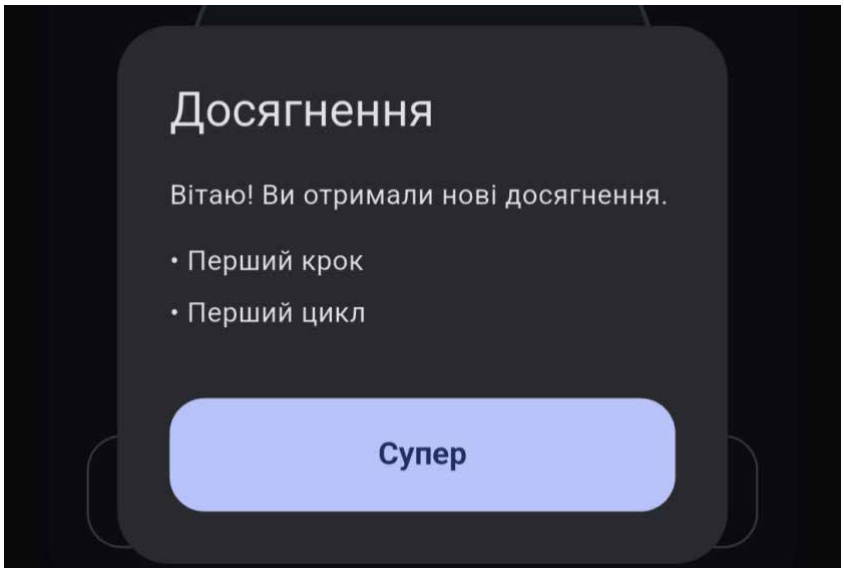

Рис. 4.8 Вікно отримання досягнень після завершення фокус-сесії

Таким чином було перевірено основний сценарій проходження фокус-сесії: запуск таймера, роботу фази фокусу, захист від випадкового дострокового завершення, перехід до перерви, завершення сесії, нарахування XP, системне сповіщення та відображення отриманих досягнень.

Після завершення фокус-сесії було перевірено оновлення статистики користувача. Цей етап потрібний для того, щоб переконатися, що результат сесії не тільки показується у вікні завершення, а й зберігається в загальному прогресі облікового запису. У статистиці відображаються кількість завершених сесій, хвилини фокусу, кількість днів поспіль, поточний рівень і накопичений XP.

Екран статистики користувача після завершення фокус-сесії наведено у додатку Г на рис. Г.4.

На цьому екрані видно, що після проходження тестової сесії в системі зафіксовано одну сесію, п’ять хвилин фокусу та 20 XP. Також відображається прогрес першого рівня. Це підтверджує зв’язок між завершенням фокус-сесії, нарахуванням XP і оновленням загального прогресу користувача.

Окремо було перевірено блок аналітики активності та попередній перегляд досягнень. На екрані статистики користувач може бачити активність за останні сім днів, а також перейти до детальнішої аналітики за вибраний період. Нижче

відображаються отримані й заблоковані досягнення, що дозволяє швидко оцінити поточний стан прогресу.

Екран статистики з активністю за 7 днів і попереднім переглядом досягнень наведено у додатку Г на рис. Г.5.

Під час тестування було перевірено, що дані завершеної сесії відображаються не тільки в загальних показниках, а й у блоці активності за період. Також було перевірено, що отримані досягнення позначаються окремо, а заблоковані залишаються недоступними до виконання відповідних умов.

Додатково було перевірено окремий екран аналітики фокус-сесій. Він дозволяє переглядати результати за різні проміжки часу: сім днів, тридцять днів, шість місяців, один рік або весь час. Такий екран потрібний для того, щоб користувач міг оцінити не тільки результат однієї сесії, а й свою активність за певний період.

Екран аналітики фокус-сесій наведено у додатку Г на рис. Г.6.

На екрані аналітики відображаються основні показники за вибраний період: кількість сесій, хвилини фокусу, XP і завершені цикли. Це підтверджує, що збережені результати сесій використовуються для формування зведених даних, а не лише для миттєвого повідомлення після завершення таймера.

Також було перевірено екран досягнень. Після завершення тестової сесії користувач отримав перші досягнення, які стали доступними для перегляду в загальному списку. На цьому екрані одночасно відображаються відкриті та ще заблоковані досягнення, тому користувач може бачити вже отриманий прогрес і подальші цілі.

Екран досягнень користувача наведено у додатку Г на рис. Г.7.

Для заблокованих досягнень застосунок показує умови відкриття. Це дає користувачеві зрозумілу мотивацію для подальшого проходження фокус-сесій. Під час тестування було відкрито один із заблокованих бейджів, щоб перевірити, чи відображається його назва, опис, вимога для отримання та поточний стан.

Екран перегляду умови заблокованого досягнення наведено на рис. 4.9.

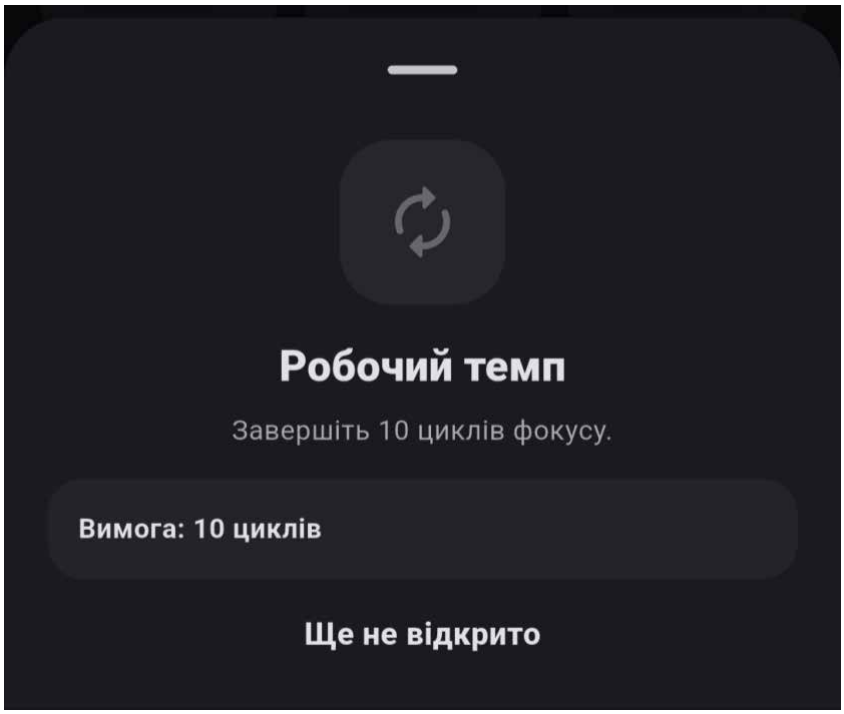

Рис. 4.9 Екран перегляду умови заблокованого досягнення

На рис. 4.9 видно, що досягнення «Робочий темп» ще не відкрито, а для його отримання потрібно завершити 10 циклів фокусу. Такий механізм робить систему досягнень зрозумілою для користувача, оскільки застосунок не просто показує заблокований бейдж, а пояснює конкретну умову його відкриття.

Окремо було перевірено екран налаштувань користувача. Він дозволяє змінювати поведінку застосунку відповідно до власних потреб. У верхній частині екрана відображається інформація про обліковий запис, зокрема електронна пошта, з якої виконано вхід. Також користувач може змінити тему застосунку, обравши світлий, темний або системний режим оформлення, та перемкнути мову інтерфейсу між українською й англійською.

Екран налаштувань користувача наведено в додатку Г на рис. Г.8.

На цьому ж екрані розміщено параметри проходження фокус-сесії. Користувач може ввімкнути автоматичний перехід між етапами після завершення поточної фази або вимкнути показ підтвердження перед достроковим завершенням сесії. Це дає змогу зробити проходження сесій більш автоматизованим або, навпаки, залишити додаткове підтвердження для захисту від випадкового завершення.

Налаштування сповіщень і виходу з облікового запису наведено на рис. 4.10.

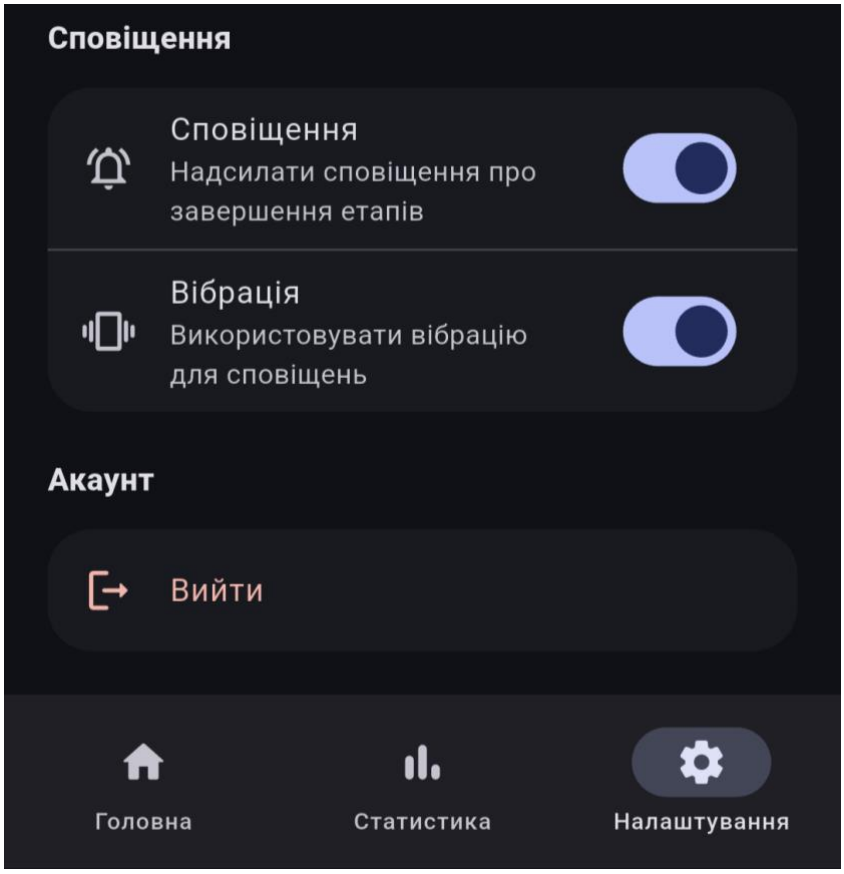

Рис. 4.10 Екран налаштувань користувача, що ілюструє керування сповіщеннями та вихід з акаунта

У блоці сповіщень користувач може ввімкнути або вимкнути повідомлення про завершення етапів, а також окремо налаштувати використання вібрації. Нижче розміщено кнопку виходу з облікового запису. Під час тестування було перевірено, що перемикачі налаштувань змінюють стан, а екран коректно відображає параметри застосунку в темній темі.

У результаті ручного функціонального тестування було перевірено основні сценарії роботи PieceTime: реєстрацію користувача, підтвердження електронної пошти, вибір і налаштування режиму, запуск та проходження фокус-сесії, завершення таймера, нарахування XP, отримання досягнень, оновлення статистики, перегляд аналітики та зміну налаштувань застосунку. Проведена перевірка показала, що застосунок коректно виконує основні функції та зберігає результат роботи користувача в межах його облікового запису.

4.2 Вимоги до апаратного та програмного забезпечення

Апаратні та програмні вимоги визначають умови, за яких застосунок може бути встановлений і коректно використовуватися користувачем. Для мобільного застосунку PieceTime основним середовищем роботи є «Android»-пристрій. Частина функцій виконується безпосередньо на пристрої користувача, а частина пов’язана з хмарними сервісами «Firebase», які використовуються для автентифікації та збереження даних.

Оскільки застосунок працює не лише як локальний таймер, а й як система збереження прогресу користувача, для нього важлива взаємодія між клієнтською частиною, хмарною базою даних і системними службами «Android». Таймер, інтерфейс і локальні сповіщення виконуються на мобільному пристрої, тоді як автентифікація, статистика, історія сесій і досягнення пов’язані з віддаленими сервісами. Тому перед визначенням вимог потрібно показати, де саме розміщені основні компоненти системи. Діаграму розміщення компонентів застосунку PieceTime наведено на рис. 4.11.

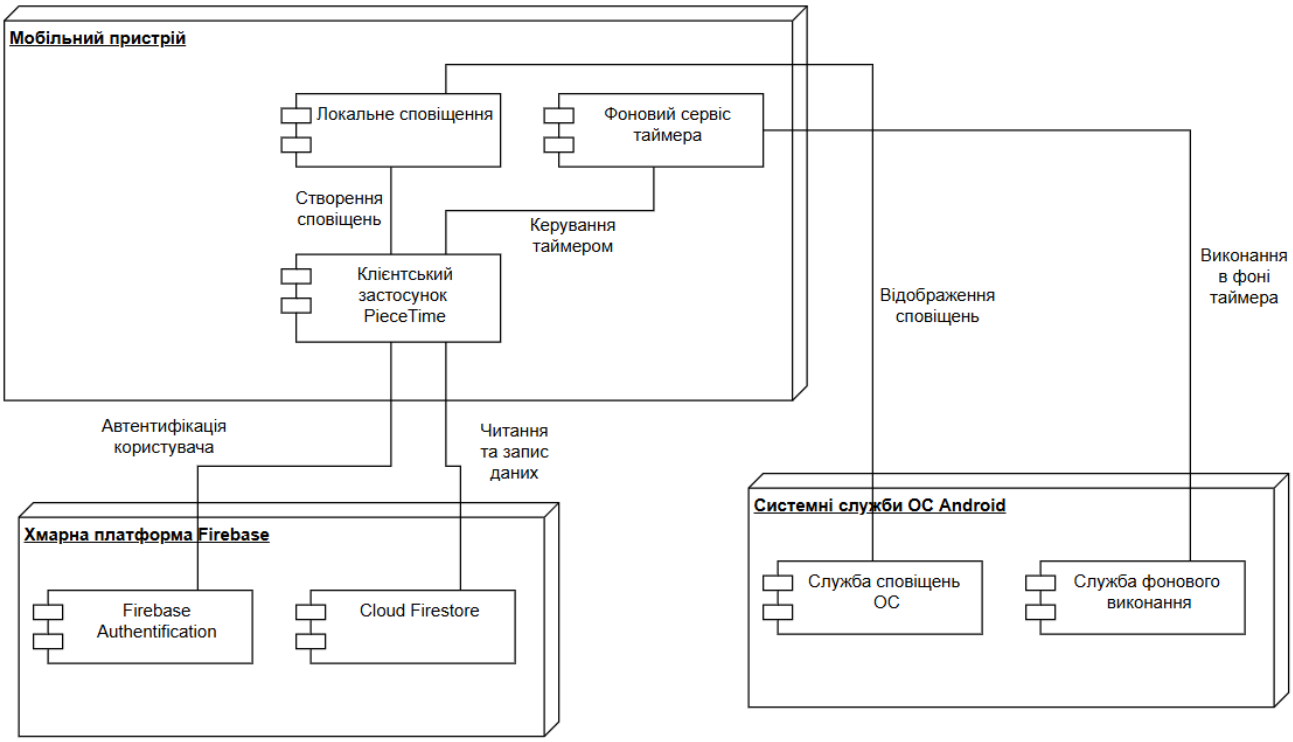

Рис. 4.11 Діаграма розміщення компонентів застосунку PieceTime

На діаграмі показано три основні частини системи: мобільний пристрій користувача, хмарну платформу «Firebase» та системні служби ОС «Android». На мобільному пристрої встановлюється клієнтський застосунок PieceTime, який відповідає за інтерфейс, запуск фокус-сесій, керування таймером і взаємодію з локальними сповіщеннями. «Firebase Authentication» використовується для реєстрації, входу та підтвердження електронної пошти, а «Cloud Firestore» – для збереження статистики, історії сесій, XP, досягнень і налаштувань користувача.

Системні служби «Android» забезпечують показ локальних сповіщень і виконання таймера у фоні [19]. Це потрібно для того, щоб користувач міг отримувати повідомлення про завершення фази або всієї сесії навіть тоді, коли застосунок згорнуто.

На основі діаграми розміщення можна визначити вимоги до пристрою користувача та програмного середовища. Оскільки PieceTime є мобільним застосунком, основні вимоги стосуються версії «Android», наявності вільного місця, доступу до мережі та системних дозволів для роботи сповіщень. Вимоги до апаратного та програмного забезпечення наведено в табл. 4.

Таблиця 4

Вимоги до апаратного та програмного забезпечення для роботи застосунку PieceTime

Компонент	**Мінімальні вимоги**	**Рекомендовані вимоги**
Пристрій	Смартфон або планшет з Android	Смартфон або планшет з Android
Операційна система	Android 8.0 або новіша версія	Android 10 або новіша версія
Оперативна пам’ять	2 ГБ	4 ГБ або більше
Вільне місце	180 МБ	250 МБ або більше

Таблиця 4 (продовження)

Компонент	Мінімальні вимоги	Рекомендовані вимоги
Інтернет-з'єднання	Потрібне для реєстрації, входу та синхронізації	Стабільне Wi-Fi або мобільне з'єднання
Системні дозволи	Сповіщення, будильники та нагадування	Увімкнені сповіщення, будильники та нагадування

Для встановлення PieceTime користувачеві потрібен «Android»-пристрій, який підтримує встановлення «APK»-файлів. Поточний розмір встановленого застосунку становить приблизно 177 МБ, тому мінімально потрібно передбачити не менше 180 МБ вільного місця. Рекомендовано мати більший запас пам'яті, оскільки під час роботи застосунок може зберігати локальні дані, налаштування та тимчасові записи, які потрібні для стабільної роботи.

До програмних вимог належить операційна система «Android» 8.0 або новіша версія. Для використання облікового запису, підтвердження електронної пошти, синхронізації статистики, XP, досягнень і параметрів власного режиму потрібне підключення до Інтернету. При цьому сам таймер запускається на мобільному пристрої, тому короткочасна відсутність мережі не повинна заважати проходженню фокус-сесії.

Окремо потрібно надати системні дозволи «Android». Дозвіл на сповіщення потрібний для показу повідомлень про завершення фази або всієї сесії [18]. Дозвіл на будильники та нагадування використовується для запланованих повідомлень, які мають спрацьовувати в потрібний момент. Без цих дозволів застосунок може запускатися, але повідомлення про завершення фази або сесії можуть не відображатися коректно.

4.3 Склад інсталяційного пакету

Для використання мобільного застосунку PieceTime його потрібно встановити на «Android»-пристрій. Основним інсталяційним файлом є «APK»-збірка, яка містить скомпільований застосунок, ресурси інтерфейсу,

налаштування «Android» та необхідні компоненти для запуску програми на реальному пристрої.

Інсталяційний пакет PieceTime формується під «Android» і призначений для встановлення безпосередньо на смартфон або планшет користувача. Після встановлення застосунок може бути запущений з головного меню пристрою. Користувач проходить реєстрацію або вхід, підтверджує електронну пошту, надає потрібні системні дозволи та отримує доступ до основної функціональності: вибору режиму, проходження фокус-сесії, перегляду статистики та досягнень.

Оскільки PieceTime є мобільним застосунком, користувачеві не потрібно встановлювати середовище розробки, окремі бібліотеки або додаткові програмні модулі. Усі необхідні ресурси, службові компоненти та налаштування входять до складу «APK»-файлу. Це спрощує процес встановлення, оскільки для запуску застосунку достатньо перенести інсталяційний файл на «Android»-пристрій і виконати стандартну процедуру встановлення.

Після першого запуску застосунок може запросити системні дозволи, необхідні для коректної роботи окремих функцій. Насамперед це стосується дозволів на показ сповіщень, а також будильників і нагадувань. Вони потрібні для того, щоб користувач міг отримувати повідомлення про завершення фази фокус-сесії або всієї сесії навіть тоді, коли застосунок згорнуто.

Також потрібно враховувати, що частина функцій PieceTime пов’язана з хмарними сервісами «Firebase». Сам «APK»-файл містить клієнтську частину застосунку та конфігурацію підключення, але реєстрація користувача, підтвердження електронної пошти, синхронізація статистики, XP, досягнень і параметрів режимів виконуються через мережеве з’єднання. Тому для повноцінного використання застосунку після встановлення потрібен доступ до Інтернету. Склад інсталяційного пакету застосунку PieceTime наведено в табл. 5.

Таблиця 5

Склад інсталяційного пакету застосунку PieceTime

Компонент	**Призначення**
APK-файл застосунку PieceTime	Основний файл для встановлення застосунку на Android-пристрій
Скомпільований клієнтський застосунок	Забезпечує запуск інтерфейсу, таймера, статистики, досягнень і налаштувань
Ресурси інтерфейсу	Містять іконки, графічні елементи, тему, splash screen та інші візуальні ресурси
Файли локалізації	Забезпечують відображення текстів інтерфейсу українською та англійською мовами
Налаштування Android Manifest	Визначають назву застосунку, іконку, головну активність, дозволи та системні компоненти
Компоненти локальних сповіщень	Використовуються для повідомлень про завершення фази або всієї фокус-сесії
Компоненти фонового таймера	Забезпечують роботу таймера під час згортання застосунку
Конфігурація підключення до Firebase	Містить параметри, необхідні для підключення застосунку до Firebase Authentication та Cloud Firestore

Під час встановлення користувачеві не потрібно окремо додавати бібліотеки або налаштовувати середовище розробки. Усі необхідні компоненти для запуску застосунку входять до складу «APK»-файлу. Окремі системні параметри застосунку, зокрема дозволи, активності та службові компоненти, визначаються у файлі «Android Manifest» [24]. Також користувач має надати системні дозволи «Android» на надсилання сповіщень, а також на використання будильників і нагадувань. Ці дозволи потрібні для коректної роботи локальних повідомлень під час фокус-сесії.

Після встановлення PieceTime взаємодіє з хмарною платформою «Firebase», яка не входить до APK як окремий локальний компонент, але використовується застосунком під час роботи. «Firebase Authentication» забезпечує реєстрацію, вхід і підтвердження електронної пошти, а Cloud Firestore використовується для збереження статистики, XP, досягнень, історії сесій і параметрів власного режиму.

Отже, інсталяційний пакет PieceTime складається з «APK»-файлу та вбудованих ресурсів, необхідних для роботи мобільного застосунку. Для повноцінного використання системи потрібен «Android»-пристрій, встановлений застосунок, надані системні дозволи та доступ до Інтернету для авторизації й синхронізації даних користувача.

Висновки до розділу 4

У четвертому розділі було перевірено працездатність мобільного застосунку PieceTime після реалізації основних функцій. Для цього було проведено ручне функціональне тестування на «Android»-пристрої після встановлення «APK»-збірки. Такий підхід дав змогу перевірити роботу застосунку в умовах, близьких до реального використання, а не лише в середовищі розробки.

Під час тестування було перевірено основні сценарії роботи системи: реєстрацію користувача, підтвердження електронної пошти, вибір готового режиму, налаштування власного режиму, запуск фокус-сесії, перехід між фазами фокусу та перерви, завершення сесії, нарахування XP, відкриття досягнень, оновлення статистики та формування аналітичних даних. Окремо було перевірено роботу локальних сповіщень і системних дозволів «Android», які потрібні для повідомлень про завершення фази або всієї сесії.

Також у розділі було визначено апаратні та програмні вимоги для встановлення й використання PieceTime. Основним середовищем роботи застосунку є «Android»-пристрій з підтримкою встановлення «APK»-файлів, доступом до Інтернету для авторизації та синхронізації, а також наданими дозволами на сповіщення, будильники й нагадування. Діаграма розміщення

показала взаємодію мобільного застосунку з «Firebase Authentication», «Cloud Firestore» та системними службами «Android».

Крім цього, було описано склад інсталяційного пакету. Основним компонентом є «APK»-файл, який містить скомпільований клієнтський застосунок, ресурси інтерфейсу, файли локалізації, налаштування «Android Manifest», компоненти локальних сповіщень, фонового таймера та конфігурацію підключення до «Firebase». Після встановлення користувач не потребує додаткового середовища розробки або окремого налаштування бібліотек.

Отже, результати тестування підтвердили, що застосунок PieceTime коректно виконує основні функції, зберігає прогрес користувача в межах його облікового запису та може бути встановлений і протестований на реальному «Android»-пристрої. Це підтверджує готовність програмного продукту до демонстрації та подальшого практичного використання.

ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи було створено мобільний застосунок PieceTime для підтримки фокус-сесій із використанням елементів гейміфікації. Система дозволяє користувачеві організовувати роботу у вигляді послідовних циклів фокусу та відпочинку, обирати готові режими, налаштовувати власний режим, відстежувати результати активності та зберігати особистий прогрес.

Розроблений застосунок поєднує функції таймера, статистики, аналітики та мотиваційної системи. Користувач отримує XP за хвилини фокусу, завершені цикли та повністю завершені сесії, а також може відкривати досягнення за виконання певних умов. Завдяки цьому PieceTime не обмежується простим відліком часу, а допомагає підтримувати регулярність і бачити результат виконаної роботи.

Особливістю застосунку є підтримка збереження та синхронізації даних користувача. Результати фокус-сесій, статистика, XP, досягнення та налаштування режимів зберігаються в межах облікового запису. За відсутності мережевого з’єднання результати сесій можуть бути тимчасово збережені локально, що зменшує ризик втрати прогресу та робить застосунок придатним для використання в нестабільних мережевих умовах.

Результати тестування підтвердили коректну роботу основних сценаріїв PieceTime: реєстрації користувача, підтвердження електронної пошти, вибору та налаштування режиму, запуску й завершення фокус-сесії, переходу між фазами, нарахування XP, відкриття досягнень, оновлення статистики, роботи аналітики та локальних сповіщень. Застосунок було перевірено на реальному «Android»-пристрої після встановлення «APK»-збірки.

Основною перевагою PieceTime є поєднання простого інструмента для керування часом із системою мотивації та відстеження прогресу. На відміну від звичайного таймера, застосунок показує користувачеві накопичений результат, допомагає аналізувати активність і створює додатковий стимул повертатися до

фокус-сесій. Це робить програмний продукт корисним для навчання, роботи та повсякденних задач, які потребують концентрації.

Отже, поставлену мету бакалаврської кваліфікаційної роботи досягнуто. Було створено мобільний застосунок, який допомагає користувачеві організовувати фокусовану діяльність, чергувати роботу з відпочинком, накопичувати статистику активності та підтримувати мотивацію за допомогою ігрових механік.

У подальшому PieceTime може бути розширений окремим екраном користувацького акаунта, де користувач зможе керувати даними профілю, змінювати пароль та переглядати основну інформацію про обліковий запис [25]. Також доцільним напрямом розвитку є додавання можливості створювати й зберігати кілька власних режимів фокус-сесій для різних типів діяльності. Оскільки застосунок розроблено з використанням «Flutter», у майбутньому можна розглянути його адаптацію для «iOS»-пристроїв, оскільки цей фреймворк підтримує створення й випуск застосунків для цієї платформи [26].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Pomodoro Technique – Why it works & how to do it. Todoist. URL: https://www.todoist.com/ru/productivity-methods/pomodoro-technique (дата звернення: 18.04.2026).
2. Pomodoro Technique – Time Management Method. URL: https://www.pomodorotechnique.com/ (дата звернення: 18.04.2026).
3. Жученя К. Засіб від прокрастинації: техніка Помодоро. Happy Monday. URL: https://happymonday.ua/tehnika-pomodoro (дата звернення: 18.04.2026).
4. Forest: Stay focused, be present. URL: https://www.forestapp.cc/ (дата звернення: 12.05.2026).
5. Focus To-Do: Pomodoro Technique & Tasks. URL: https://www.focustodo.cn/ (дата звернення: 12.05.2026).
6. Pomofocus: Pomodoro Timer. URL: https://pomofocus.io/ (дата звернення: 12.05.2026).
7. Deterding S., Dixon D., Khaled R., Nacke L. From Game Design Elements to Gamefulness: Defining “Gamification”. URL: https://dl.acm.org/doi/10.1145/2181037.2181040 (дата звернення: 13.05.2026).
8. Hamari J., Koivisto J., Sarsa H. Does Gamification Work? A Literature Review of Empirical Studies on Gamification. URL: https://www.computer.org/csdl/proceedings-article/hicss/2014/2504d025/12OmNzE54xe (дата звернення: 13.05.2026).
9. Flutter documentation. URL: https://docs.flutter.dev/ (дата звернення: 13.05.2026).
10. Dart documentation. URL: https://dart.dev/docs (дата звернення: 13.05.2026).
11. Firebase Authentication. Firebase Documentation. URL: https://firebase.google.com/docs/auth (дата звернення: 14.05.2026).

12. Cloud Firestore. Firebase Documentation. URL: https://firebase.google.com/docs/firestore (дата звернення: 14.05.2026).
13. Firestore in Native mode documentation. Google Cloud. URL: https://docs.cloud.google.com/firestore/native/docs (дата звернення: 14.05.2026).
14. Riverpod. Dart package. URL: https://pub.dev/packages/riverpod (дата звернення: 14.05.2026).
15. Shared preferences. Flutter package. URL: https://pub.dev/packages/shared_preferences (дата звернення: 14.05.2026).
16. flutter_local_notifications. Flutter package. URL: https://pub.dev/packages/flutter_local_notifications (дата звернення: 15.05.2026).
17. flutter_foreground_task. Flutter package. URL: https://pub.dev/packages/flutter_foreground_task (дата звернення: 15.05.2026).
18. About notifications. Android Developers. URL: https://developer.android.com/develop/ui/views/notifications (дата звернення: 15.05.2026).
19. Foreground services overview. Android Developers. URL: https://developer.android.com/develop/background-work/services/fgs (дата звернення: 16.05.2026).
20. Android Studio documentation. Android Developers. URL: https://developer.android.com/studio/intro (дата звернення: 20.05.2026).
21. About repositories. GitHub Docs. URL: https://docs.github.com/en/repositories/creating-and-managing-repositories/about-repositories (дата звернення: 20.05.2026).
22. Internationalizing Flutter apps. Flutter documentation. URL: https://docs.flutter.dev/ui/internationalization (дата звернення: 21.05.2026).
23. Material Design 3. URL: https://m3.material.io/ (дата звернення: 21.05.2026).

24. App manifest overview. Android Developers. URL: https://developer.android.com/guide/topics/manifest/manifest-intro (дата звернення: 21.05.2026).
25. Build and release an iOS app. Flutter documentation. URL: https://docs.flutter.dev/deployment/ios (дата звернення: 21.05.2026).

ДОДАТОК А

ДІАГРАМА АКТИВНОСТІ ПРОХОДЖЕННЯ ФОКУС-СЕСІЇ

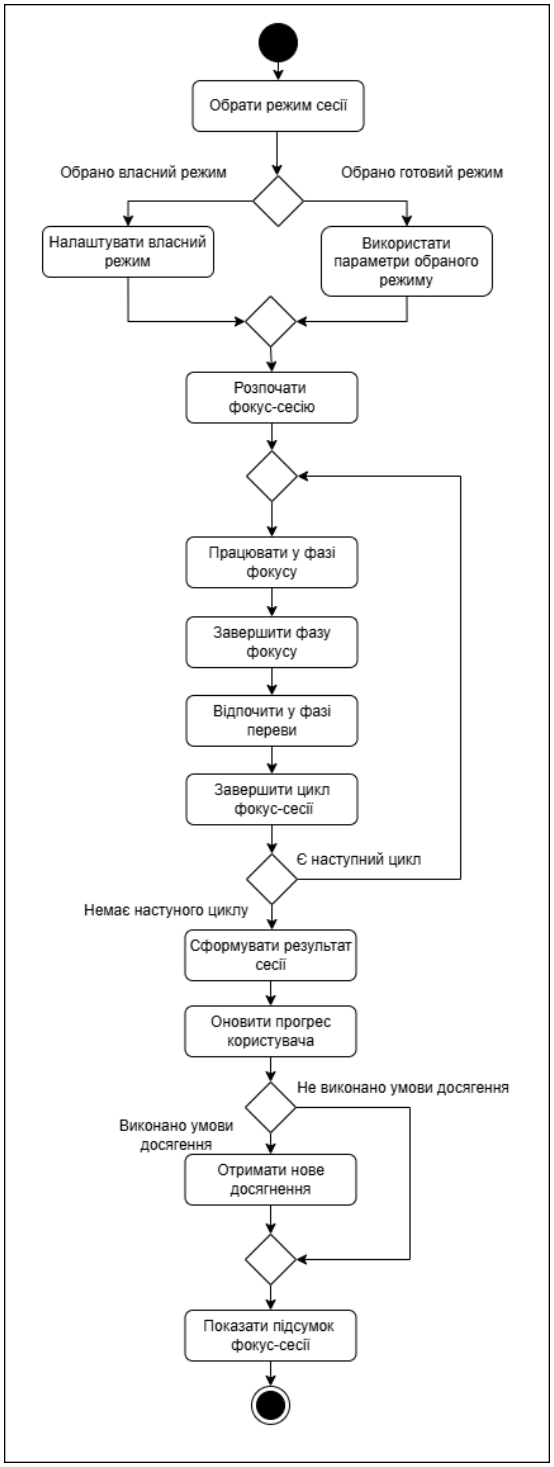

Рис. А.1 Діаграма активності проходження фокус-сесії

ДОДАТОК Б

БЛОК-СХЕМИ АЛГОРИТМІВ РОБОТИ ОСНОВНИХ МОДУЛІВ ЗАСТОСУНКУ

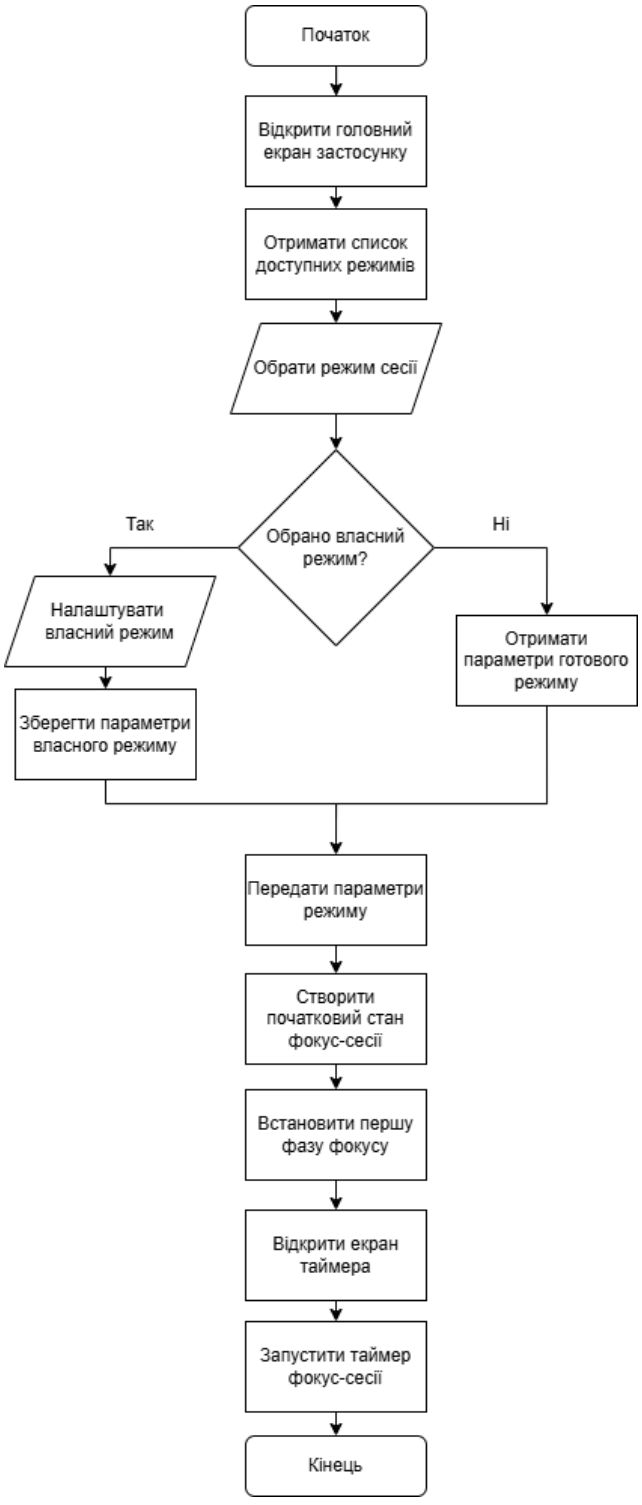

Рис. Б.1 Блок-схема алгоритму запуску фокус-сесії

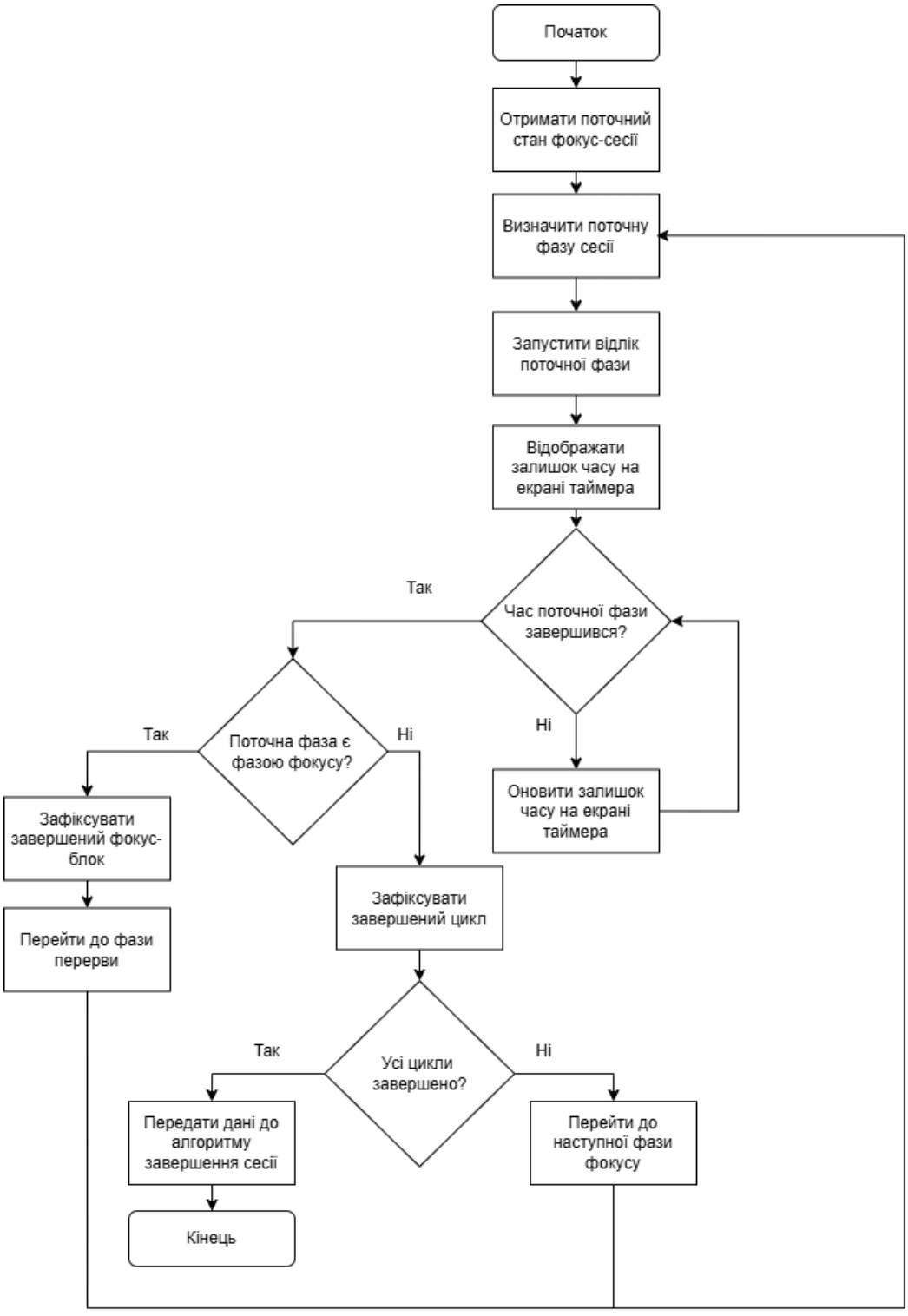

Рис. Б.2 Блок-схема алгоритму проходження фокус-сесії

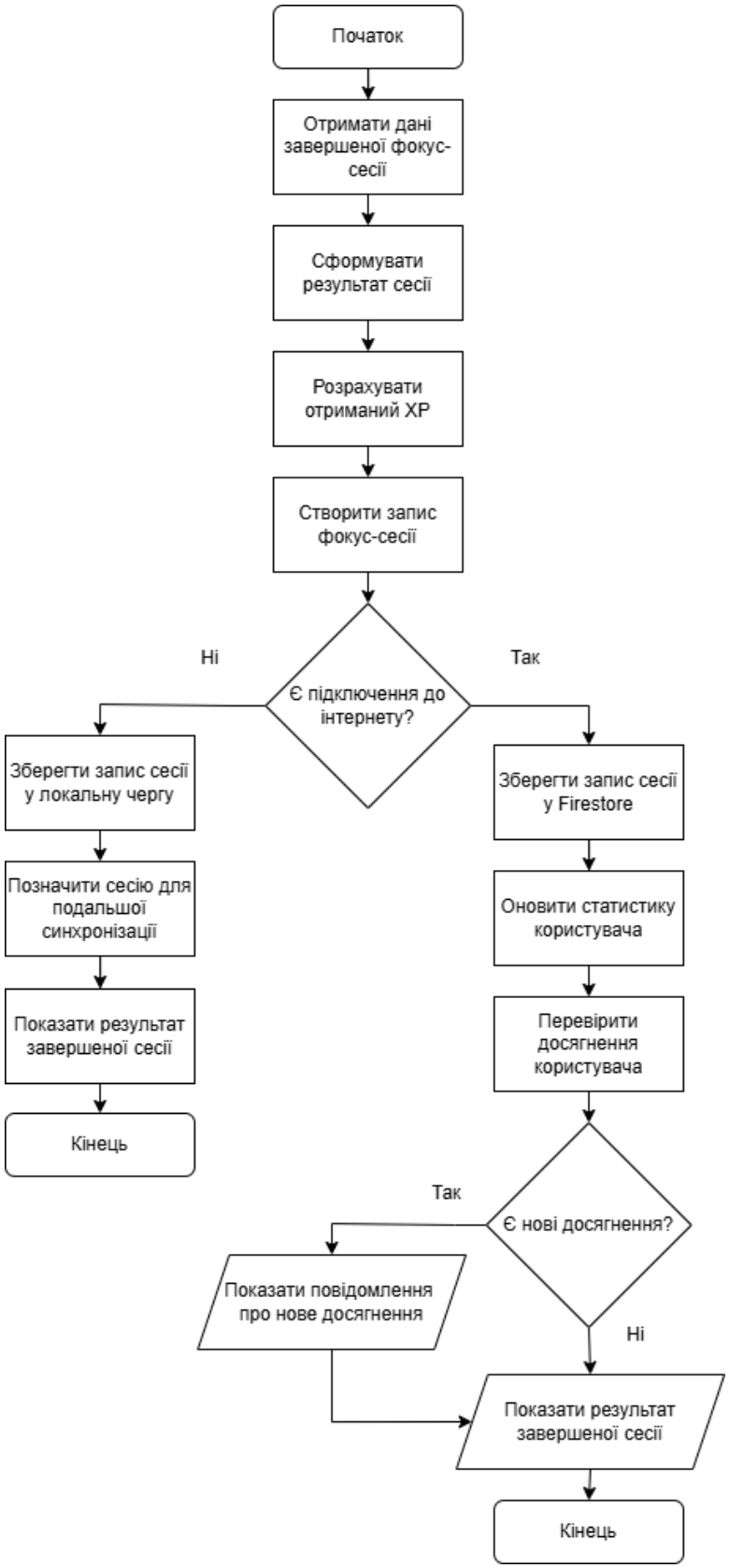

Рис. Б.3 Блок-схема алгоритму завершення та збереження фокус-сесії

ДОДАТОК В

ФРАГМЕНТИ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ОСНОВНИХ МОДУЛІВ ЗАСТОСУНКУ PIECETIME

Лістинг В.1 Фрагмент реалізації запуску фокус-сесії та фонового таймера

```
void start() {
  if (state.isRunning) return;

  final settings =
      ref.read(appSettingsProvider).value ?? const AppSettings.initial();

  final sessionStartedAt = state.startedAt ?? DateTime.now();

  _endsAt = DateTime.now().add(
    Duration(seconds: state.remainingSeconds),
  );

  state = state.copyWith(
    isRunning: true,
    startedAt: sessionStartedAt,
  );

  _ticker?.cancel();

  _ticker = Timer.periodic(
    const Duration(seconds: 1),
    (_) => _tick(),
  );

  final notificationText = _notificationText(
    phase: state.phase,
    presetName: state.preset.name,
  );

  if (settings.notificationsEnabled) {
    ref.read(notificationServiceProvider).schedulePhaseDone(
      after: Duration(seconds: state.remainingSeconds),
      title: notificationText.title,
      body: notificationText.body,
```

```
      withVibration: settings.vibrationEnabled,
    );
  }

  ref.read(focusFgServiceProvider).start(
    seconds: state.remainingSeconds,
    title: notificationText.title,
    body: notificationText.body,
  );
}
```

Лістинг В.2 Фрагмент реалізації керування станом фокус-сесії

```
void _completePhase({required bool rewardUser}) {
  final preset = state.preset;

  if (state.phase == FocusPhase.focus) {
    state = state.copyWith(
      phase: FocusPhase.rest,
      isRunning: false,
      remainingSeconds: _restSeconds(preset),
      completedFocusBlocks: rewardUser
          ? state.completedFocusBlocks + 1
          : state.completedFocusBlocks,
    );
    return;
  }

  final updatedCompletedCycles = rewardUser
      ? state.completedCycles + 1
      : state.completedCycles;

  final isLastCycle = state.cycleIndex >= preset.cycles;

  if (isLastCycle) {
    state = state.copyWith(
      isRunning: false,
      remainingSeconds: 0,
      completedCycles: updatedCompletedCycles,
      isSessionCompleted: rewardUser,
    );
```

```
    return;
  }

  state = state.copyWith(
    phase: FocusPhase.focus,
    cycleIndex: state.cycleIndex + 1,
    isRunning: false,
    remainingSeconds: _focusSeconds(preset),
    completedCycles: updatedCompletedCycles,
  );
}
```

Лістинг В.3 Фрагмент реалізації збереження результату сесії та перевірки досягнень

```
Future<List<BadgeDefinition>> _saveSessionIfNeeded(TimerState state) async {
  if (_sessionSaved) return [];
  if (!_hasProgress(state)) return [];

  _sessionSaved = true;

  final user = FirebaseAuth.instance.currentUser;
  if (user == null) return [];

  final startedAt = state.startedAt;
  if (startedAt == null) return [];

  final earnedXp = _earnedXp(state);

  final session = FocusSessionRecord(
    presetId: state.preset.id.name,
    presetName: state.preset.name,
    focusMin: state.preset.focusMin,
    breakMin: state.preset.breakMin,
    plannedCycles: state.preset.cycles,
    completedFocusBlocks: state.completedFocusBlocks,
    completedCycles: state.completedCycles,
    earnedXp: earnedXp,
    isSessionCompleted: state.isSessionCompleted,
    startedAt: startedAt,
    finishedAt: DateTime.now(),
  );
```

```
  await ref
      .read(sessionHistoryRepositoryProvider)
      .saveSession(userId: user.uid, session: session);

  final badgeResult = await ref
      .read(badgeAwarderProvider)
      .evaluateAndUnlockBadges(userId: user.uid);

  if (!badgeResult.hasNewBadges) return [];

  return badgeResult.newlyUnlocked;
}
```

Лістинг В.4 Фрагмент реалізації визначення рівня користувача за кількістю XP

```
int _xpRequiredForLevel(int level) {
  return 100 + (level - 1) * (level - 1) * 50;
}

int _calculateLevel(int totalXp) {
  var level = 1;
  var remainingXp = totalXp;

  while (remainingXp >= _xpRequiredForLevel(level)) {
    remainingXp -= _xpRequiredForLevel(level);
    level++;
  }

  return level;
}

int _xpForLevelStart(int level) {
  var total = 0;

  for (var currentLevel = 1; currentLevel < level; currentLevel++) {
    total += _xpRequiredForLevel(currentLevel);
  }

  return total;
}
```

```
int _xpForNextLevel(int level) {
  return _xpForLevelStart(level) + _xpRequiredForLevel(level);
}
```

ДОДАТОК Г

ЕКРАННІ ФОРМИ МОБІЛЬНОГО ЗАСТОСУНКУ PIECETIME

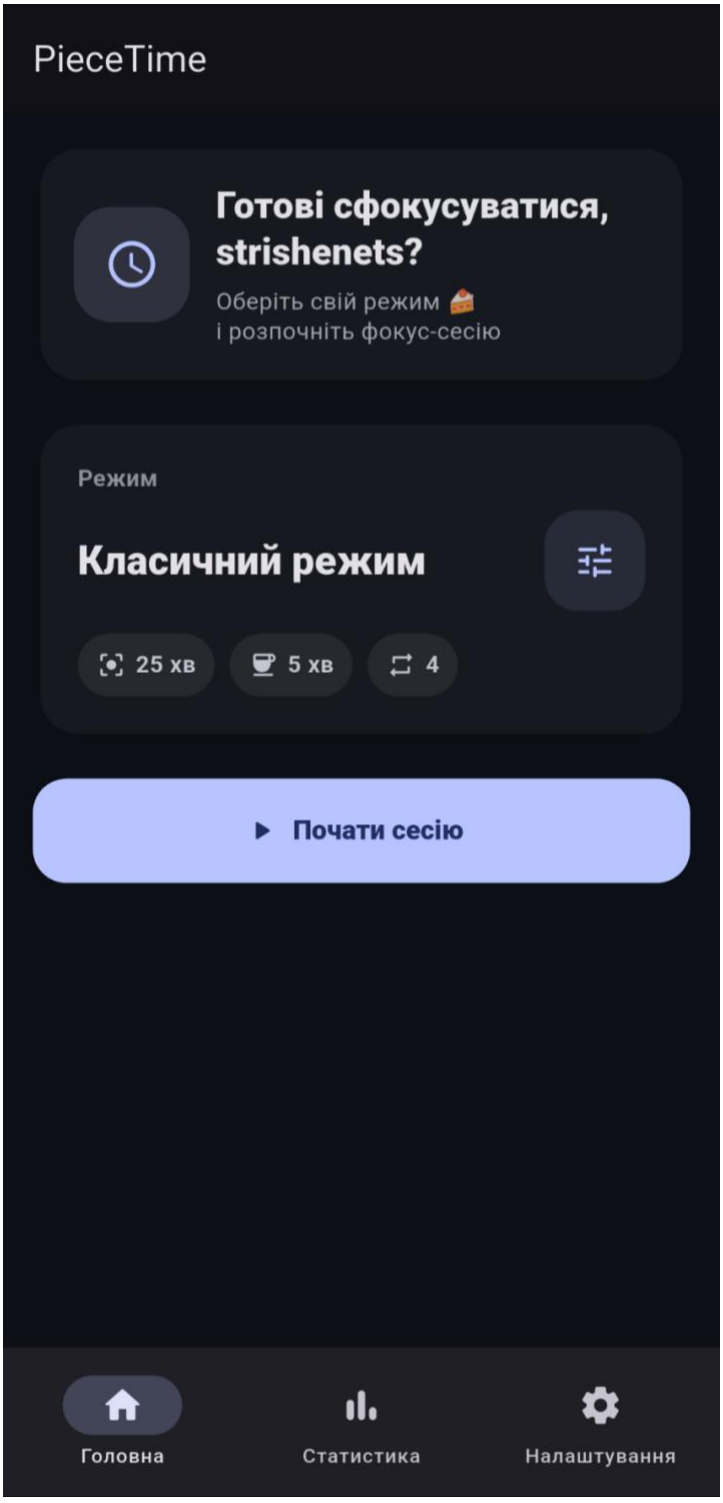

Рис. Г.1 Головний екран застосунку, що ілюструє режим фокус-сесії за замовчуванням та нижню навігацію

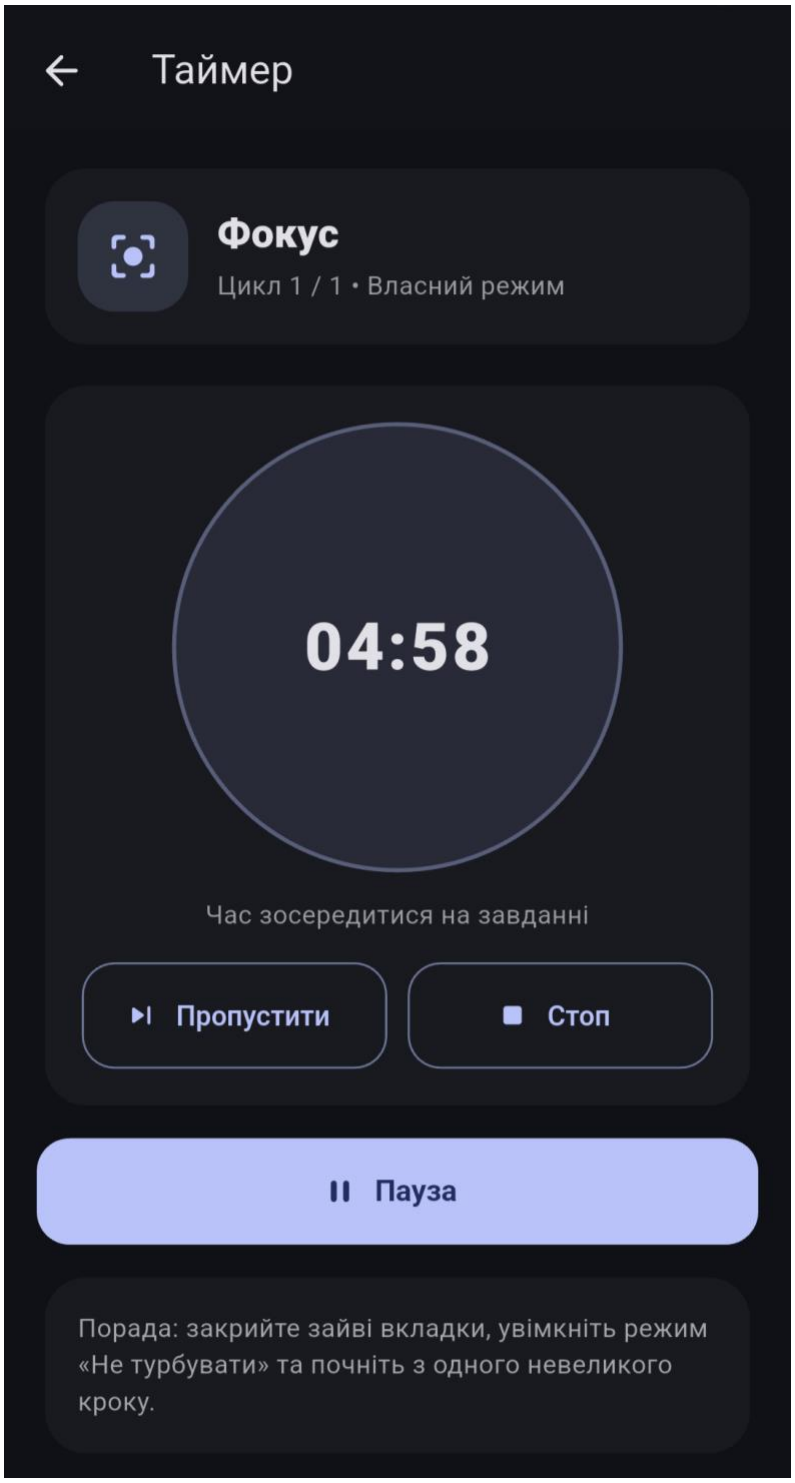

Рис. Г.2 Екран фокус-сесії під час фази фокусу, що ілюструє роботу таймера

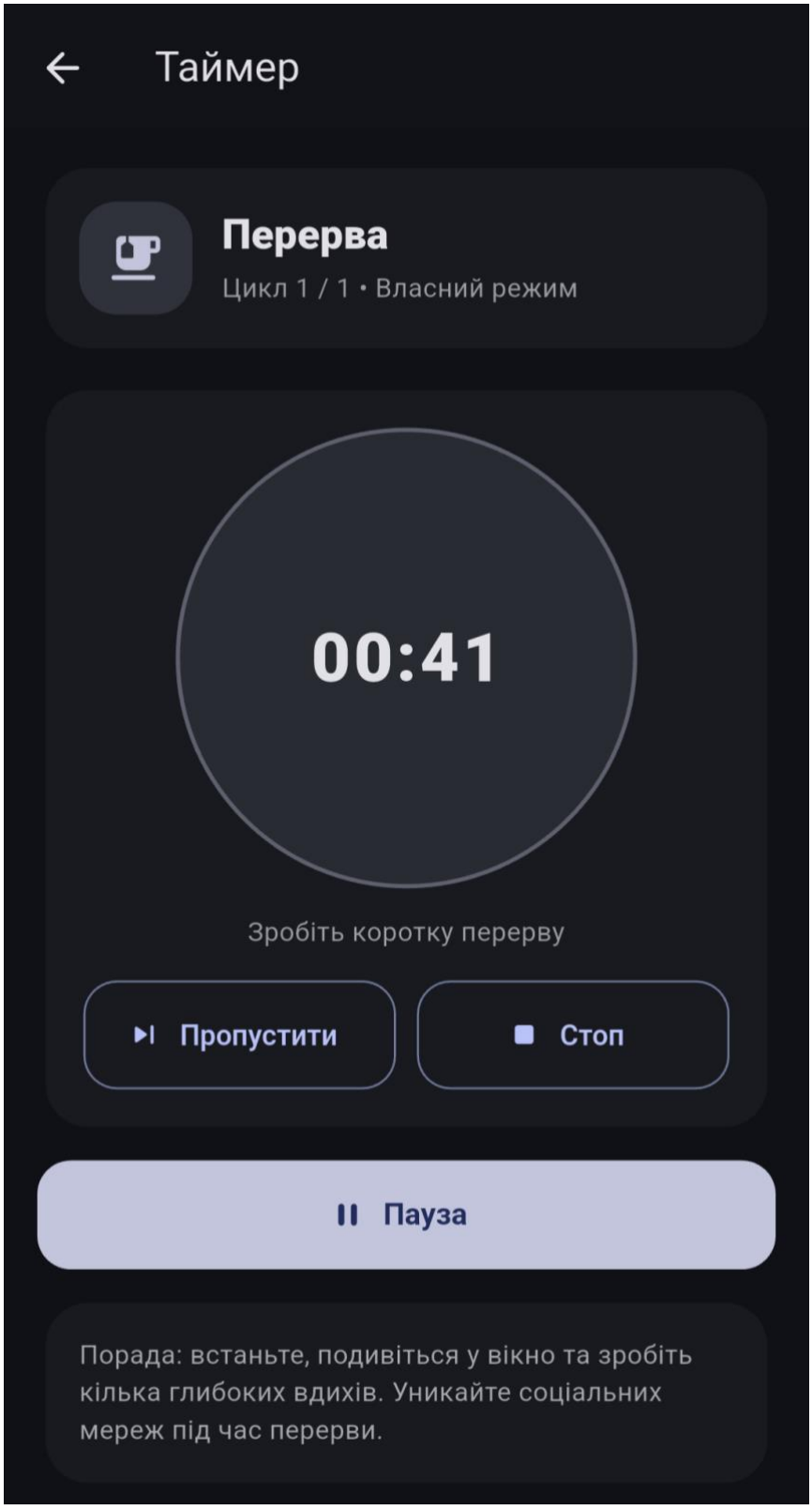

Рис. Г.3 Екран фокус-сесії під час фази перерви, що ілюструє перехід між фазами

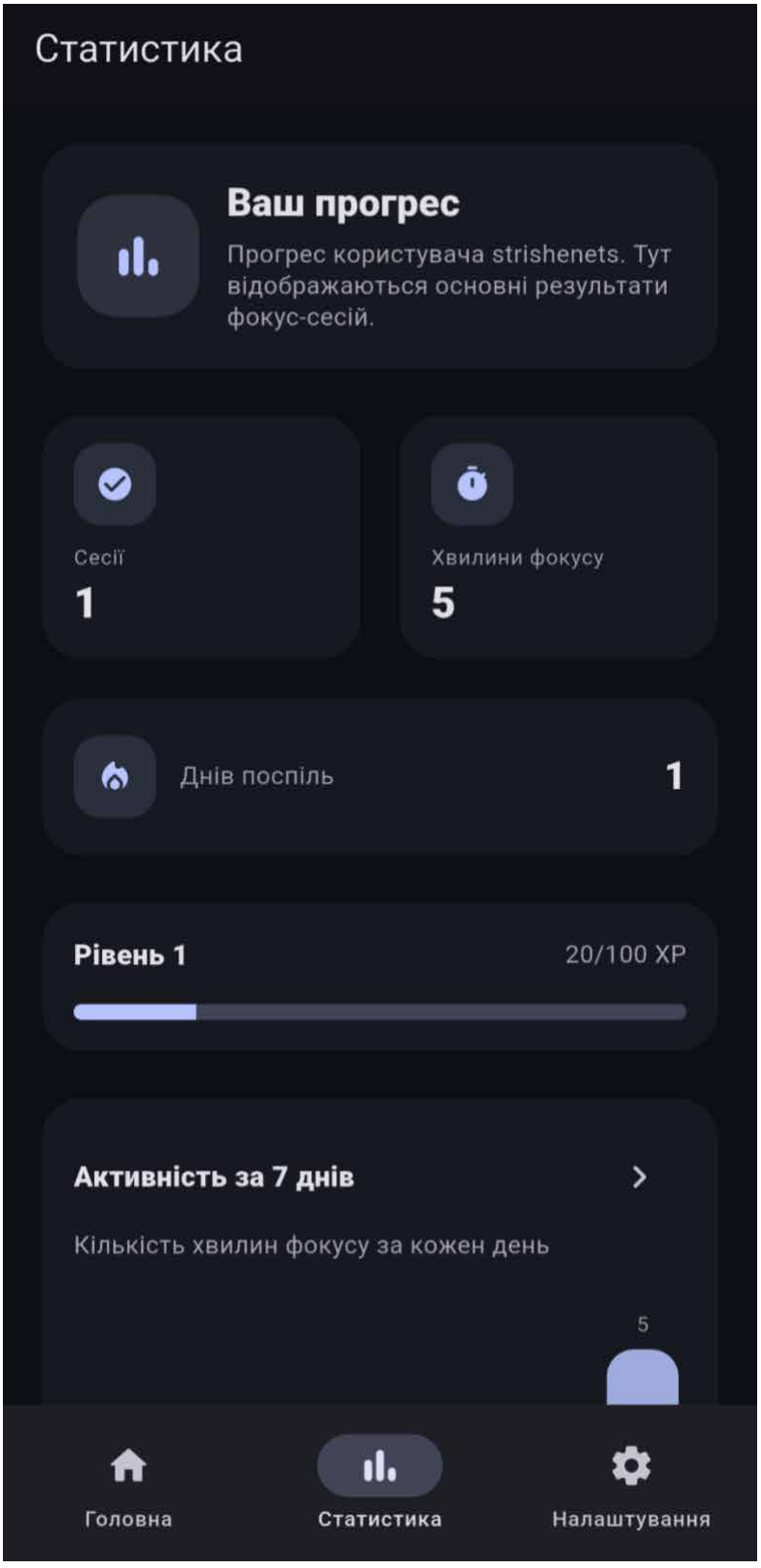

Рис. Г.4 Екран статистики користувача після завершення фокус-сесії

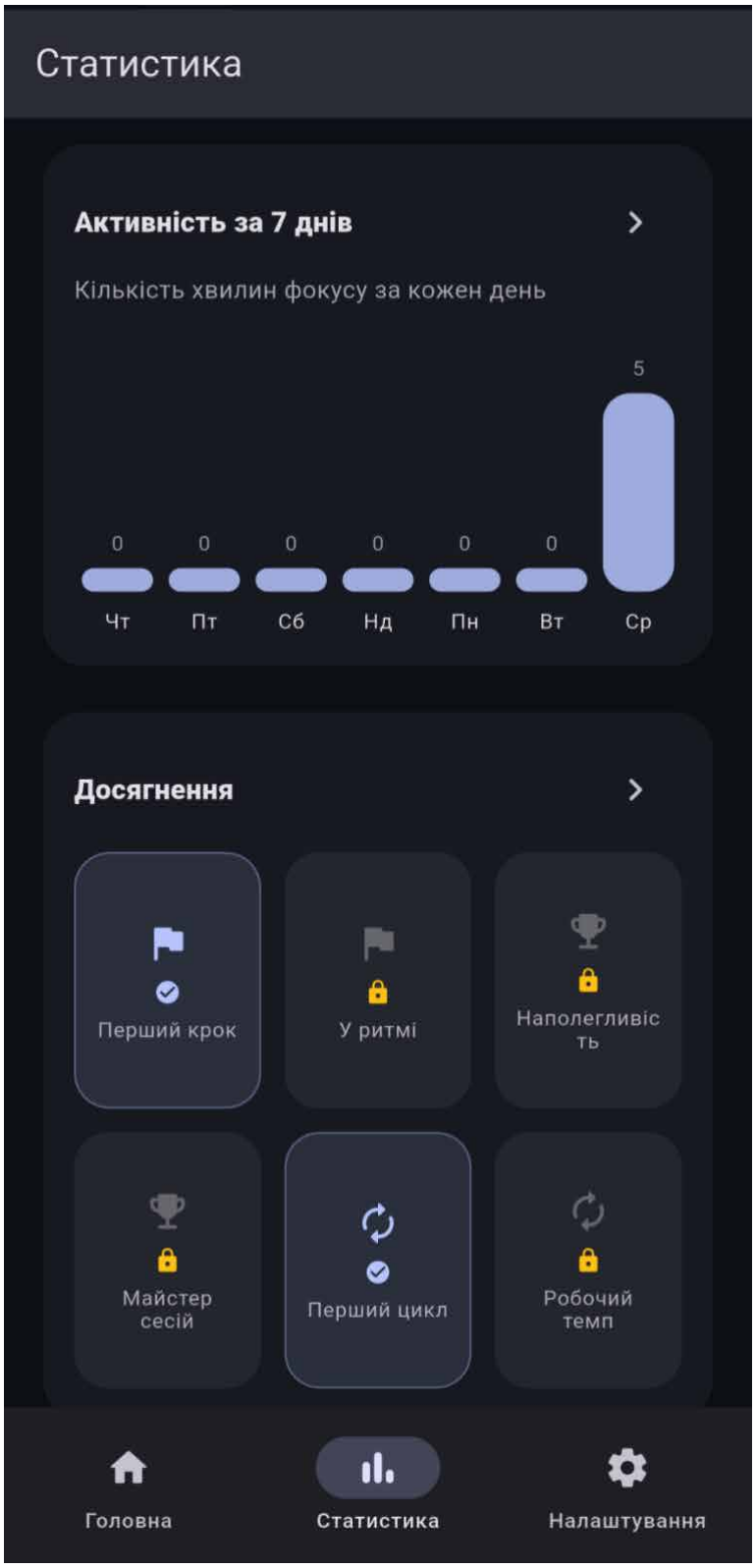

Рис. Г.5 Екран статистики з аналітикою активності та попереднім переглядом досягнень

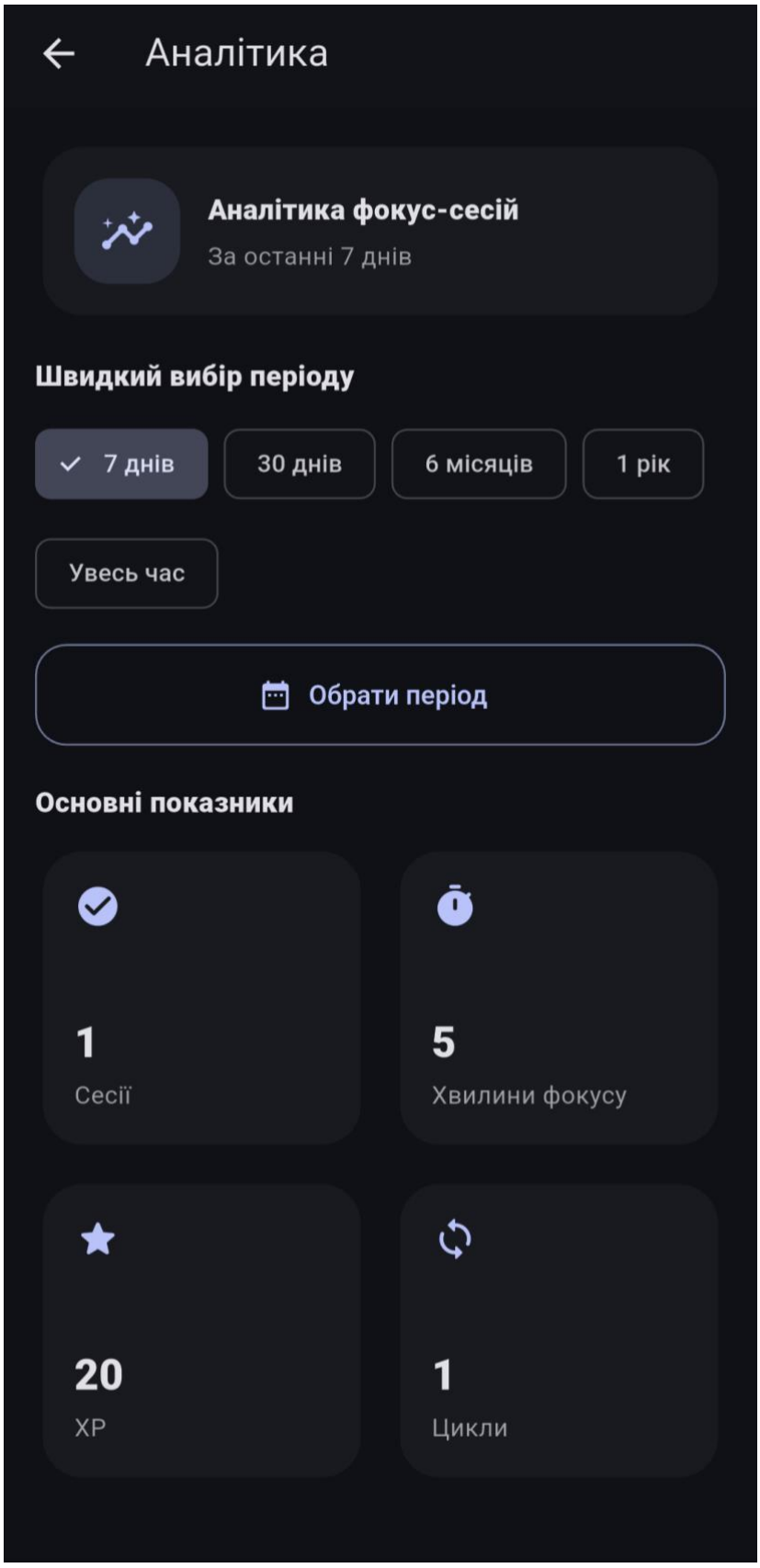

Рис. Г.6 Екран аналітики фокус-сесій

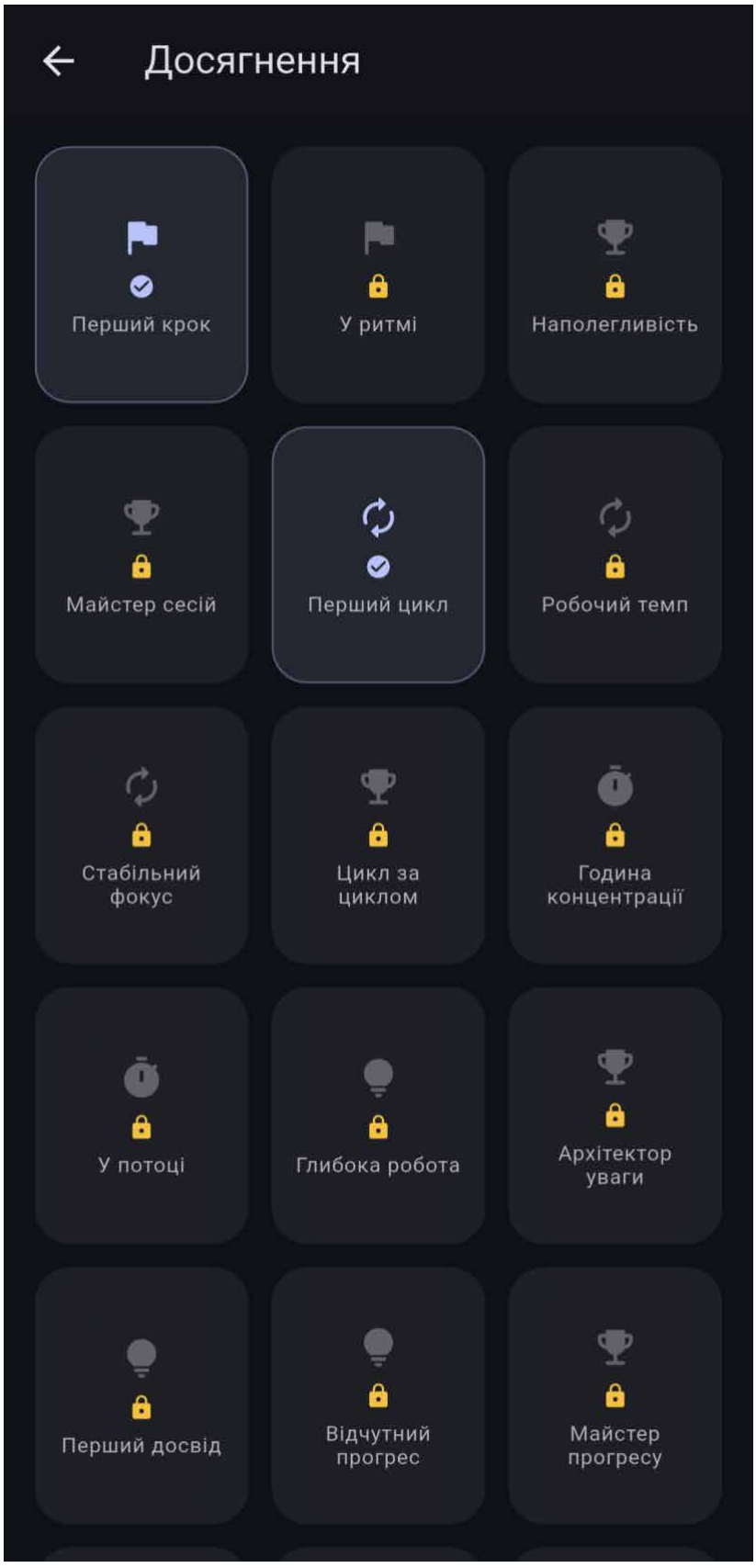

Рис. Г.7 Екран досягнень користувача після завершення фокус-сесії

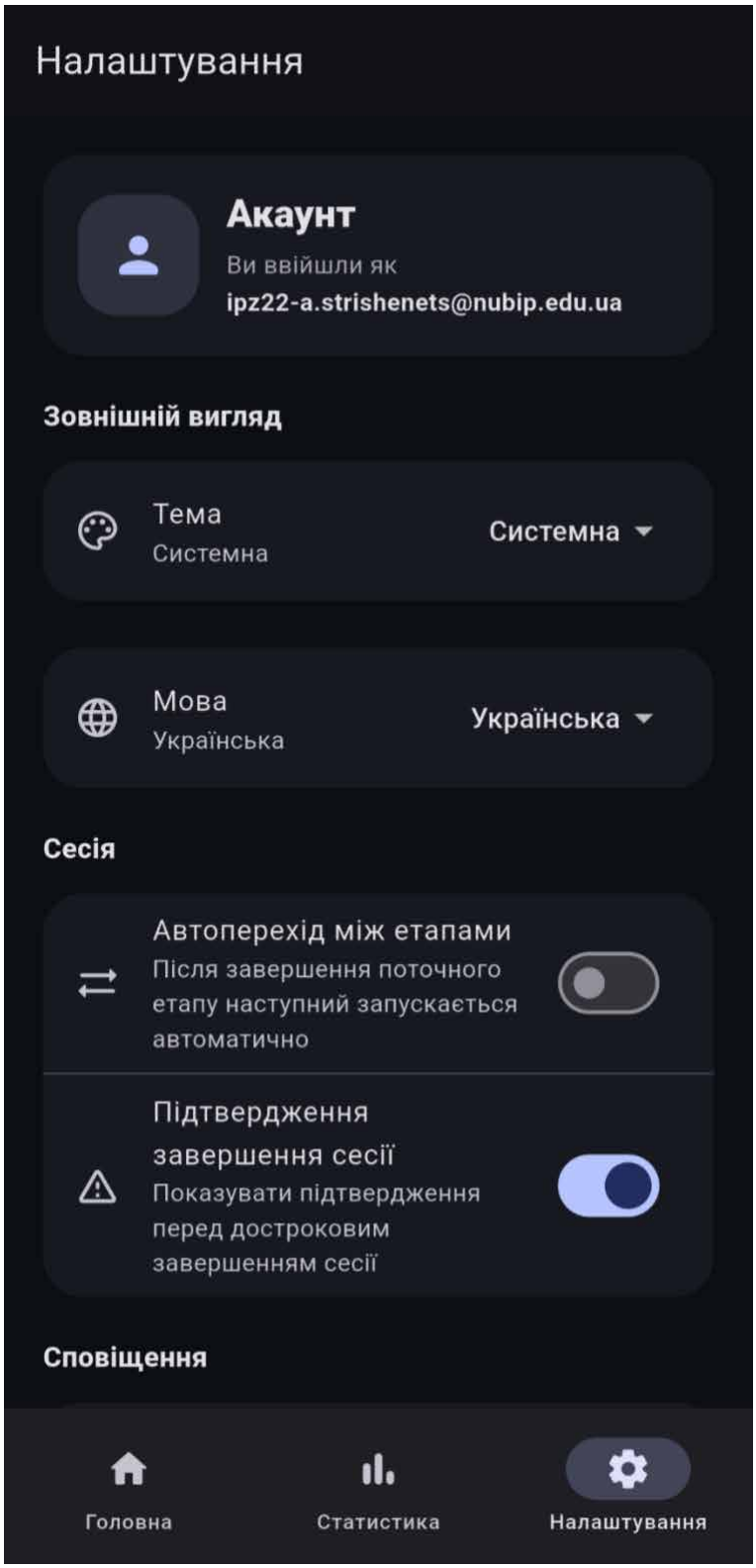

Рис. Г.8 Екран налаштувань користувача, що ілюструє параметри акаунта, теми, мови та проходження сесії

ДОДАТОК Д

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Стрішенець Ангеліна Олександрівна, здобувачка НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення гейміфікованого мобільного застосунку підтримки фокус-сесії» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

- [] інструменти генеративного ШІ не використовувалися.
- [ ] інструмент ШІ ChatGPT був використаний для:
  - [ ] перекладу іншомовних джерел;
  - [✓] стилістичного редагування та перевірки граматики;
  - [ ] допомоги у написанні/відлагодженні програмного коду;
  - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___»_____ 2026 р. _____ **Ангеліна СТРІШЕНЕЦЬ**

(підпис)