ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ»»

Факультет комп'ютерно-інформаційних технологій

Кафедра комп’ютерних інформаційних систем і технологій

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Тема «Розробка методики інтеграції гейміфікованих елементів у користувацькі інтерфейси мобільних книгарень»

Виконала студентка групи ІК-9-22-Б1КНТ Карпачова Є.Ю.

Керівник: к.т.н., професор Авер’янова Ю.А.

Рецензент: к.т.н., доцент

Допущено до захисту

Завідувач кафедри _____ д.е.н., професор Кавун С.В.

(підпис)

Київ 2026

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи: 127 с., 23 рис., 20 табл., 1 додаток, 53 джерел.

ГЕЙМІФІКАЦІЯ, МОБІЛЬНА КНИГАРНЯ, FLUTTER, DART, BLOC-АРХІТЕКТУРА, FIREBASE, SQLITE, FIGMA, WCAG 2.1, UX-ДИЗАЙН.

Об'єктом дослідження є процес інформаційної та когнітивної взаємодії користувача з графічним інтерфейсом мобільних застосунків у сегменті цифрової дистрибуції книжкового контенту як динамічна система людино-машинної взаємодії.

Метою роботи є наукове обґрунтування, проєктування та практична розробка цілісної інформаційної моделі та функціонального прототипу гейміфікованої системи в межах інтерфейсу мобільної книгарні для підвищення показників користувацької залученості у різноманітних контекстах застосування – від дозвіллєвого читання до освітніх та корпоративних сценаріїв.

Методи розробки базуються на мові програмування Dart у середовищі Flutter з використанням архітектурного патерну BLoC для управління станом, бази даних SQLite для локального зберігання ігрового прогресу, хмарного сховища Firebase Firestore для синхронізації даних та інструменту Figma для прототипування інтерфейсу.

Результатом роботи є розроблена методика інтеграції гейміфікованих елементів у користувацькі інтерфейси мобільних книгарень, що включає архітектуру трьох доменних класів Dart (UserProgress, VirtualEconomy, AchievementsAndQuests), алгоритмічну базу системи нарахування XP та управління квестами, сім спеціалізованих Flutter-віджетів та верифікований High-fidelity прототип із показником SUS 82.5 балів. Розроблена методика є відтворюваною та може застосовуватись для гейміфікації будь-якого мобільного застосунку з освітнім або читацьким контентом.

GAMIFICATION, MOBILE BOOKSTORE, FLUTTER, DART, BLOC ARCHITECTURE, FIREBASE, SQLITE, FIGMA, WCAG 2.1, UX DESIGN.

The object of research is the process of informational and cognitive interaction between the user and the graphical interface of mobile applications in the segment of digital book content distribution as a dynamic human-machine interaction system.

The aim of the work is the scientific substantiation, design and practical development of a comprehensive information model and functional prototype of a gamified system within the interface of a mobile bookstore to improve user engagement metrics across various application contexts – from leisure reading to educational and corporate scenarios.

Development methods are based on the Dart programming language in the Flutter framework using the BLoC architectural pattern for state management, SQLite database for local storage of game progress, Firebase Firestore cloud storage for data synchronization, and Figma for interface prototyping.

The result of the work is a developed methodology for integrating gamified elements into mobile bookstore user interfaces, including the architecture of three domain Dart classes (UserProgress, VirtualEconomy, AchievementsAndQuests), an algorithmic base for XP accrual and quest management, seven specialized Flutter widgets, and a verified High-fidelity prototype with a SUS score of 82.5 points. The developed methodology is reproducible and can be applied to gamify any mobile application with educational or reading content.

ЗМІСТ

ВСТУП……..6

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ ГЕЙМІФІКАЦІЇ В ІТ-СИСТЕМАХ

1.1 Концепція гейміфікації як інструменту управління поведінкою користувача……...……8

1.2 Огляд існуючих програмних рішень для мобільних книгарень……..10

1.3. Порівняльний аналіз архітектурних підходів до інтеграції ігрових елементів……13

1.4. Формулювання технічного завдання на розробку системи……..……..17

РОЗДІЛ 2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ АРХІТЕКТУРИ МОБІЛЬНОЇ КНИГАРНІ

2.1 Побудова моделі використання: ролі та сценарії взаємодії…….20

2.2 Проєктування структури даних: які атрибути потрібні для збереження прогресу користувача……25

2.3. Розробка діаграми станів для опису переходів між екранами та ігровими статусами……....41

2.4. Обґрунтування вибору платформи та інструментарію для реалізації інтерфейсу……..48

РОЗДІЛ 3 РОЗРОБКА АЛГОРИТМІЧНОЇ БАЗИ ГЕЙМІФІКОВАНИХ МОДУЛІВ

3.1 Алгоритми нарахування винагород та управління рівнями користувача...54

3.2. Проєктування системи тригерів та пуш-сповіщень на основі дій користувача.……....60

3.3 Технічні аспекти забезпечення доступності (Accessibility) згідно зі стандартами WCAG……...……64

3.4 Методика інтеграції ігрових елементів у стандартні UI-компоненти мобільного застосунку…….…69

РОЗДІЛ 4 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНТЕРФЕЙСНОГО ПРОТОТИПУ

4.1. Створення інтерактивного макета високої деталізації……....74

4.2. Опис логіки роботи гейміфікованих віджетів……..89

4.3 Тестування прототипу на відповідність технічним вимогам та зручність навігації……..…..103

4.4 Оцінка продуктивності запропонованих рішень……112

ВИСНОВКИ……...…..117

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ…….…120

ДОДАТОК А…….125

ВСТУП

Процес глобальної цифровізації суттєво трансформував архітектуру книжкового ринку, перемістивши основний фокус споживання контенту у площину мобільних технологій. Згідно з даними Global eBook Report, частка електронних книг щороку зростає, а мобільні пристрої стають основним каналом споживання цифрового контенту [52]. Проте сучасні мобільні книгарні стикаються з критичним викликом – низьким показником утримання користувачів: стандартні транзакційні інтерфейси, орієнтовані на модель «пошук – придбання», виявляються недостатньо ефективними для формування стійкої лояльності [33]. Гейміфікація в контексті мобільних застосунків не є суто візуальним доповненням, а являє собою складну програмно-алгоритмічну інтеграцію механік ігрового дизайну в неігрові контексти [29]. Вона базується на створенні петель зворотного зв'язку, що перетворюють пасивне використання інтерфейсу на активний ігровий досвід [37]. Пінк підкреслює, що внутрішні мотиватори – майстерність, автономія та мета – є значно ефективнішими для формування стійкої поведінки, ніж зовнішні винагороди [43], а Кепп підтверджує особливу ефективність гейміфікованих систем в освітньому контексті [29]: студенти з дедлайнами, учасники читацьких клубів та корпоративні команди – усі потребують інструменту з вимірюваним прогресом і часовими тригерами. Еял визначає тригерно-нагородну петлю як ключовий механізм формування цифрових звичок – саме цей механізм реалізує система щоденного стріку застосунку «SmartBook» [16].

Об'єктом дослідження є цілісний процес інформаційної та когнітивної взаємодії користувача з графічним інтерфейсом мобільних застосунків у сегменті цифрової дистрибуції книжкового контенту як динамічна система людино-машинної взаємодії, що охоплює всі етапи – від первинного входу до формування довготривалої лояльності [42, 10].

Предметом дослідження є сукупність методів, алгоритмів та архітектурних рішень, що забезпечують методичну інтеграцію гейміфікованих компонентів у структуру користувацьких інтерфейсів мобільних книгарень, включаючи логічні моделі нарахування винагород, механізми управління ігровими станами та алгоритми візуалізації прогресу [34, 15].

Метою роботи є наукове обґрунтування, проєктування та практична розробка цілісної інформаційної моделі та функціонального прототипу гейміфікованої системи в межах інтерфейсу мобільної книгарні для підвищення показників користувацької залученості у різноманітних контекстах застосування – від дозвіллєвого читання до освітніх та корпоративних сценаріїв [34, 29]. Результатом досягнення мети має стати інтерактивний прототип високої точності, що демонструє здатність запропонованої моделі ефективно корелювати з психологічними та технічними аспектами людино-машинної взаємодії [23, 27].

Для досягнення мети визначено такі завдання: – проаналізувати існуючі підходи до гейміфікації мобільних застосунків та класифікувати архітектурні підходи до інтеграції ігрових елементів; – сформулювати технічне завдання на розробку гейміфікованої системи з урахуванням вимог продуктивності, доступності та модульності; – спроєктувати структуру даних та UML-діаграми для формалізації поведінки гейміфікованих модулів; – розробити алгоритмічну базу системи нарахування винагород, тригерів та пуш-сповіщень; – реалізувати методику інтеграції гейміфікованих Flutter-компонентів відповідно до принципу декоратора; – розробити інтерактивний High-fidelity прототип у Figma та провести його двоетапне тестування; – оцінити продуктивність запропонованих рішень та сформулювати рекомендації щодо усунення виявлених юзабіліті-проблем.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ ГЕЙМІФІКАЦІЇ В ІТ-СИСТЕМАХ

1.1 Концепція гейміфікації як інструменту управління поведінкою користувача

Концепція гейміфікації розглядається не як сукупність ігрових атрибутів, а як стратегічний засіб керування поведінкою користувача-споживача шляхом проєктування специфічних інформаційних просторів [53]. В основі цієї концепції лежить ідея застосування ігрових механік та елементів дизайну ігор у неігрових контекстах для стимулювання внутрішньої мотивації та формування стійких алгоритмів взаємодії з програмним продуктом [12, 13]. Гейміфікація трансформує пасивне споживання контенту мобільної книгарні на активний процес, де кожна дія користувача – від відкриття застосунку до завершення читання розділу – стає частиною керованої системи досягнень. З інженерної точки зору, цей процес описується як створення «петель зворотного зв'язку» (Feedback Loops), де система аналізує вхідні дані про активність суб'єкта та генерує відповідні візуальні або статусні винагороди, що спонукають до подальшої взаємодії з функціоналом [53]. Кепп підкреслює, що ефективна гейміфікація завжди базується на чіткій цільовій поведінці: розробник має спочатку визначити, яку саме дію він хоче стимулювати, і лише потім обирати відповідну ігрову механіку [29].

Управління поведінкою через гейміфіковані інтерфейси базується на глибокому розумінні архітектури мотивації, зокрема через призму фреймворку Octalysis, розробленого Ю-кай Чоу [7]. Ця концепція виділяє вісім ключових стимулів, серед яких розвиток, дефіцит та почуття власності, що інтегруються в UI/UX дизайн через такі компоненти, як індикатори прогресу, віртуальні колекції та обмежені в часі події. Таким чином, гейміфікація виступає як надбудова над функціональною логікою застосунку, яка не змінює його основне призначення, але суттєво змінює шлях користувача (User Journey), роблячи його більш передбачуваним та емоційно насиченим [7]. Використання ігрових

елементів дозволяє розробникам ефективно долати проблему когнітивного опору, що часто виникає при засвоєнні нових функцій інтерфейсу. Норман зазначає, що добре спроєктований інтерфейс має відповідати ментальній моделі користувача, а ігрові механіки є ефективним інструментом для формування таких моделей [42]. Завдяки трансформації рутинного пошуку літератури у формат інтерактивного квесту, процес взаємодії стає більш природним. Такий підхід базується на класичній ієрархічній моделі, що включає рівні ігрових динамік, механік та конкретних компонентів, запропонованій Вербахом та Хантером [50].

Науковий підхід до гейміфікації як засобу керування поведінкою підтверджується дослідженнями Детердінга щодо «дизайну, орієнтованого на мотивацію», де обґрунтовується перехід від утилітарних інтерфейсів до систем, що здатні утримувати увагу користувача через задоволення базових психологічних потреб у компетентності та автономії [12, 14]. Важливим теоретичним підґрунтям є теорія самодетермінації Раяна та Дечі, яка стверджує, що ефективне управління поведінкою в IT-системах можливе лише за умови підтримки внутрішніх локусів контролю користувача [44]. Пінк доповнює цю думку, стверджуючи, що найстійкіша мотивація виникає тоді, коли людина відчуває майстерність, автономію та усвідомлює мету своїх дій [43] – саме ці три елементи закладено в основу системи рівнів, квестів та досягнень застосунку «SmartBook». Фундаментальним є також внесок Хамарі, чиї емпіричні дослідження доводять пряму кореляцію між впровадженням ігрових алгоритмів та зміною патернів використання цифрових платформ [24], що робить гейміфікацію незамінним інструментом сучасної програмної інженерії для оптимізації поведінкових метрик. Каньман додатково пояснює цей ефект через призму двосистемного мислення: ігрові механіки апелюють до швидкої інтуїтивної системи прийняття рішень, знижуючи когнітивне навантаження при взаємодії з інтерфейсом [28].

1.2 Огляд існуючих програмних рішень для мобільних книгарень

Сучасний ринок програмного забезпечення для мобільних пристроїв у сегменті книжкової дистрибуції представлений широким спектром рішень, що варіюються від вузькоспеціалізованих «читалок» до глобальних маркетплейсів [52]. Аналіз існуючих систем дозволяє класифікувати їх за архітектурними принципами побудови інтерфейсу та методами інтеграції ігрових механік. Огляд існуючих програмних рішень демонструє значну різницю між глобальними екосистемами та локальними українськими розробками в контексті архітектурної зрілості ігрових механік [13, 53]. Якщо світові лідери вже інтегрували гейміфікацію на рівні системних процесів та аналізу поведінкових даних у реальному часі, то вітчизняні застосунки наразі перебувають на етапі впровадження зовнішніх маркетингових інструментів для стимулювання активності. Купер та колеги підкреслюють, що порівняльний аналіз конкурентних рішень є обов'язковим етапом проєктування нового інтерфейсу: він дозволяє виявити архітектурні прогалини та сформулювати унікальну ціннісну пропозицію [10].

Глобально Amazon Kindle представляє найбільш технологічно досконалу модель «глибокої сервісної інтеграції». Основним інструментом управління поведінкою тут є модуль Reading Insights, який функціонує як складний аналітичний надбудовний компонент. Система автоматично збирає телеметрію кожної дії – від тривалості відображення сторінки до частоти використання словника. Завдяки технології Whispersync, ці дані синхронізуються між пристроями, дозволяючи формувати безперервні «ланцюжки читання». Програмна логіка Kindle орієнтована на внутрішню мотивацію: користувач отримує візуальне підтвердження свого інтелектуального прогресу, що мінімізує ефект «заміщення мотивації», характерний для систем із грошовими винагородами.

Goodreads демонструє підхід, заснований на «соціальній архітектурі». У цьому застосунку гейміфікація виступає як засіб публічної репрезентації досягнень. Головний механізм – Annual Reading Challenge, він дозволяє користувачеві встановлювати кількісні цілі, які стають видимими для його соціального графа. З інженерної точки зору, система базується на принципах соціального доказу, де прогрес-бари друзів та спільні обговорення виступають основними стимулами. Проте, через застарілість UI-шару, взаємодія з ігровими елементами часто є статичною, що знижує динаміку залучення порівняно з сучасними мобільними іграми.

Bookmate фокусується на «алгоритмічній гейміфікації». У цьому застосунку управління поведінкою реалізовано через персоналізовані «полиці» та рекомендаційні стрічки, що створюють ілюзію нескінченного дослідження контенту. Програмна логіка тут спрямована на усунення когнітивного опору при виборі нової книги, що є прикладом «невидимої» гейміфікації, де користувач залучається до процесу не через бали, а через високу релевантність запропонованих сценаріїв взаємодії.

Серед українських програмних продуктів Librarius є прикладом функціонально-центричного підходу. У його архітектурі гейміфікація представлена лише базовими візуальними маркерами, такими як відсотковий прогрес прочитання окремої книги. Розробники зосередилися на технічній стабільності та зручності інтерфейсу читця, проте відсутність системних нагород чи ігрових рівнів обмежує можливості для керованого формування звички читання [16]. Librarius має потенціал для надбудови ігрової логіки над уже існуючою стабільною базою збору даних, що відповідає підходу поступового впровадження гейміфікації, описаному Кеппом [29].

Видавництво та книжковий клуб «КСД» представляє специфічну для українського ринку модель, що базується на ієрархічній системі клубних привілеїв. Архітектура цього рішення фокусується на механіці «закритої

спільноти», де статус користувача безпосередньо керує доступним функціоналом та контентом. Програмна логіка побудована навколо автоматизованого підвищення рангу клієнта залежно від тривалості членства та регулярності транзакцій. В інтерфейсі це реалізовано через персоналізовані маркери статусу та розблокування ексклюзивних розділів каталогу, що є класичним прикладом використання механіки «дефіциту» за фреймворком Octalysis [7]. Технічно це забезпечується складними тригерами в базі даних, які синхронізують історію активності з правами доступу до API. Головною особливістю підходу «КСД» є створення зовнішньої мотивації через систему рівнів та знижок. Проте, на відміну від глобальних лідерів на кшталт Amazon Kindle, гейміфікація в «КСД» обмежена етапом придбання книги. В архітектурі застосунку відсутні інструменти для відстеження самого процесу читання в реальному часі, що створює розрив у користувацькому досвіді (UX Gap) між покупкою та споживанням контенту [50]. Аріелі пояснює цей феномен через концепцію «ірраціональної поведінки»: користувач, який отримав зовнішню винагороду за покупку, але не отримує підкріплення під час читання, поступово втрачає мотивацію до завершення книги [1].

Абук демонструє найбільш персоналізований підхід серед українських розробників. Хоча в застосунку відсутні класичні ігрові атрибути (як-от бейджі), він активно використовує елементи «емоційного дизайну» [42]. Система аналізує вподобання та формує добірки, що створює відчуття персонального кураторства – механіка, що відповідає Core Drive 4 (Власність) фреймворку Octalysis [7]. Для переходу на рівень глобальних конкурентів Абуку необхідно формалізувати статистику прослуховування у повноцінну систему досягнень, що дозволить перетворити аудіо-досвід на ігровий квест відповідно до методики, запропонованої Вербахом та Хантером [50].

Yakaboo займає проміжне положення, наближаючись до гібридної моделі. Поєднання великої бази даних про купівельні звички з розвиненою рекомендаційною системою дозволяє застосунку ефективно керувати увагою

користувача [52]. Проте, як і в інших локальних гравців, гейміфікація в Yakaboo залишається переважно «зовнішньою» (бонуси, знижки), тоді як світовий досвід вказує на ефективність «внутрішніх» ігрових механік – самовдосконалення, статус, колекціонування [53]. Детермінінг та Наке підкреслюють, що зовнішня мотивація через матеріальні винагороди є ефективною лише короткостроково і не формує стійкої залученості без підкріплення внутрішніми мотиваторами [14].

Таким чином, порівняння світового та локального ринків виявляє суттєвий простір для наукового пошуку. Українські продукти мають розвинену інфраструктуру для транзакцій, але потребують впровадження методик, які б дозволили ігровим елементам стати частиною самого процесу читання [24, 52]. Саме цю прогалину адресує методика, розроблена у цій роботі: інтеграція гейміфікованих елементів безпосередньо у читацький досвід, а не лише у транзакційну частину застосунку [24].

1.3 Порівняльний аналіз архітектурних підходів до інтеграції ігрових елементів

Порівняльний аналіз архітектурних підходів до інтеграції ігрових елементів дозволяє класифікувати існуючі рішення за ступенем їхнього впливу на програмну логіку та користувацький досвід. Ці підходи розділяють за рівнем абстракції: від поверхневих інтерфейсних надбудов до глибокої сервісної інтеграції в ядро системи [50]. Розуміння цих відмінностей є критичним для розробки ефективної інформаційної моделі, яка б не просто імітувала гру, а керувала поведінкою користувача на алгоритмічному рівні [34, 47].

Перший та найбільш розповсюджений підхід – транзакційна архітектура (Marketing-driven Architecture). Він домінує в українських продуктах, таких як КСД та Yakaboo. У цій моделі ігрова логіка (нарахування балів, статусів чи бонусів) реалізована як окремий модуль, що активується лише в момент здійснення фінансової операції. З технічної точки зору, це реалізується через тригери в базі даних, які оновлюють стан лояльності клієнта після

підтвердження оплати [47]. Перевагою є простота впровадження та прямий вплив на комерційні показники, проте головним недоліком залишається розрив у користувацькому досвіді (UX Gap): користувач перестає бути «гравцем» одразу після покупки, оскільки процес читання ніяк не фіксується системою [53]. Аріелі пояснює цю проблему через концепцію «роз'єднаності підкріплення»: винагорода, відокремлена від цільової дії у часі, втрачає свою мотиваційну силу [1].

Другий підхід – сервісно-орієнтована архітектура з низькорівневим трекінгом (Event-driven Service Architecture), яскравим представником якої є Amazon Kindle. Тут гейміфікація інтегрована безпосередньо в середовище споживання контенту. Система збирає телеметрію кожної дії користувача (гортання сторінок, час активності екрана) та обробляє ці події в реальному часі через спеціалізований мікросервіс [18]. Це дозволяє створювати складніші ігрові петлі, такі як «щоденні серії», які базуються на фактичній поведінці читача, а не на покупках. Така модель забезпечує найвищий рівень утримання, оскільки ігрові механіки підтримують внутрішню мотивацію до саморозвитку [44]. Еван визначає подібну архітектуру як «доменно-орієнтовану»: ігрова логіка є невід'ємною частиною доменної моделі, а не зовнішньою надбудовою [15].

Третій підхід – соціально-мережева архітектура (Social-graph Architecture), характерна для платформ типу Goodreads. В основі лежить реляційна модель, де ігровий прогрес одного користувача є тригером для сповіщення або взаємодії з іншими вузлами мережі [24]. Основна механіка – публічні виклики – базується на мережевому ефекті та соціальному підкріпленні, що відповідає Core Drive 5 (Соціальний вплив) фреймворку Octalysis [7]. Технічна складність тут полягає в забезпеченні цілісності даних при масових оновленнях стрічки активності та управлінні великою кількістю зовнішніх запитів до API соціальних функцій [47]. Чалдіні визначає соціальний доказ як один із найпотужніших механізмів

впливу на поведінку: бачення прогресу інших користувачів суттєво підвищує власну мотивацію [8].

Четвертий підхід – адаптивна алгоритмічна архітектура (AI-driven / Invisible Gamification), що спостерігається у Bookmate та Абук. Це найбільш сучасний підхід, де замість явних атрибутів гри (балів чи значків) використовуються предиктивні моделі [14]. Система керує поведінкою через фільтрацію контенту та персоналізовані сценарії, що знижує когнітивне навантаження відповідно до принципу Міллера – не більше семи одиниць інформації одночасно [38]. Це «невидима» гейміфікація, де роль ігрового майстра виконує алгоритм рекомендацій, що веде користувача за оптимальним шляхом залучення [7]. Норман визначає такий підхід як вищий рівень проєктування досвіду: система передбачає потреби користувача замість того, щоб реагувати на них [42].

Підсумовуючи порівняння, можна стверджувати, що для розробки методики інтеграції гейміфікації в мобільні книгарні найбільш перспективним є гібридний підхід [13]. Він має поєднувати стабільність транзакційної моделі локальних гравців із глибиною поведінкового трекінгу глобальних систем. Це дозволить створити інформаційну систему, де ігрові механіки супроводжують користувача на всіх етапах: від вибору книги до її повного прочитання, що є ключовим для підвищення показників LTV та лояльності в сучасному IT-середовищі [16, 53].

ЗАСТОСУНОК	ТИП АРХІТЕКТУРИ	ОСНОВНИЙ ФОКУС ГЕЙМІФІКАЦІЇ	КЛЮЧОВИЙ ІНСТРУМЕНТ ЗАЛУЧЕННЯ	РІВЕНЬ АВТОМАТИЗАЦІЇ ЗБОРУ ДАНИХ
AMAZON KINDLE	Сервісно-орієнтована	Процес читання та саморозвиток	Reading Insights (серії днів, значки)	Високий (автоматичний трекінг сторінок)
GOODREADS	Соціально-мережева	Соціальне підкріплення та статус	Annual Reading Challenge (публічні цілі)	Середній (ручне або напівавтоматичне введення)
BOOKMATE	Адаптивна алгоритмічна	Дослідження та усунення когнітивного опору	Персоналізовані рекомендаційні полиці	Високий (аналіз поведінкових логів)
КСД	Транзакційна (Клубна)	Ексклюзивність та фінансова вигода	Ієрархія статусів членства (ранги)	Низький (базується на фактах покупок)
LIBRARIUS	Функціонально-центрична	Візуалізація поточного прогресу	Відсотковий індикатор прочитання	Середній (локальний трекінг у ридері)
АБУК	Емоційно-центрична	Аудіальний досвід та лояльність	Персоналізовані добірки та звернення	Середній (статистика прослуховування)
YAKABOO	Гібридна (Маркетплейс)	Купівельна активність та вибір	Бонусна система та смарт-рекомендації	Високий (Big Data аналіз транзакцій)

Таблиця 1.3 – Порівняльна характеристика мобільних застосунків за параметрами гейміфікації та архітектури

Таким чином, порівняльний аналіз архітектурних підходів дозволяє сформулювати три ключові висновки. По-перше, глобальні лідери – Kindle та Bookmate – фокусуються на процесі споживання контенту (як користувач читає), тоді як українські продукти – КСД та Yakaboo – переважно на процесі придбання (як користувач купує). Цей розрив підтверджує наявність суттєвого потенціалу для впровадження внутрішніх ігрових механік у локальні застосунки [24, 52]. Хамарі та колеги доводять, що гейміфікація є найефективнішою саме тоді, коли ігрові механіки вбудовані безпосередньо у цільову дію користувача, а не є зовнішнім доповненням до неї [24].

По-друге, перехід до високорівневої гейміфікації потребує впровадження систем збору телеметрії в реальному часі (Event-driven архітектура), що наразі є точкою росту для вітчизняного програмного забезпечення [47]. Саммервілль

визначає подібні системи збору подій як фундамент для побудови адаптивної бізнес-логіки: без достовірних даних про поведінку користувача неможливо побудувати ефективну систему тригерів та нагород. Саме цей принцип реалізовано у діаграмі послідовності підрозділу 2.1 цієї роботи, де кожна зміна сторінки є окремою подією, що ініціює ланцюг ігрових обчислень [47].

По-третє, у зарубіжних аналогах переважає робота з внутрішньою мотивацією – компетентність, майстерність, автономія, – тоді як локальні рішення будуються на зовнішніх стимулах: знижках, бонусах та статусі покупця [44, 43]. Ряан та Дечі у теорії самодетермінації доводять, що зовнішня мотивація є ефективною лише короткостроково і не формує стійкої залученості без підкріплення внутрішніми потребами [44]. Пінк доповнює цю думку, стверджуючи, що системи, побудовані на принципах майстерності та автономії, демонструють значно вищі показники довгострокового утримання користувачів [43]. Саме подолання цього дисбалансу є ключовою науковою задачею цієї роботи: розроблена методика інтеграції гейміфікованих елементів орієнтована на формування внутрішньої мотивації через систему рівнів, квестів та досягнень, що супроводжують користувача безпосередньо під час читання, а не лише під час придбання контенту [7, 53].

1.4 Формулювання технічного завдання на розробку системи

Метою розробки є створення мобільного застосунку «SmartBook», який поєднує функціонал електронної книгарні з розгалуженою системою гейміфікації для управління поведінкою користувача. Технічне завдання базується на стандартах формування вимог до програмного забезпечення, визначає архітектурні та функціональні межі проєктування мобільної книгарні з елементами гейміфікації [51]. Основний акцент зміщується з чистої дистрибуції контенту на створення динамічного середовища, що стимулює активність користувача через систему винагород та зворотного зв'язку [53]. Вігерс та Біті підкреслюють, що якісне технічне завдання має охоплювати одночасно

функціональні, інтерфейсні та нефункціональні вимоги, оскільки лише їх сукупність забезпечує цілісність архітектурного рішення [51].

Функціональні вимоги до гейміфікованих модулів. Центральним елементом системи є модуль управління станом користувача, який повинен забезпечувати трекінг прогресу читання: алгоритм має вираховувати прогрес на основі поточної сторінки відносно загальної кількості та автоматично оновлювати статус у профілі та базі даних при кожній зміні екрана читання [18, 19]. Система рівнів та XP (Experience Points) передбачає розробку логіки нарахування балів досвіду за дії: придбання книги, завершення розділу, щоденний вхід. Досягнення певної кількості XP є тригером для зміни рівня користувача та розблокування нових візуальних елементів або досягнень [7, 53]. Механізм щоденних завдань реалізується через циклічні квести з оновленням кожні 24 години, наприклад «Прочитати 20 сторінок» або «Додати 2 книги до списку бажань» [16]. Фогг визначає подібні мікрозавдання як ключовий інструмент формування стійкої звички: малі досяжні цілі знижують психологічний бар'єр входу та формують регулярну взаємодію з продуктом [20].

Вимоги до інтерфейсу та взаємодії. Інтерфейс має працювати як «жива система», де кожна ігрова подія знаходить негайний візуальний відгук [46]. Час відгуку на ігрові події – анімація отримання балів, відкриття нагороди – не має перевищувати 200 мс, щоб забезпечити відчуття плавності та ігрового занурення [41]. Нільсен встановлює цей поріг як критичний: затримка понад 200 мс руйнує відчуття безперервності взаємодії та сприймається користувачем як збій системи [41]. Компоненти мають змінювати свій стан залежно від прогресу – прогрес-бар змінює колір з нейтрального на зелений при досягненні порогу у 80% цілі, що візуально сигналізує про близьке завершення завдання [45]. Проєктування здійснюється згідно зі стандартом WCAG 2.1 рівня AA [49], що передбачає високу контрастність тексту на ігрових віджетах та підтримку екранних дикторів TalkBack та VoiceOver для незрячих користувачів [26, 35]. Шнайдерман визначає інклюзивність як фундаментальний принцип

проєктування інтерфейсів: система, недоступна для частини аудиторії, не може вважатись якісним рішенням [46].

Вимоги до структури даних. Для забезпечення стабільної роботи ігрових механік архітектура даних повинна бути гнучкою та відмовостійкою [47]. Система повинна зберігати стан прогресу у локальному сховищі SQLite, а при відновленні з'єднання синхронізувати дані з хмарою без конфліктів версій [30, 19]. Дані про досягнення зберігаються у форматі JSON для легкого масштабування, що дозволяє додавати нові ідентифікатори нагород без зміни схеми бази даних [15]. Еванс підкреслює, що гнучкість схеми даних є критичною для систем із постійно еволюціонуючою бізнес-логікою – яким і є гейміфікована система з новими квестами та досягненнями [15].

Нефункціональні вимоги. Адаптивність передбачає, що гейміфіковані віджети – шкали досвіду, вітрини нагород – мають коректно масштабуватися на екранах від 320dp до 600dp+, зберігаючи пропорції та читабельність елементів відповідно до рекомендацій платформ [26, 35]. Модульність архітектури базується на принципах композиції: нові ігрові механіки, наприклад «Турнірна таблиця», мають впроваджуватися як окремі модулі без рефакторингу основної логіки навігації [34, 47]. Мартін визначає модульність як ключову характеристику чистої архітектури: система, де зміна одного компонента не вимагає змін в інших, є стійкою до масштабування та технічного боргу [34].

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ АРХІТЕКТУРИ МОБІЛЬНОЇ КНИГАРНІ

2.1 Побудова моделі використання: ролі та сценарії взаємодії

Побудова моделі використання системи «SmartBook» є етапом формалізації функціональних меж застосунку, де теоретичні вимоги ТЗ трансформуються у конкретні сценарії взаємодії між акторами та програмними модулями. У межах цього процесу, згідно з методологією Коберна, виділяються три ключові ролі, кожна з яких має специфічний набір повноважень та вплив на стан системи [9]. Фаулер підкреслює, що чітке визначення акторів є першим кроком до побудови зрозумілої та несуперечливої архітектури застосунку [21].

Основним актором виступає Користувач (Читач), чия активність – від перегляду каталогу до безпосереднього читання – є основним джерелом вхідних даних для ігрового рушія [53]. Допоміжну роль виконує Адміністратор, який керує контентом та параметрами гейміфікації. Роль системного актора відведена Системному таймеру, що автоматично ініціює скидання щоденних циклів квестів кожні 24 години без прямого втручання людини – підхід, що відповідає принципу автономних тригерів у сучасних програмних системах [47].

Центральний сценарій взаємодії зосереджений навколо процесу читання, який у гейміфікованій системі перестає бути лінійним. Коли користувач активує модуль ридера, система ініціює фоновий процес трекінгу, що при кожній зміні сторінки викликає вкладений сценарій оновлення прогресу [18]. Цей алгоритм не лише перераховує відсоток прочитаного, а й виступає тригером для нарахування очок досвіду (XP), базуючись на принципах «петель залучення» [53]. Еял визначає подібний цикл як основу формування звички: дія – тригер – нагорода – інвестиція [16]. Якщо накопичений досвід досягає критичної позначки, автоматично запускається сценарій розширення – видача нагороди або підвищення рівня, що супроводжується візуальною реакцією інтерфейсу з часом відгуку до 200 мс відповідно до вимог підрозділу 1.4 [41]. Такий підхід

забезпечує безперервний цикл зворотного зв'язку, де кожна технічна операція з даними має миттєве відображення в ігровій логіці профілю [45].

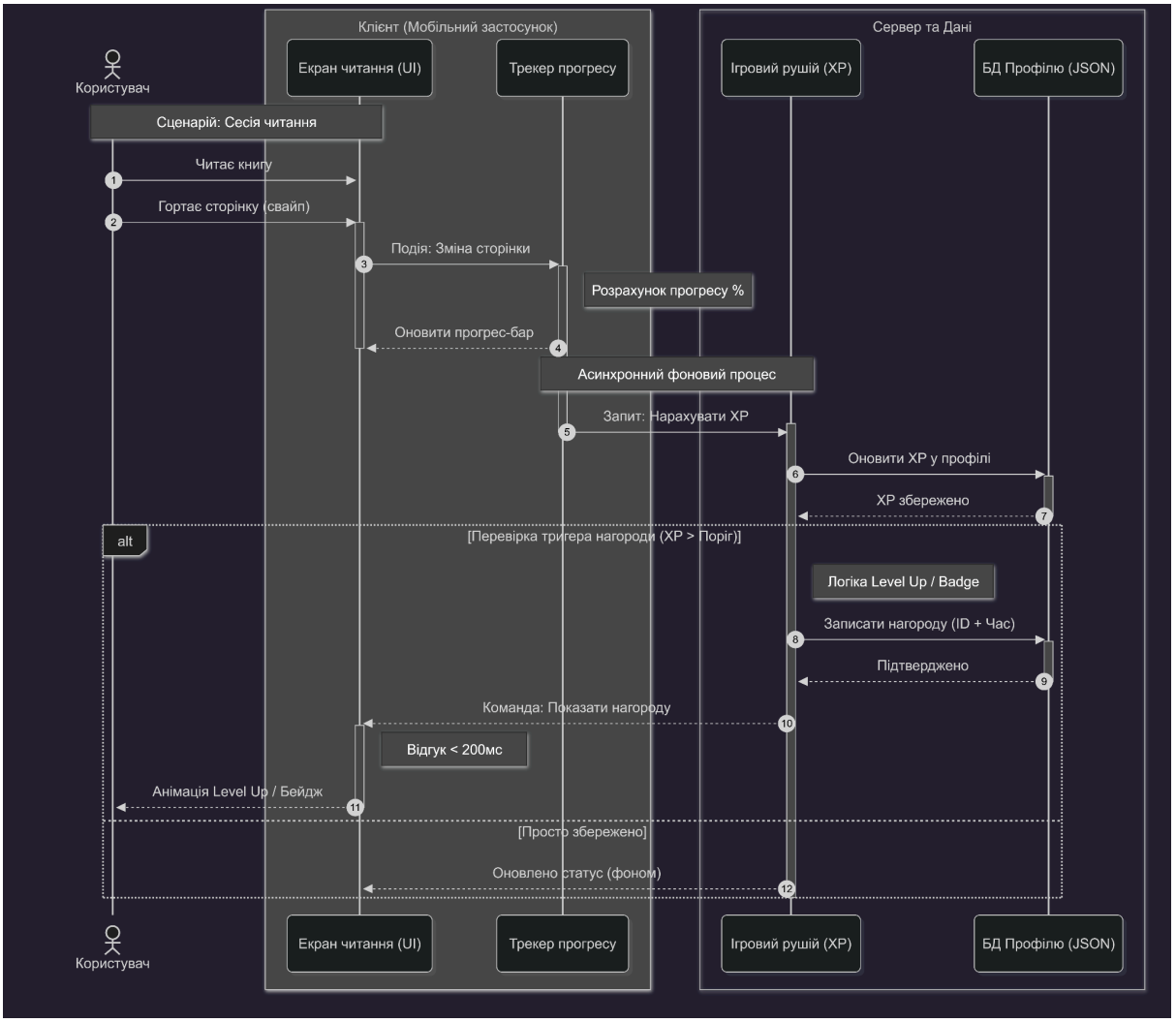


Рисунок 2.1 – Візуалізація архітектурної взаємодії

На діаграмі послідовності зображено повний цикл взаємодії між користувачем та програмними модулями під час сесії читання. Архітектурно система розділена на клієнтську частину та серверну частину відповідно до принципів розподіленої архітектури [47]. На першому етапі (кроки 1-4) користувач гортає сторінку, екран читання передає сигнал до трекера прогресу, який розраховує новий відсоток прочитаного безпосередньо на пристрої для забезпечення миттєвого відгуку [18]. На другому етапі (крок 5) трекер асинхронно відправляє запит до ігрового рушія на сервері – користувач продовжує читати, не чекаючи відповіді [19]. На третьому етапі (кроки 6-7)

ігровий рушій оновлює бали досвіду у базі даних профілю [30]. Блок альтернативної логіки реалізує два сценарії: при виконанні умови XP > Поріг (кроки 8-11) рушій записує нову нагороду та запускає анімацію «Level Up» з часом відгуку менше 200 мс [41, 45]; при звичайному збереженні (крок 12) інтерфейс оновлюється у фоновому режимі без відволікання користувача від читання [46].

Окремим доменом взаємодії є управління щоденними завданнями та системою досягнень. Діаграма демонструє реалізацію принципу Offline-first відповідно до вимог відмовостійкості підрозділу 1.4 [30, 19]. При відкритті розділу квестів застосунок спочатку звертається до локального кешу для миттєвого відображення збережених даних, після чого паралельно синхронізується з серверним ігровим рушієм [47]. Сценарій моніторингу та динамічної адаптації (кроки 8–9) реалізує вимогу ТЗ щодо реактивності інтерфейсу: при досягненні порогу у 80% спрацьовує внутрішній тригер, що змінює стан UI-компонентів – колір та форму – відповідно до принципів мікроінтеракцій [45]. При цьому система здійснює програмний контроль контрастності при зміні станів згідно зі стандартом WCAG 2.1 [49]. Сценарій взаємодії з нагородами (кроки 10–13) описує зчитування масиву achievement_id із часовими мітками з JSON-профілю для побудови галереї нагород [15]

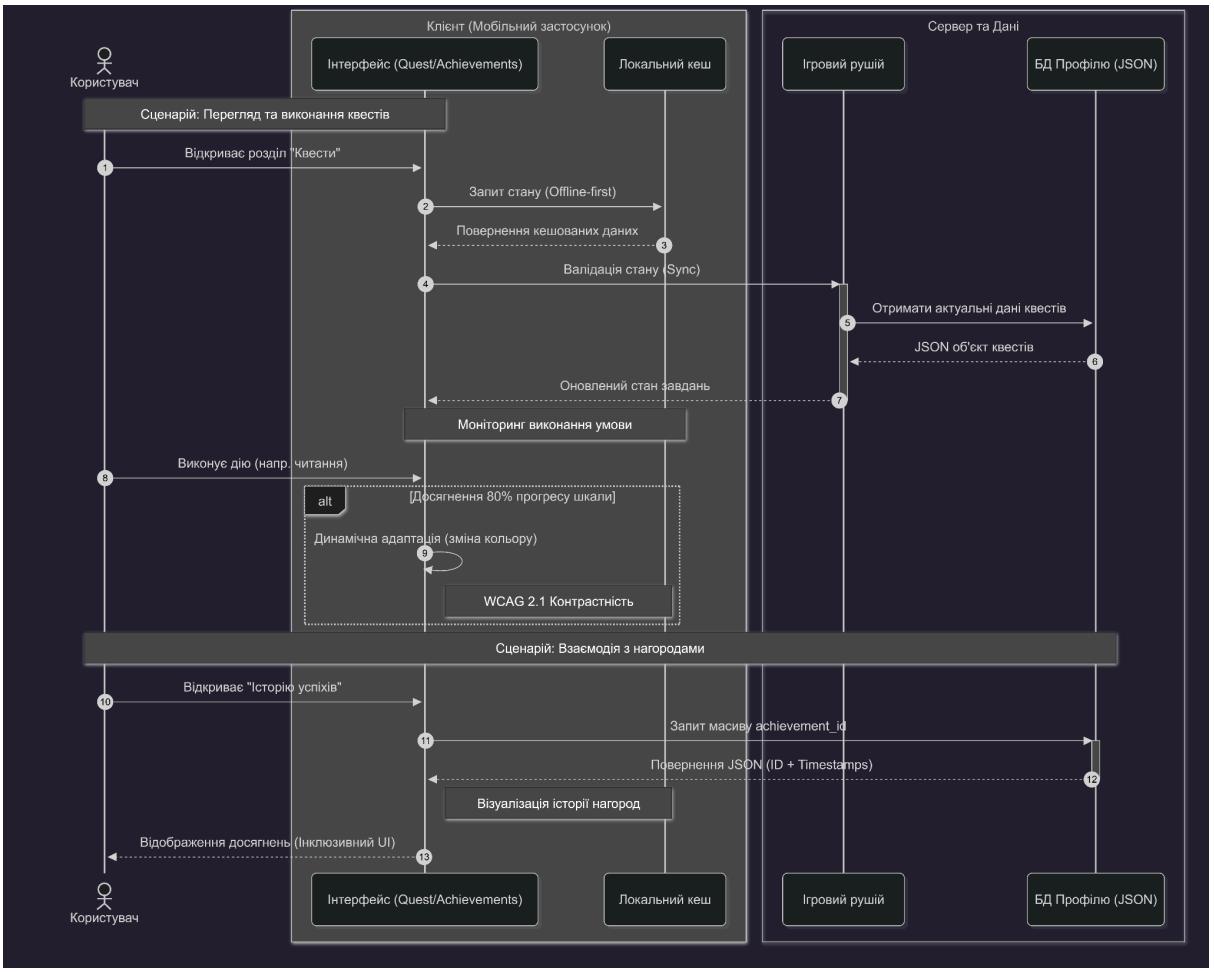


Рисунок 2.2 – Діаграма послідовності, яка відображає логіку перевірки завдань та звернення до JSON-профілю

На діаграмі представлено два ключові сценарії взаємодії, що відбуваються в межах гібридної архітектури клієнт-сервер. Основна увага приділяється безперервності користувацького досвіду та динамічній адаптації інтерфейсу [47].

У першому сценарії – перегляду та виконання квестів (кроки 1-7) – реалізується принцип Offline-first, що забезпечує стабільність системи навіть при нестабільному інтернет-з'єднанні [30]. При ініціації (крок 1) користувач відкриває розділ квестів. Гібридна перевірка (кроки 2-3) передбачає звернення застосунку спочатку до локального кешу, що дозволяє миттєво відобразити дані, збережені під час попередньої сесії [30, 19]. Валідація (кроки 4-7) відбувається паралельно: система надсилає запит до ігрового рушія на сервері для

синхронізації, сервер зчитує актуальний JSON-об'єкт квестів із бази даних і повертає оновлений стан [15, 19]. Це гарантує, що користувач завжди бачить актуальні завдання незалежно від якості мережевого з'єднання [47].

У другому сценарії – моніторингу та динамічної адаптації (кроки 8-9) – візуалізовано технічне виконання вимоги ТЗ щодо реактивності інтерфейсу відповідно до підрозділу 1.4 [46]. При виконанні цільової дії (крок 8) – читанні певної кількості сторінок – коли лічильник досягає порогу у 80%, спрацьовує внутрішній тригер (крок 9). Система не просто оновлює цифри, а змінює стан UI-компонентів – колір та форму – відповідно до принципів мікроінтеракцій [45]. При цьому здійснюється програмний контроль контрастності при зміні станів згідно зі стандартом WCAG 2.1 [49], що забезпечує інклюзивність інтерфейсу для всіх категорій користувачів [26, 35].

У третьому сценарії – взаємодії з нагородами (кроки 10-13) – описується робота з персональними даними успіху. При відкритті історії успіхів інтерфейс робить запит до бази даних (кроки 10-11). Система витягує масив achievement_id разом із часовими мітками (крок 12) відповідно до JSON-структури профілю, описаної у підрозділі 2.2 [15]. Завдяки зчитуванню конкретних ідентифікаторів застосунок будує графічну галерею нагород на основі легких текстових даних із профілю (крок 13), що відповідає принципу мінімізації обсягу даних, що передаються між сервером і клієнтом [34, 47].

Формалізація цих сценаріїв через модель використання дозволяє чітко розмежувати функціональну логіку книгарні та механізми управління поведінкою [21]. Це створює надійний фундамент для подальшого проєктування структури даних, де кожен Use Case знаходить своє втілення у конкретних масивах та об'єктах профілю користувача [15, 34]. Таким чином, модель використання виступає сполучною ланкою між вимогами до ігрових модулів та їхньою програмною реалізацією, забезпечуючи модульність системи, що є критичним аспектом сучасної програмної інженерії [47, 51].

2.2 Проєктування структури даних: атрибути для збереження прогресу користувача

Проєктування структури даних для гейміфікованої системи мобільної книгарні базується на принципі документо-орієнтованої архітектури, що дозволяє об'єднати всі динамічні показники користувача в межах єдиної ієрархічної моделі [15]. Такий підхід забезпечує максимальну швидкість обробки даних, оскільки мобільний застосунок отримує повний стан ігрового профілю за один запит до бази даних [19]. Еванс підкреслює, що правильно спроєктована доменна модель даних є фундаментом для побудови стійкої бізнес-логіки: зміни у вимогах мають мінімальний вплив на архітектуру, якщо структура даних відображає реальну предметну область [15]. Це критично для виконання вимог ТЗ щодо продуктивності підрозділу 1.4, оскільки миттєве оновлення інтерфейсу безпосередньо залежить від швидкості зчитування поточних значень прогресу та накопиченого досвіду [41].

Центральним аспектом проєктування є забезпечення цілісності через сувору типізацію атрибутів [48]. Числові показники, такі як бали досвіду (XP) чи баланс віртуальної валюти, зберігаються виключно у цілочисельному форматі, що дозволяє уникнути похибок при розрахунку рівнів за складними математичними моделями. Дарт як мова програмування з суворою статичною типізацією надає компілятору можливість виявляти некоректні типи даних на етапі збірки, що є критичним для запобігання помилкам нарахування XP або ігрової валюти [17]. Кожна зміна стану обов'язково супроводжується часовою міткою у форматі ISO 8601, що є фундаментальною умовою для реалізації відмовостійкості та коректної синхронізації [19]. При роботі в офлайн-режимі система порівнює локальні та серверні мітки, запобігаючи втраті ігрового прогресу користувача відповідно до вимог підрозділу 1.4 [30]. Практична реалізація описаної структури даних виконана мовою Dart у середовищі Flutter, обґрунтування вибору яких наведено у підрозділі 2.4. Мартін визначає чітку типізацію даних як один із ключових принципів чистої архітектури: некоректні

типи є джерелом прихованих помилок, що проявляються лише в продакшн-середовищі [34].

Атрибути системи досвіду та рівнів (XP Metrics). Блок атрибутів системи досвіду та рівнів (XP Metrics) є фундаментом довгострокової мотивації користувача, оскільки він перетворює абстрактний процес читання на вимірюваний цифровий прогрес [7]. Проєктування цих метрик базується на принципах ігрового дизайну та математичного моделювання поведінки користувача [53]. Хамарі та колеги емпірично підтверджують, що системи вимірюваного прогресу є найбільш ефективним механізмом довгострокового утримання у гейміфікованих застосунках [24].

Загальна кількість очок досвіду (total_xp). Це кумулятивний показник, який слугує «історією цінності» користувача в системі. Зберігання цього атрибута у цілочисельному форматі (integer) є критичним для забезпечення точності при великих обсягах даних [17]. Кожна дія в застосунку (прочитана сторінка, завершена книга, написаний відгук) має свою вагу в XP. Важливо, що цей показник є незмінним у бік зменшення, що створює у користувача відчуття постійного зростання та накопиченого інтелектуального капіталу – механіка, що відповідає Core Drive 4 (Власність) фреймворку Octalysis [7].

Поточний рівень користувача (current_level). Цей атрибут є дискретним відображенням статусу користувача. Він розраховується на основі total_xp за формулою геометричної прогресії з коефіцієнтом 1.1: поріг кожного наступного рівня визначається як $100 \times 1.1^{(level - 1)}$. Використання нелінійної прогресії дозволяє підтримувати високий темп винагород на початку та створювати відчуття елітарності на вищих етапах, що відповідає механіці Core Drive 2 фреймворку Octalysis [7]. Чеік-Мусса та колеги підтверджують, що нелінійні моделі прогресу є ефективнішими за лінійні з точки зору підтримки мотивації на різних етапах взаємодії з продуктом [6]. З точки зору архітектури, зміна

значення current_level виступає головним тригером для оновлення UI та розблокування нових функцій або нагород [18].

Досвід до наступного рівня (xp_to_next_level). На відміну від статичного total_xp, цей похідний атрибут реалізований як обчислюваний getter і розраховується як різниця між порогом наступного рівня та поточним значенням total_xp. Він є критично важливим для візуальної складової інтерфейсу – саме на його основі заповнюється шкала прогресу [45]. Наявність чіткої цифрової дистанції до цілі використовує психологічний ефект наближення до мети (Goal Gradient Effect), що стимулює користувача дочитати главу або книгу, щоб швидше заповнити залишок шкали [6]. Ряан та Дечі визначають подібні механіки як підтримку потреби у компетентності – одного з трьох базових психологічних потреб теорії самодетермінації [44].

Практичну реалізацію описаних атрибутів наведено у рисунку 2.3.

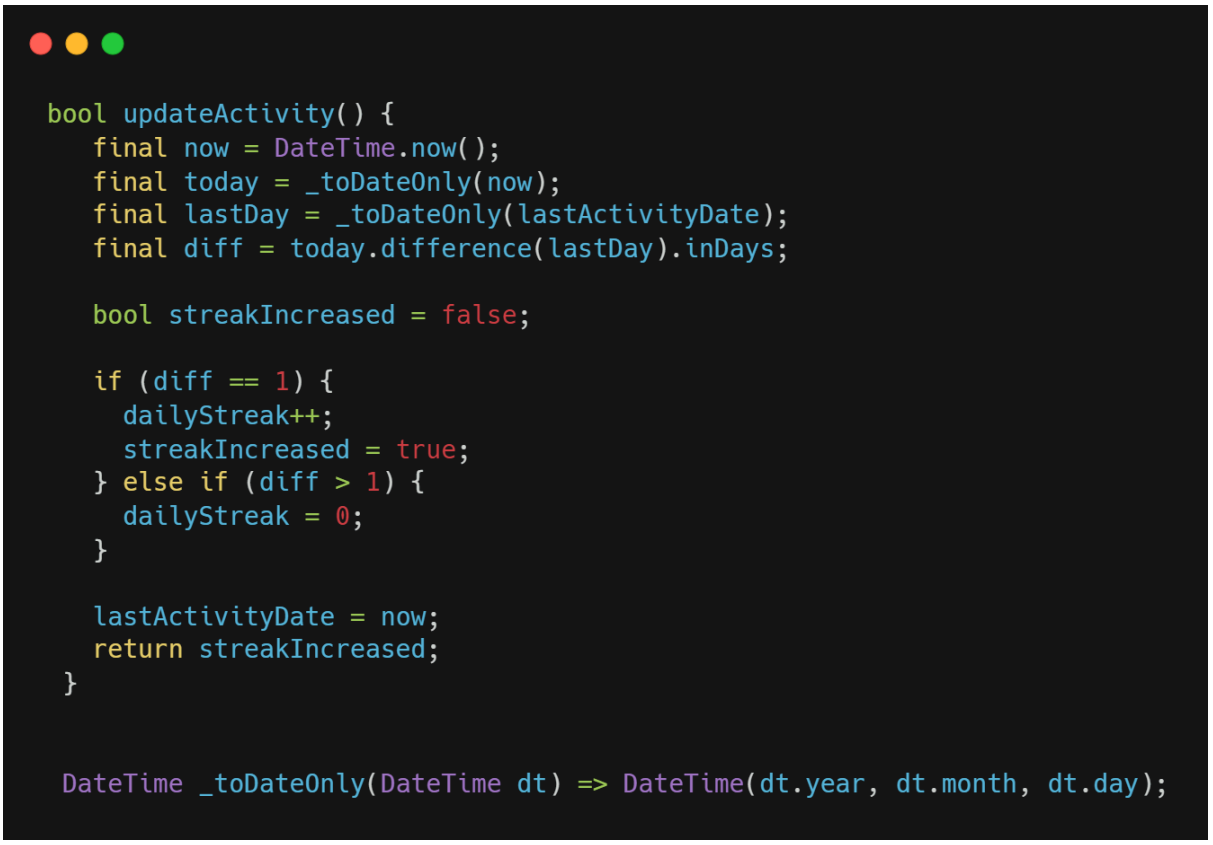
```dart
import 'dart:math';

class UserProgress {
  int totalXp;
  int currentLevel;
  int dailyStreak;
  DateTime lastActivityDate;

  UserProgress({
    this.totalXp = 0,
    this.currentLevel = 1,
    this.dailyStreak = 0,
    DateTime? lastActivityDate,
  }) : lastActivityDate = lastActivityDate ?? DateTime.now();


  int addExperience(int points) {
    if (points <= 0) return 0;
    totalXp += points;
    return _checkLevelUp();
  }

  int _checkLevelUp() {
    int levelsGained = 0;
    while (totalXp >= _calculateThresholdForLevel(currentLevel + 1)) {
      currentLevel++;
      levelsGained++;
      print('Рівень підвищено! Поточний рівень: $currentLevel');
    }
    return levelsGained;
  }

  int _calculateThresholdForLevel(int level) {
    if (level <= 1) return 0;
    return (100 * pow(1.1, level - 1)).round();
  }

  int get xpToNextLevel {
    return max(0, _calculateThresholdForLevel(currentLevel + 1) - totalXp);
  }

  double get levelProgressPercentage {
    final currentThreshold = _calculateThresholdForLevel(currentLevel);
    final nextThreshold = _calculateThresholdForLevel(currentLevel + 1);
    final totalNeededInLevel = nextThreshold - currentThreshold;

    if (totalNeededInLevel <= 0) return 1.0; // захист від ділення на нуль

    final xpInCurrentLevel = totalXp - currentThreshold;
    return (xpInCurrentLevel / totalNeededInLevel).clamp(0.0, 1.0);
  }
```

Рисунок 2.3 – Клас UserProgress для управління прогресом користувача

Лічильник днів безперервного читання (daily_streak). Це ключова метрика для забезпечення щоденного повернення користувача до застосунку (retention) [16]. Атрибут працює за логікою календарного лічильника: якщо наступного календарного дня користувач не зафіксував активність читання, лічильник автоматично обнуляється. Така поведінка реалізована через порівняння календарних дат, а не абсолютного 24-годинного інтервалу, що робить логіку інтуїтивно зрозумілою для користувача [17]. Високе значення daily_streak стає самостійною цінністю, яку користувач прагне зберегти – механіка уникнення втрат (Core Drive 8) фреймворку Octalysis [7]. Аріелі підтверджує цей ефект через теорію неприйняття втрат: страх втратити накопичений результат є значно сильнішим мотиватором, ніж бажання отримати нову винагороду [1].

Дата останньої активності (last_activity_date). Це повноцінний атрибут класу, який зберігає мітку часу останньої зафіксованої активності користувача у форматі ISO 8601 для коректної міжплатформної синхронізації [19]. Він є обов'язковою умовою коректної роботи лічильника daily_streak, оскільки саме на його основі система обчислює різницю між поточною датою та датою попередньої сесії. З точки зору архітектури, обидва атрибути – daily_streak та last_activity_date – є нерозривно пов'язаними і оновлюються синхронно в межах одного методу updateActivity(), що унеможливлює їх розсинхронізацію та відповідає принципу атомарності операцій [34].

Практичну реалізацію описаної логіки наведено у рисунку 2.4.



```dart
bool updateActivity() {
   final now = DateTime.now();
   final today = _toDateOnly(now);
   final lastDay = _toDateOnly(lastActivityDate);
   final diff = today.difference(lastDay).inDays;

   bool streakIncreased = false;

   if (diff == 1) {
     dailyStreak++;
     streakIncreased = true;
   } else if (diff > 1) {
     dailyStreak = 0;
   }

   lastActivityDate = now;
   return streakIncreased;
 }


DateTime _toDateOnly(DateTime dt) => DateTime(dt.year, dt.month, dt.day);
```

Рисунок 2.4 – Метод updateActivity() для відстеження щоденної активності користувача

Система віртуальної економіки (Virtual Currency) виконує роль мосту між ігровими досягненнями та реальною цінністю для користувача. Вона базується на механізмах позитивного підкріплення, де кожна корисна дія (читання, виконання квестів, активність у спільноті) конвертується в ліквідний ігровий ресурс [53]. Хамарі та Лехдонвірта доводять, що віртуальна валюта є одним із найефективніших інструментів монетизації залученості: вона створює внутрішній ринок, де поведінка користувача безпосередньо визначає його купівельну спроможність [25].

Клас віртуальної економіки (VirtualEconomy). Клас містить два ключові атрибути: softCurrency – цілочисельний лічильник балансу користувача, та lastTransactionTimestamp – мітка часу останньої фінансової операції. Конструктор класу реалізований через іменовані параметри з дефолтними значеннями: початковий баланс дорівнює нулю, а мітка часу ініціалізується

поточним моментом через оператор null-коалесценції (??) мови Dart, якщо значення не передано явно [17]. Метод earn() відповідає за нарахування ігрової валюти користувачу. Він приймає обов'язковий параметр amount та необов'язковий іменований параметр reason. Вхідні дані валідуються на початку методу: якщо передане значення є нульовим або від'ємним, система генерує виняток типу ArgumentError, що унеможливлює некоректне збільшення балансу [34]. Після успішної валідації метод збільшує баланс, оновлює мітку часу через допоміжний метод _updateTimestamp() та повертає актуальне значення softCurrency. Практичну реалізацію описаної логіки наведено у рисунку 2.5.

Практичну реалізацію описаної логіки наведено у рисунку 2.5.

```dart
import 'dart:math';

class VirtualEconomy {
  int softCurrency;
  DateTime lastTransactionTimestamp;

  VirtualEconomy({
    this.softCurrency = 0,
    DateTime? lastTransactionTimestamp,
  }) : lastTransactionTimestamp = lastTransactionTimestamp ?? DateTime.now();

  int earn(int amount, {String reason = 'activity'}) {
    if (amount <= 0) throw ArgumentError('Сума нарахування має бути більше 0');
    softCurrency += amount;
    _updateTimestamp();
    print(' +$amount монет ($reason). Баланс: $softCurrency');
    return softCurrency;
  }
```

Рисунок 2.5 – Клас VirtualEconomy: ініціалізація та метод нарахування ігрової валюти

Метод списання ігрової валюти (spend()). Даний метод реалізує логіку витрачання накопиченої валюти користувачем і є парним до методу earn(). Повертає булеве значення (bool), що дозволяє викликаючому коду однозначно визначити результат операції без використання винятків [17]. Метод реалізує дворівневу валідацію вхідних даних. На першому рівні перевіряється

коректність параметра amount: якщо значення є нульовим або від'ємним, генерується виняток ArgumentError. На другому рівні перевіряється достатність балансу через допоміжний метод canAfford(): якщо коштів недостатньо, метод повертає false без зміни стану об'єкта, що гарантує цілісність даних [34]. У разі успішного проходження обох перевірок виконується списання суми з балансу, оновлюється мітка часу та метод повертає true. Такий підхід відповідає принципу розділення відповідальності (Single Responsibility Principle): бізнес-логіка перевірки балансу винесена в окремий метод, що робить код повторно використовуваним і зручним для тестування [47]. Практичну реалізацію описаної логіки наведено у рисунку 2.6.

```
bool spend(int amount, {String reason = 'purchase'}) {
    if (amount <= 0) throw ArgumentError('Сума списання має бути більше 0');
    if (!canAfford(amount)) {
      print('❌ Недостатньо монет. Потрібно: $amount, є: $softCurrency');
      return false;
    }
    softCurrency -= amount;
    _updateTimestamp();
    print('💸 -$amount монет ($reason). Баланс: $softCurrency');
    return true;
  }
```

Рисунок 2.6 – Метод spend() для списання ігрової валюти

Метод перевірки достатності балансу (canAfford()). Даний метод є допоміжним і реалізує єдину відповідальність – перевірку того, чи має користувач достатній баланс для здійснення витрати. Завдяки синтаксису стрілочної функції (=>), метод записано в одному рядку, що підкреслює його лаконічність та відсутність побічних ефектів [17]. З архітектурної точки зору, винесення цієї перевірки в окремий метод є свідомим проєктним рішенням, що дозволяє використовувати canAfford() у двох контекстах: всередині методу spend() для захисту від некоректного списання, а також на рівні UI для динамічного блокування кнопки покупки [46]. Такий підхід відповідає

принципу DRY (Don't Repeat Yourself) та спрощує подальше тестування логіки [34]. Практичну реалізацію описаного методу наведено у рисунку 2.7.

Рисунок 2.7 – Метод canAfford() для перевірки достатності балансу

Метод вирішення конфліктів синхронізації (resolveSync()). Даний метод реалізує логіку узгодження локального та серверного стану балансу користувача в умовах роботи застосунку без доступу до мережі [30]. Він приймає параметр serverTimestamp типу DateTime та порівнює його з локальною міткою lastTransactionTimestamp. Логіка методу базується на принципі «остання операція має пріоритет» (Last Write Wins) [19]: якщо локальна мітка є новішою за серверну, локальні дані відправляються на сервер для оновлення – метод повертає true. В протилежному випадку серверні дані вважаються актуальнішими та ініціюється відновлення балансу з хмарного сховища – метод повертає false. Повернення булевого значення є свідомим архітектурним рішенням: викликаючий код отримує чіткий сигнал про напрямок синхронізації без необхідності аналізувати внутрішній стан об'єкта [47]. Практичну реалізацію описаного методу наведено у рисунку 2.8.

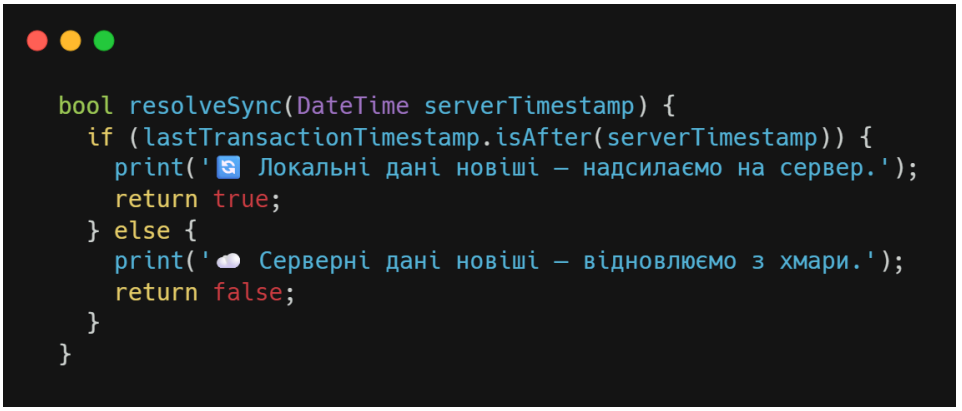


Рисунок 2.8 – Метод resolveSync() для вирішення конфліктів синхронізації

Похідний атрибут форматування мітки часу (lastTransactionIso8601). Даний getter відповідає за перетворення внутрішнього об'єкта DateTime у рядкове представлення стандарту ISO 8601 [19]. Логіка перетворення складається з двох послідовних кроків: спочатку викликається метод toUtc(), який конвертує локальний час у координований універсальний час (UTC), що є критично важливим для розподіленої системи з користувачами у різних часових поясах [47]. Після цього метод toIso8601String() перетворює об'єкт на рядок у міжнародному стандартному форматі. Інкапсуляція цього перетворення у getter відповідає принципу DRY та гарантує, що всі компоненти системи отримують мітку часу в єдиному стандартизованому форматі [34]. Практичну реалізацію описаного getter наведено у рисунку 2.9.

```
String get lastTransactionIso8601 =>
lastTransactionTimestamp.toUtc().toIso8601String();
```

Рисунок 2.9 – Getter lastTransactionIso8601 для форматування мітки часу

Згідно з фреймворком Octalysis Ю-кай Чоу, віртуальна валюта задіює четвертий драйвер мотивації – «Володіння та власність» [7]. Наявність накопичень стимулює користувача продовжувати взаємодію із застосунком, оскільки він відчуває відповідальність за свій «віртуальний капітал» [7]. Аріелі доповнює це поняттям «ефекту власності»: люди оцінюють речі, якими вже володіють, значно вище, ніж аналогічні речі, якими ще не володіють, – і цей психологічний механізм повністю переноситься на віртуальні активи [1].

Структура досягнень та квестів (Achievements & Quests). Для забезпечення гнучкості та можливості масштабування ігрових механік структура блоку досягнень та квестів проєктується як динамічний масив об'єктів [15]. Це дозволяє системі оперувати великою кількістю завдань, не перевантажуючи основний об'єкт профілю надлишковою інформацією. Кепп визначає динамічні системи досягнень як ключовий елемент освітньої

гейміфікації: можливість масштабування завдань без зміни базової архітектури є критичною для довгострокової підтримки продукту [29].

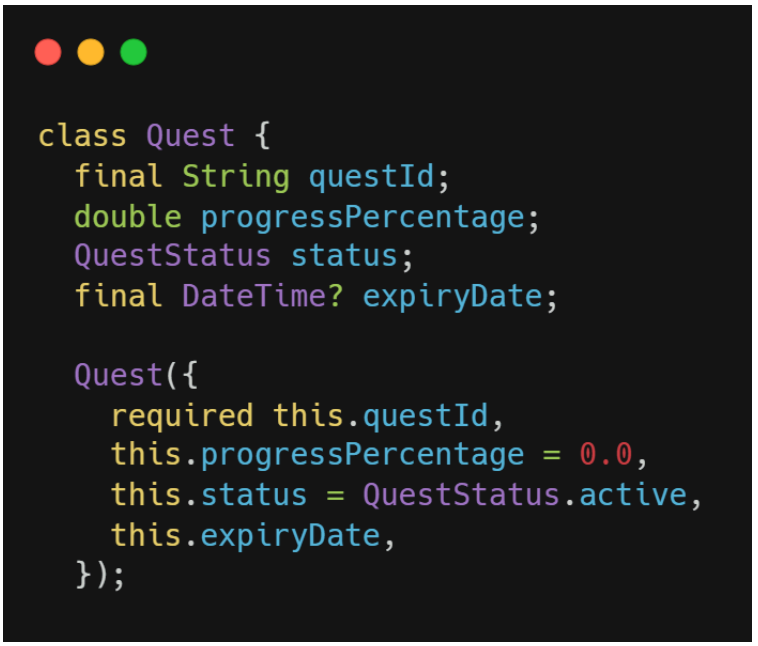
Перелік статусів квесту (QuestStatus). Для типізації стану квесту використовується конструкція enum, яка обмежує множину допустимих значень трьома чітко визначеними станами: active – квест активний і виконується, completed – умови квесту виконані, але нагороду ще не отримано, claimed – нагороду вже отримано. Використання enum замість рядкових констант є архітектурно виваженим рішенням: компілятор Dart гарантує, що жоден інший стан не може бути присвоєний атрибуту типу QuestStatus, що унеможливлює помилки через опечатки або некоректні значення на етапі компіляції [17]. Мартін визначає подібний підхід як реалізацію принципу «Make Illegal States Unrepresentable»: система має бути спроєктована так, щоб некоректний стан був фізично неможливим [34].

Модель досягнення (Achievement). Клас реалізує мінімальну незмінну структуру даних для зберігання факту розблокування досягнення. Обидва атрибути оголошено з модифікатором final, що гарантує незмінність об'єкта після створення – досягнення не може бути «скасовано» або змінено, що відповідає бізнес-логіці системи [34]. Атрибут achievementId є обов'язковим параметром конструктора (required), тоді як unlockedAt є необов'язковим: якщо мітка часу не передана явно, вона автоматично встановлюється на поточний момент через оператор (??) [17]. Метод toString() реалізований у форматі, зручному для логування та дебагу, і повертає рядкове представлення об'єкта з усіма ключовими атрибутами [47]. Практичну реалізацію описаних структур наведено у рисунку 2.10.

```dart
enum QuestStatus { active, completed, claimed }


class Achievement {
  final String achievementId;
  final DateTime unlockedAt;

  Achievement({
    required this.achievementId,
    DateTime? unlockedAt,
  }) : unlockedAt = unlockedAt ?? DateTime.now();

  @override
  String toString() =>
      'Achievement(id: $achievementId, unlockedAt: ${unlockedAt.toIso8601String()})';
}
```

Рисунок 2.10 – Перелік QuestStatus та модель даних Achievement

Модель квесту (Quest). Клас реалізує структуру даних для представлення окремого ігрового завдання та є центральним елементом системи квестів [15]. Атрибут questId оголошено як final String – незмінний рядковий ідентифікатор, що є обов'язковим параметром конструктора. Він слугує посиланням на глобальний довідник квестів і не може бути змінений після створення об'єкта [34]. Атрибут progressPercentage оголошено як змінний (double) з дефолтним значенням 0.0, що дозволяє системі динамічно оновлювати рівень виконання завдання в діапазоні від 0 до 100. Атрибут status типу QuestStatus ініціалізується значенням QuestStatus.active за замовчуванням. Атрибут expiryDate є необов'язковим (DateTime?) і може містити значення null для безстрокових завдань або конкретну дату завершення для часових квестів [17]. Конструктор класу використовує виключно іменовані параметри, що підвищує читабельність коду та унеможливлює помилки, пов'язані з неправильним порядком аргументів [47]. Практичну реалізацію описаної моделі наведено у рисунку 2.11.



```dart
class Quest {
  final String questId;
  double progressPercentage;
  QuestStatus status;
  final DateTime? expiryDate;

  Quest({
    required this.questId,
    this.progressPercentage = 0.0,
    this.status = QuestStatus.active,
    this.expiryDate,
  });
```

Рисунок 2.11 – Модель даних Quest: структура та ініціалізація

Метод оновлення прогресу (updateProgress()). Метод приймає нове значення прогресу та одразу застосовує .clamp(0.0, 100.0), що гарантує неможливість виходу за межі допустимого діапазону незалежно від переданого значення [17]. Після оновлення прогресу метод перевіряє умову завершення: якщо progressPercentage досягає 100% і квест ще перебуває у стані active, статус автоматично змінюється на completed. Завершальним кроком викликається допоміжний метод _checkNearCompletion().

Допоміжний метод виявлення наближення до мети (_checkNearCompletion()). Метод реалізує психологічний ефект градієнта мети (Goal Gradient Effect): при досягненні 80% прогресу система надсилає сигнал UI для зміни візуальних акцентів – кольору, анімації або інших індикаторів [6]. Чеік-Мусса та колеги підтверджують, що візуальне підкріплення при наближенні до мети суттєво підвищує ймовірність її досягнення [6]. Метод є приватним (_), що підкреслює його допоміжну роль і унеможливлює виклик ззовні класу відповідно до принципу інкапсуляції [34].

Метод отримання нагороди (claimReward()). Метод захищає систему від повторного нарахування винагороди – критична вимога для балансу віртуальної економіки [50]. Нагорода може бути отримана лише у стані completed – будь-яка інша спроба повертає false без змін стану. При успішному виклику статус переводиться у claimed, що остаточно блокує повторне отримання тієї самої нагороди. Верберг та Хантер визначають захист від подвійного нарахування як фундаментальну вимогу до будь-якої гейміфікованої системи нагород [50].

Похідний атрибут перевірки терміну дії (isExpired). Getter реалізує дворівневу логіку: якщо expiryDate є null, квест вважається безстроковим і повертає false. В іншому випадку порівнюється поточний час із датою завершення через isAfter() [17]. Це дозволяє системі автоматично видаляти прострочені квести з локального кешу навіть в офлайн-режимі відповідно до вимог підрозділу 1.4 [30]. Практичну реалізацію описаних методів наведено у рисунку 2.12.

```
void updateProgress(double value) {
    progressPercentage = value.clamp(0.0, 100.0);
    if (progressPercentage >= 100.0 && status == QuestStatus.active) {
      status = QuestStatus.completed;
      print(' Квест $questId завершено!');
    }
    _checkNearCompletion();
  }

  void _checkNearCompletion() {
    if (progressPercentage >= 80.0 && status == QuestStatus.active) {
      print(' Квест $questId майже завершено ($progressPercentage%)! Змінюємо візуальні акценти.');
    }
  }

  bool claimReward() {
    if (status != QuestStatus.completed) {
      print(' Квест $questId ще не завершено або нагороду вже отримано.');
      return false;
    }
    status = QuestStatus.claimed;
    print('Нагороду за квест $questId отримано!');
    return true;
  }

  bool get isExpired {
    if (expiryDate == null) return false;
    return DateTime.now().isAfter(expiryDate!);
  }

  @override
  String toString() =>
      'Quest(id: $questId, progress: $progressPercentage%, status: ${status.name}, expired: $isExpired)';
```

Рисунок 2.12 – Методи класу Quest: логіка прогресу, нагород та терміну дії

Клас управління досягненнями та квестами (AchievementsAndQuests). Клас є агрегатором, що об'єднує дві колекції: achievementsUnlocked – типізований список розблокованих досягнень (List<Achievement>), та activeQuests – список активних квестів (List<Quest>). Обидві колекції ініціалізуються порожніми масивами в initializer list конструктора, що гарантує їх існування з моменту створення об'єкта [17]. Метод unlockAchievement() перевіряє наявність досягнення у колекції через any() з лямбда-функцією порівняння ідентифікаторів перед додаванням, запобігаючи дублюванню [34]. Метод addQuest() реалізує аналогічну логіку захисту від дублікатів через перевірку унікальності questId, що є критичним для коректності нарахування нагород [50]. Метод removeExpiredQuests() використовує removeWhere() з посиланням на getter isExpired, забезпечуючи автоматичне очищення локального кешу від неактуальних завдань навіть в офлайн-режимі [30]. Метод findQuest() повертає квест за ідентифікатором або null, використовуючи try-catch для сумісності з базовою бібліотекою Dart без додаткових залежностей [17]. Практичну реалізацію описаного класу наведено у рисунку 2.13.

```dart
class AchievementsAndQuests {
  final List<Achievement> achievementsUnlocked;
  final List<Quest> activeQuests;

  AchievementsAndQuests()
      : achievementsUnlocked = [],
        activeQuests = [];


  void unlockAchievement(String achievementId) {
    final alreadyUnlocked =
        achievementsUnlocked.any((a) => a.achievementId == achievementId);
    if (alreadyUnlocked) {
      print(' Досягнення $achievementId вже розблоковано.');
      return;
    }
    achievementsUnlocked.add(Achievement(achievementId: achievementId));
    print(' Досягнення $achievementId розблоковано!');
  }


  void addQuest(Quest quest) {
    final exists = activeQuests.any((q) => q.questId == quest.questId);
    if (exists) {
      print(' Квест ${quest.questId} вже існує.');
      return;
    }
    activeQuests.add(quest);
    print(' Квест ${quest.questId} додано.');
  }

  void removeExpiredQuests() {
    final before = activeQuests.length;
    activeQuests.removeWhere((q) => q.isExpired);
    final removed = before - activeQuests.length;
    if (removed > 0) print(' Видалено $removed прострочених квестів.');
  }

  Quest? findQuest(String questId) {
    try {
      return activeQuests.firstWhere((q) => q.questId == questId);
    } catch (_) {
      return null;
    }
  }

  @override
  String toString() {
    return 'AchievementsAndQuests('
        'achievements: ${achievementsUnlocked.length}, '
        'quests: ${activeQuests.length})';
  }
}
```

Рисунок 2.13 – Клас AchievementsAndQuests: управління досягненнями та квестами

2.3 Розробка діаграми станів для опису переходів між екранами та ігровими статусами.

Діаграма станів є формальним інструментом моделювання динамічної поведінки системи, що описує множину можливих станів об'єкта та умови переходів між ними [21]. У контексті гейміфікованої мобільної книгарні «SmartBook» діаграма станів виконує подвійну функцію: з одного боку, вона моделює навігаційні переходи між екранами застосунку, з іншого – формалізує зміни ігрових статусів користувача, що виникають як реакція на його поведінку. Фаулер визначає діаграму станів як найбільш придатний UML-артефакт для опису систем із реактивною поведінкою, де стан об'єкта безпосередньо визначає доступний функціонал інтерфейсу [21]. Саммервілль підкреслює, що формалізація станів системи є обов'язковим кроком перед початком реалізації: без чіткого визначення допустимих станів та умов переходів між ними виникає ризик некоректної поведінки у граничних випадках [47]. Мартін доповнює цю думку, стверджуючи, що скінченний автомат є найефективнішим патерном для управління складною логікою переходів у мобільних застосунках [34].

Діаграма станів навігаційних переходів. Навігаційна архітектура застосунку «SmartBook» побудована за принципом стану-як-джерела-істини (State as a Single Source of Truth), де поточний екран користувача є прямим відображенням його ігрового контексту [34]. Система розрізняє п'ять базових навігаційних станів: Гостьовий режим, Головний екран, Екран читання, Екран профілю та Екран квестів і нагород. Фаулер підкреслює, що чітке визначення навігаційних станів є фундаментом для побудови передбачуваної та тестованої архітектури мобільного застосунку [21].

Початковим станом є Гостьовий режим, в якому застосунок функціонує з обмеженим функціоналом без збереження ігрового прогресу. Тригером переходу до стану Головного екрану є успішна автентифікація користувача, що одночасно ініціює завантаження його JSON-профілю з хмарного сховища та синхронізацію

локального кешу [19, 30]. Цей перехід є необоротним у межах сесії: система не повертається до Гостьового режиму без явного виходу з облікового запису, що відповідає принципу захисту ігрового прогресу користувача [47].

З Головного екрану користувач може перейти до двох ключових доменів: Екрану читання та Екрану профілю. Перехід до Екрану читання відбувається при відкритті книги та є найбільш критичним з точки зору ігрової логіки, оскільки саме цей стан активує фоновий процес трекінгу прогресу [18]. Перебування у стані Екрану читання супроводжується безперервним моніторингом активності: кожна зміна сторінки є внутрішньою подією, що оновлює лічильники прогресу без зміни навігаційного стану відповідно до архітектурного принципу розділення навігаційного та ігрового стану [34]. Нільсен підкреслює, що внутрішні зміни стану не повинні переривати поточний контекст взаємодії користувача: фонові процеси мають бути невидимими для користувача та не впливати на плавність його досвіду [41]. Вихід з Екрану читання є тригером для асинхронного збереження накопиченого за сесію досвіду (XP) та перевірки умов виконання активних квестів [19, 47]. Еял визначає цей момент як критичну точку петлі залучення: система має негайно підтвердити результат сесії, щоб замкнути цикл дія-нагорода [16].

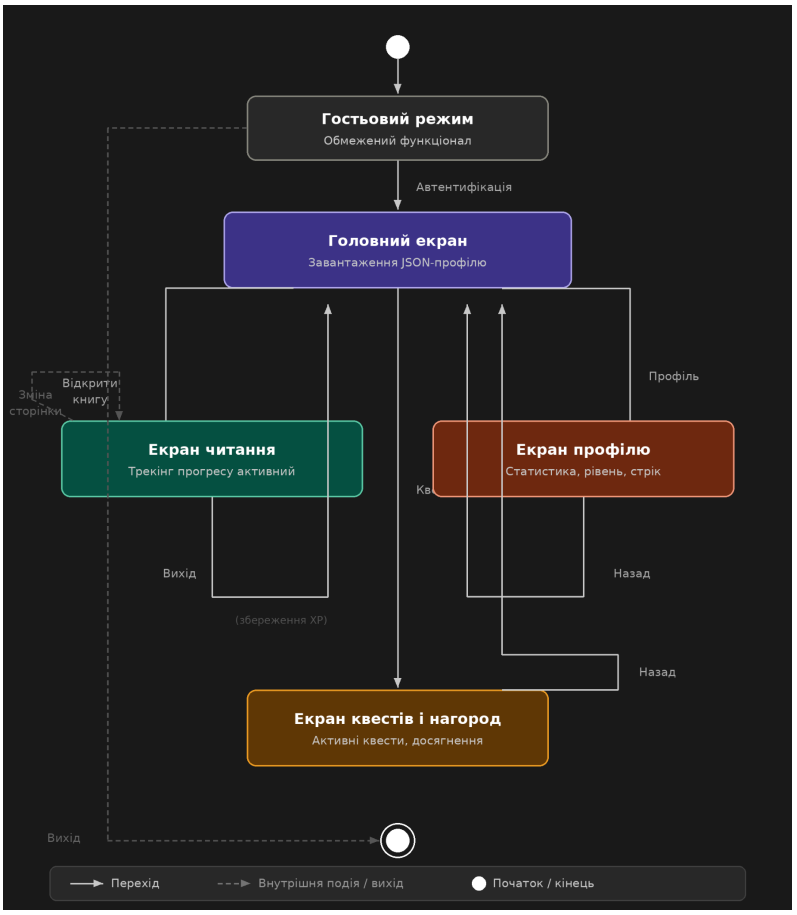


Рисунок 2.14 – Діаграма станів навігаційних переходів застосунку «SmartBook»

Діаграма станів ігрового профілю користувача. Ігровий профіль користувача є центральним об'єктом системи, стан якого змінюється незалежно від навігаційного контексту [15]. Моделювання його поведінки через діаграму станів дозволяє формалізувати правила нарахування винагород та запобігти виникненню некоректних станів, наприклад повторного отримання вже claimed нагороди [34]. Фаулер підкреслює, що моделювання складних об'єктів через діаграму станів є найефективнішим способом виявлення граничних випадків ще на етапі проєктування, до початку реалізації [21].

Профіль користувача може перебувати в одному з чотирьох агрегованих станів: Новий користувач, Активний читач, Користувач із досягненням рівня та Неактивний користувач. Стан Нового користувача характеризується нульовими значеннями всіх ігрових метрик: total_xp = 0, current_level = 1, daily_streak = 0. Перехід до стану Активного читача відбувається при першому нарахуванні

досвіду, що є тригером для ініціалізації last_activity_date [17]. Чоу визначає цей перший перехід як критичний момент онбордингу: система має забезпечити миттєву та видиму нагороду за першу дію, щоб сформувати позитивне очікування від подальшої взаємодії [7].

Всередині стану Активного читача система безперервно обробляє події нарахування XP та перевіряє умову підвищення рівня через геометричну прогресію з коефіцієнтом 1.1 [6]. При виконанні умови total_xp >= threshold(current_level + 1) відбувається перехід до проміжного стану Підвищення рівня, що супроводжується анімацією «Level Up» з часом відгуку до 200 мс відповідно до вимог підрозділу 1.4 [41], після чого система автоматично повертається до стану Активного читача з оновленим значенням current_level. Саффер визначає анімацію підвищення рівня як «момент тріумфу» – мікроінтеракцію, що має бути достатньо помітною, щоб сформувати емоційний відгук, але не настільки тривалою, щоб перервати потік читання [45].

Стан Неактивного користувача настає при перевищенні добового інтервалу між сесіями, що автоматично скидає daily_streak до нуля відповідно до логіки методу updateActivity(), описаного у підрозділі 2.2 [17]. Цей стан є оборотним: відновлення активності повертає профіль до стану Активного читача, але накопичений стрік не відновлюється, що підсилює психологічний ефект уникнення втрат (Core Drive 8 за Octalysis) [7]. Аріелі підтверджує цей механізм через теорію неприйняття втрат: страх втратити накопичений результат є в середньому вдвічі сильнішим мотиватором, ніж бажання отримати еквівалентну нову винагороду [1]. Каньман доповнює це спостереженням, що люди приймають рішення асиметрично відносно можливих втрат і здобутків, що робить механіку стріку особливо потужним інструментом утримання [28].

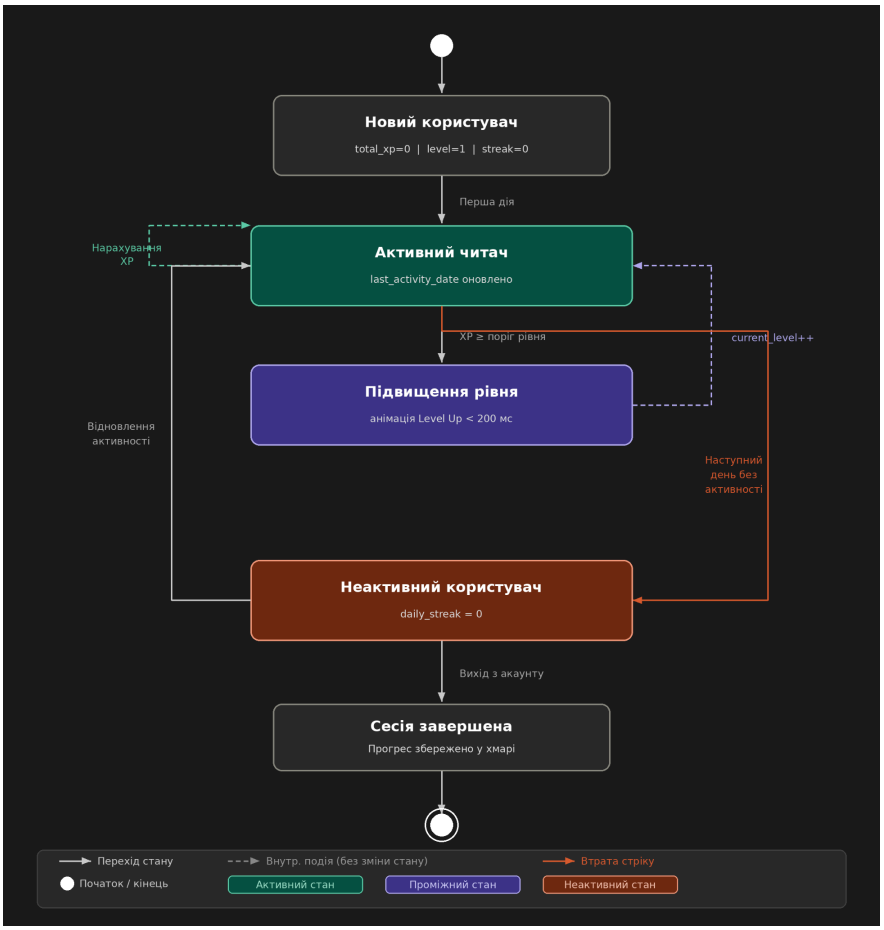


Рисунок 2.15 – Діаграма станів ігрового профілю користувача

Діаграма станів життєвого циклу квесту. Найбільш деталізованою є діаграма станів окремого квесту, оскільки саме вона формалізує правила, що запобігають критичним помилкам системи нагород – зокрема, повторному нарахуванню XP або валюти за вже виконане завдання [50]. Верберг та Хантер визначають захист від подвійного нарахування як фундаментальну вимогу до будь-якої гейміфікованої системи: порушення цілісності нагород руйнує довіру користувача до справедливості системи [50]. Мартін підкреслює, що скінченний автомат є найефективнішим патерном для реалізації подібних захисних механізмів: явне визначення термінальних станів унеможливлює виникнення некоректних переходів на рівні архітектури [34].

Життєвий цикл квесту охоплює чотири стани, що безпосередньо відповідають значенням переліку QuestStatus: active, completed, claimed, а також термінальний стан expired. Початковим станом кожного квесту є active, до якого

об'єкт переходить одразу після додавання через метод addQuest(). У цьому стані система безперервно відстежує атрибут progressPercentage, що оновлюється методом updateProgress() [17]. При досягненні значення 80% спрацьовує внутрішній тригер _checkNearCompletion(), який не змінює стан квесту, але генерує сигнал для UI щодо зміни візуальних акцентів, реалізуючи ефект градієнта мети [6]. Чеік-Мусса та колеги підтверджують, що візуальне підкріплення при наближенні до 80% порогу суттєво підвищує ймовірність завершення завдання порівняно з системами без подібного сигналу [6].

Перехід зі стану active до completed відбувається виключно при досягненні progressPercentage == 100.0, що є необхідною, але не достатньою умовою для отримання нагороди [34]. Стан completed є проміжним: квест очікує явного підтвердження від користувача через виклик claimReward(). Фогг визначає цей проміжний крок як важливий елемент поведінкової моделі: явне підтвердження отримання нагороди підсилює відчуття заслуженості результату та формує позитивне емоційне підкріплення [20]. Лише після цього підтвердження квест переходить до термінального стану claimed, з якого жодних подальших переходів не передбачено. Саммервілль визначає подібні термінальні стани як архітектурну гарантію цілісності даних: система фізично не може здійснити перехід із термінального стану, що унеможливлює повторне нарахування нагороди навіть у разі програмних помилок [47].

Паралельно зі станом active, якщо встановлено атрибут expiryDate, система безперервно перевіряє умову DateTime.now().isAfter(expiryDate) [17]. При виконанні цієї умови квест переходить до альтернативного термінального стану expired незалежно від поточного рівня прогресу, після чого підлягає автоматичному видаленню з локального кешу методом removeExpiredQuests() [30]. Аріелі пояснює мотиваційну силу обмежених у часі квестів через концепцію «ефекту дедлайну»: наближення терміну завершення суттєво підвищує інтенсивність дій користувача, що є ключовим механізмом для стимулювання щоденної активності [1].

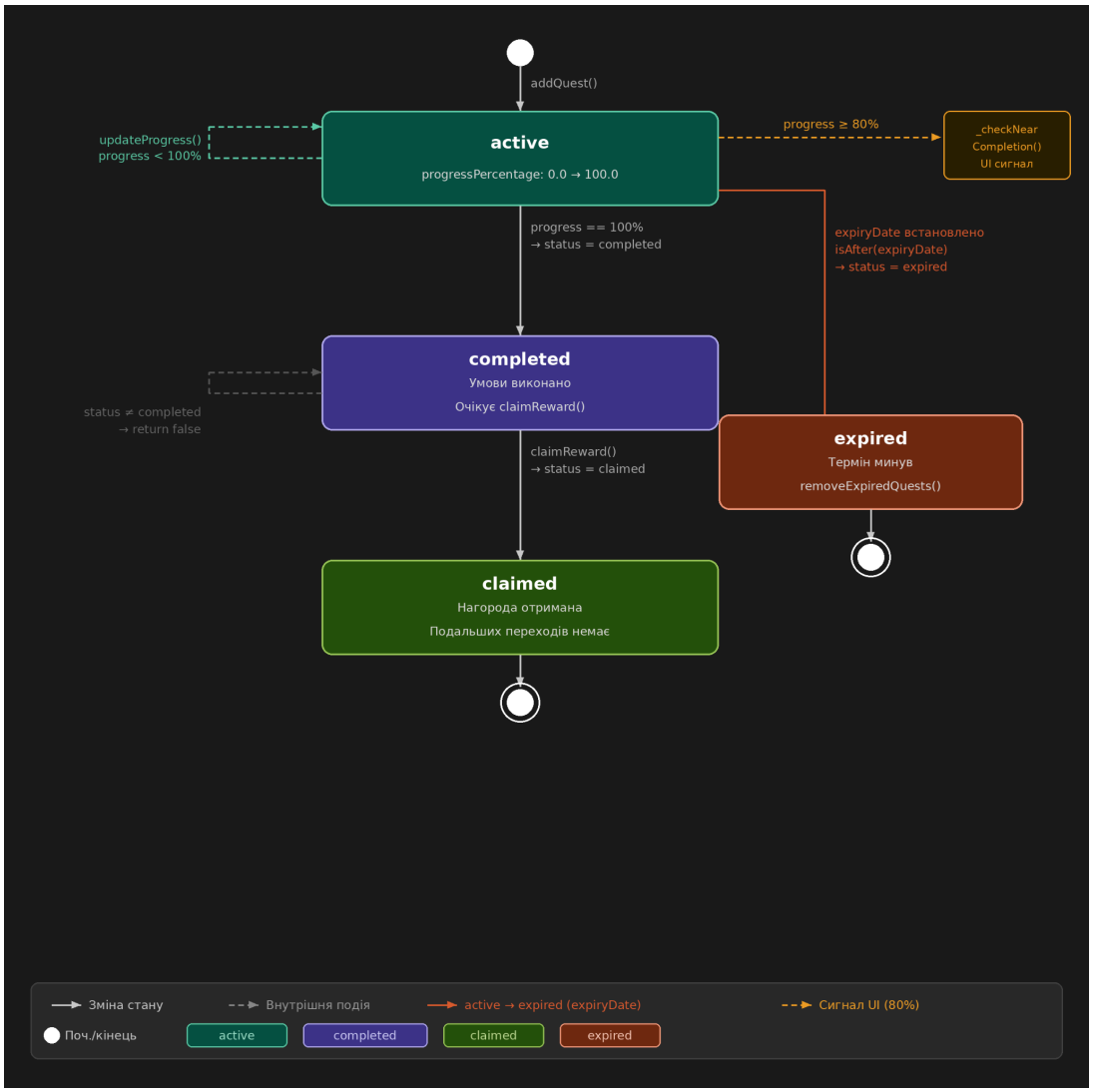


Рисунок 2.16 – Діаграма станів життєвого циклу квесту

Формалізація поведінки системи через три взаємопов'язані діаграми станів забезпечує повноту специфікації відповідно до методології Фаулера [21]: навігаційна діаграма визначає контекст взаємодії, діаграма профілю описує довгострокову еволюцію ігрових метрик, а діаграма квесту гарантує цілісність механіки нагород. Саммервілль підкреслює, що повнота специфікації є критичною умовою якості програмного продукту: кожен аспект поведінки системи має бути формально описаний до початку реалізації, щоб уникнути архітектурних рішень, прийнятих ad hoc у процесі розробки [47]. Вігерс та Біті доповнюють цю думку, стверджуючи, що взаємопов'язані моделі специфікації – навігаційна, поведінкова та доменна – є найефективнішим способом комунікації між аналітиком та розробником [51]. Разом три діаграми утворюють

формальний фундамент для подальшої розробки алгоритмічної бази гейміфікованих модулів, що є предметом наступного розділу дипломної роботи [34, 47].

2.4 Обґрунтування вибору платформи та інструментарію для реалізації інтерфейсу

Вибір технологічного стеку для реалізації гейміфікованого інтерфейсу мобільної книгарні «SmartBook» є стратегічним архітектурним рішенням, що безпосередньо визначає можливості системи щодо продуктивності анімацій, реактивності UI та масштабованості ігрових механік [40]. Саме цим рішенням обумовлено використання мови програмування Dart у підрозділах 2.2–2.3, де практична реалізація класів UserProgress, VirtualEconomy та Quest виконана у середовищі Flutter. Обґрунтування цього вибору базується на порівняльному аналізі існуючих платформ з урахуванням вимог технічного завдання, сформульованих у підрозділі 1.4, зокрема вимоги щодо часу відгуку анімацій до 200 мс, модульності архітектури та підтримки офлайн-режиму з хмарною синхронізацією. Навроцький та колеги у порівняльному дослідженні кросплатформних фреймворків підкреслюють, що вибір платформи є одним із найважливіших рішень у мобільній розробці: неправильний вибір на ранньому етапі призводить до суттєвих архітектурних обмежень, які неможливо усунути без повного переписування застосунку [40]. Вігерс та Біті доповнюють цю думку, зазначаючи, що технологічний стек має обиратися на основі чітко сформульованих нефункціональних вимог, а не особистих уподобань розробника [51].

Порівняльний аналіз мобільних платформ розробки. Сучасний ринок інструментів мобільної розробки пропонує три архітектурно відмінні підходи: нативна розробка для кожної платформи окремо (Swift/Kotlin), кросплатформна розробка на основі JavaScript-мосту (React Native) та кросплатформна розробка з власним рушієм рендерингу (Flutter). Вибір між цими підходами для системи з інтенсивними ігровими анімаціями потребує детального обґрунтування,

оскільки архітектурні рішення на рівні платформи безпосередньо визначають досяжність продуктивнісних вимог [40]. Навроцький та колеги у своєму порівняльному дослідженні встановили, що продуктивність рендерингу є ключовим критерієм вибору платформи для застосунків з інтенсивною анімацією та частими оновленнями UI [40].

Нативна розробка (Swift для iOS, Kotlin для Android) забезпечує максимальну продуктивність та прямий доступ до системних API. З точки зору вимог ТЗ щодо часу відгуку анімацій до 200 мс нативний підхід є технічно оптимальним, проте економічно недоцільним для академічного прототипу, оскільки вимагає підтримки двох незалежних кодових баз без пропорційного приросту якості кінцевого продукту та суперечить вимозі модульності підрозділу 1.4 [47]. Саммервілль визначає дублювання кодової бази як системне джерело технічного боргу, що суперечить принципам сучасної програмної інженерії та суттєво ускладнює подальше масштабування системи [47]. Мартін доповнює цю думку принципом DRY (Don't Repeat Yourself): будь-яке дублювання логіки є потенційним джерелом розсинхронізації між платформами при внесенні змін [34].

React Native використовує JavaScript-міст для комунікації між логікою застосунку та нативними компонентами UI. Ця архітектура створює системне вузьке місце для задач із високою частотою оновлення інтерфейсу, зокрема для анімацій прогрес-барів та переходів між екранами [40]. Нільсен зазначає, що затримки відгуку понад 100-200 мс руйнують відчуття безперервності взаємодії та негативно впливають на залученість користувача [41], що унеможливлює застосування React Native як основної платформи для реалізації гейміфікованих модулів. Саффер підкреслює, що мікроінтеракції – основний інструмент гейміфікованого інтерфейсу – потребують абсолютно плавного відтворення: будь-яка затримка руйнує емоційний ефект нагороди [45].

Flutter, розроблений компанією Google, використовує власний рушій рендерингу Skia та його наступник Impeller, що повністю обходить нативний UI-шар платформи [18]. Це забезпечує стабільні 60/120 кадрів на секунду на більшості сучасних пристроїв та детерміновану поведінку анімацій незалежно від версії операційної системи. Мова програмування Dart компілюється безпосередньо у нативний машинний код (AOT-компіляція), що усуває накладні витрати інтерпретатора та забезпечує передбачувану продуктивність у сценаріях реального часу [17]. Саме ці властивості Dart обумовили його використання для реалізації класів системи прогресу у підрозділі 2.2: сувора статична типізація мови дозволяє компілятору виявляти некоректні стани на етапі збірки, що є критичним для запобігання помилкам нарахування XP або ігрової валюти [17, 34]. Навроцький та колеги підтверджують, що Flutter демонструє найкращі показники продуктивності серед кросплатформних фреймворків у тестах із інтенсивним рендерингом анімацій [40].

КРИТЕРІЙ	SWIFT/KOTLIN	REACT NATIVE	FLUTTER/DART
ПРОДУКТИВНІСТЬ АНІМАЦІЙ	Висока	Середня	Висока
ЄДИНА КОДОВА БАЗА	Ні	Так	Так
ЧАС ВІДГУКУ UI	< 16 мс	50–200+ мс	50–200+ мс
СТАТИЧНА ТИПІЗАЦІЯ	Так	Ні	Так
ВІДПОВІДНІСТЬ ВИМОГАМ ТЗ	Частково	Ні	Так

Таблиця 2.1 – Порівняльна характеристика платформ мобільної розробки

На підставі цього аналізу для реалізації інтерфейсу застосунку «SmartBook» обрано Flutter/Dart як єдину платформу, що одночасно задовольняє вимогу щодо часу відгуку анімацій, забезпечує кросплатформність у межах

єдиної кодової бази та підтримує модульну архітектуру відповідно до підрозділу 1.4 [17].

Відповідність Flutter вимогам технічного завдання. Flutter відповідає вимогам ТЗ за трьома ключовими критеріями. По-перше, реактивна модель віджетів з механізмом перебудови лише змінених піддерев забезпечує ефективне оновлення UI-компонентів при зміні ігрових станів – зокрема, при оновленні прогрес-бару або зміні кольору при досягненні 80% цілі, як того вимагає підрозділ 1.4.2 [18]. По-друге, вбудована бібліотека анімацій (AnimationController, Tween, AnimatedWidget) надає декларативний інтерфейс для реалізації переходів з гарантованим часом відгуку в межах одного кадру (менше 16.7 мс при 60 fps), що значно перевищує вимогу ТЗ у 200 мс та забезпечує реалізацію анімації «Level Up», описаної у діаграмі послідовності підрозділу 2.1 [29]. По-третє, архітектура Flutter підтримує паралельний рендеринг у окремому потоці, що унеможливлює блокування анімацій під час виконання бізнес-логіки нарахування XP – тобто під час роботи методів addExperience() та _checkLevelUp(), описаних у підрозділі 2.2.1 [18, 34].

Окремо слід зазначити відповідність Flutter вимозі інклюзивності підрозділу 1.4.2. Згідно зі стандартом WCAG 2.1 рівня AA [49], інтерфейс має забезпечувати підтримку засобів доступності для користувачів із обмеженими можливостями. Flutter нативно підтримує семантичне дерево доступності, що забезпечує сумісність із екранними дикторами TalkBack (Android) [35] та VoiceOver (iOS) [26] без додаткового налаштування, що відповідає принципам універсального дизайну [46]. Навроцький та колеги підтверджують, що Flutter є єдиним кросплатформним фреймворком, що забезпечує нативну підтримку семантичного дерева доступності без використання сторонніх бібліотек [40].

Вибір архітектурного патерну та допоміжного інструментарію. Для управління станом застосунку обрано архітектурний патерн BLoC (Business Logic Component), що реалізує чітке розділення між рівнями представлення (UI), бізнес-логіки (ігрові механіки) та даних (локальний кеш, хмарне сховище)

[4]. Принцип розділення відповідальності, покладений в основу BLoC, є фундаментальним у сучасній програмній інженерії та безпосередньо відповідає вимозі модульності підрозділу 1.4.4 [47]. Мартін визначає подібне архітектурне розділення як необхідну умову для створення тестованих та масштабованих систем: компоненти, що не мають прямих залежностей між рівнями, можуть змінюватись незалежно один від одного [34]. Логіка нарахування XP, перевірки умов квестів та розрахунку рівнів, реалізована у класах підрозділу 2.2, інкапсулюється у відповідних BLoC-компонентах і не залежить від конкретного стану UI, що дозволяє впроваджувати нові ігрові механіки як окремі модулі без зміни основної логіки навігації, описаної у діаграмі станів підрозділу 2.3.1 [4, 34].

Для локального зберігання ігрового прогресу в офлайн-режимі, передбаченого підрозділом 1.4.3, обрано SQLite через бібліотеку sqflite. Реляційна модель SQLite забезпечує транзакційну цілісність операцій зі збереженням стану квестів, балансу віртуальної валюти та часових міток, необхідних для коректної роботи методу resolveSync(), описаного у підрозділі 2.2.2 [30]. Крайбіх підкреслює, що транзакційна модель SQLite є оптимальним рішенням для мобільних застосунків із вимогами до цілісності структурованих даних в офлайн-режимі [30]. Для хмарної синхронізації, передбаченої підрозділом 1.4.3, обрано Firebase Firestore як реалізацію хмарного сховища: ця документо-орієнтована база даних реального часу нативно підтримує JSON-структуру профілю користувача, описану у підрозділі 2.2, та забезпечує вирішення конфліктів синхронізації на основі часових міток відповідно до логіки методу resolveSync() [19].

Для проєктування UI-компонентів відповідно до вимог підрозділу 1.4.2 використано Figma як інструмент прототипування високої точності (High-fidelity prototype). Figma дозволяє специфікувати параметри анімацій, контрастність елементів згідно зі стандартом WCAG 2.1 [49] та адаптивність віджетів у діапазоні від 320dp до 600dp+, передбаченому підрозділом 1.4.4.

Ґотелф та Сейден підкреслюють, що застосування інструментів прототипування на етапі проєктування скорочує кількість архітектурних помилок та забезпечує відповідність кінцевого продукту специфікаціям [23]. Інструментарій доповнюється бібліотекою flutter_animate для декларативного опису складних анімаційних послідовностей [29], що технічно забезпечує дотримання вимоги щодо часу відгуку до 200 мс, встановленої у підрозділі 1.4.2.

Таким чином, обраний технологічний стек – Flutter/Dart з архітектурою BLoC, SQLite для локального кешу та Firebase Firestore для хмарної синхронізації – повністю відповідає функціональним і нефункціональним вимогам ТЗ підрозділу 1.4 [47, 51]. Він логічно випливає з практичної реалізації класів підрозділів 2.2–2.3 та забезпечує технічну основу для розробки алгоритмічної бази гейміфікованих модулів, що є предметом наступного розділу дипломної роботи [34, 40].

РОЗДІЛ 3. РОЗРОБКА АЛГОРИТМІЧНОЇ БАЗИ ГЕЙМІФІКОВАНИХ МОДУЛІВ

3.1. Алгоритми нарахування винагород та управління рівнями користувача

Алгоритмічна база системи нарахування винагород є ключовим інженерним компонентом гейміфікованої книгарні «SmartBook», оскільки саме вона перетворює дані про активність користувача на структуровані ігрові події [53]. Проєктування цих алгоритмів базується на принципі «петель зворотного зв'язку», де кожна дія користувача породжує ланцюгову реакцію обчислень, що завершується миттєвою зміною стану UI [53]. Практична реалізація виконана мовою Dart у середовищі Flutter, як обґрунтовано у підрозділі 2.4, на основі класів UserProgress, VirtualEconomy та AchievementsAndQuests, описаних у підрозділі 2.2 [34]. Хамарі та колеги емпірично підтверджують, що ефективність гейміфікованої системи прямо залежить від якості алгоритмічної бази нарахування винагород: некоректне нарахування XP є найчастішою причиною відтоку користувачів із гейміфікованих платформ [24].

Загальна архітектура алгоритму обробки подій. Система обробки ігрових подій побудована за принципом послідовного конвеєра, де вхідна подія проходить чотири послідовні етапи: валідацію, обчислення, збереження та візуалізацію [34]. Така архітектура відповідає діаграмі послідовності підрозділу 2.1, де асинхронний фоновий процес відокремлений від локального відгуку UI [18]. Схематичне зображення конвеєра наведено на рисунку 3.1.

На першому етапі система перевіряє коректність вхідного параметра та відхиляє некоректні значення відповідно до принципу «fail fast» [47]. На другому етапі виконуються основні обчислення: додавання XP та перевірка умови підвищення рівня через геометричну прогресію, описану у підрозділі 2.2.1. На третьому етапі результат зберігається у локальному кеші SQLite з оновленням часової мітки відповідно до вимог підрозділу 1.4.3 [30]. На

четвертому етапі BLoC-компонент сповіщає UI про зміну стану у межах 200 мс відповідно до вимог ТЗ [4, 41].

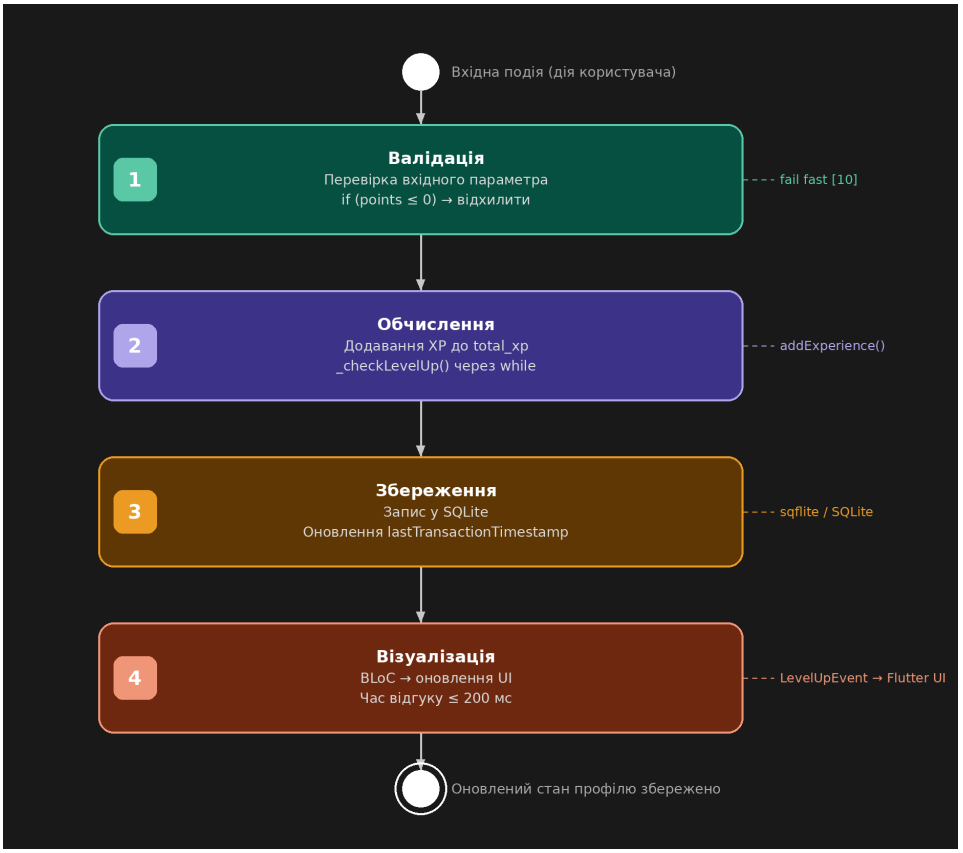


Рисунок 3.1 – Конвеєр обробки ігрової події

Алгоритм нарахування XP реалізує систему диференційованих ваг, де різні типи дій мають різну цінність у балах [7]. Ряан та Дечі підтверджують, що змінна система винагород формує стійкіші поведінкові паттерни, ніж рівномірне нарахування, оскільки підтримує внутрішню мотивацію через відчуття прогресу та компетентності [44].

ПОДІЯ	ВАГА (XP)	ОБҐРУНТУВАННЯ
ПРОЧИТАНА СТОРІНКА	2	Базова одиниця активності
ЗАВЕРШЕНИЙ РОЗДІЛ	15	Завершення проміжної цілі
ЗАВЕРШЕНА КНИГА	100	Досягнення головної цілі
ЩОДЕННИЙ ВХІД	10	Стимулювання retention
НАПИСАНИЙ ВІДГУК	25	Залучення до спільноти
НАПИСАНИЙ ВІДГУК	50	Виконання ігрового завдання

Таблиця 3.1 – Система ваг ігрових подій

Центральним елементом алгоритму є метод addExperience() у поєднанні з _checkLevelUp(). Критичною архітектурною деталлю є використання циклу while замість умовного оператора if для перевірки умови підвищення рівня, що забезпечує коректну обробку сценарію, коли користувач долає кілька порогових значень за одну транзакцію [34]. Алгоритм роботи методів наведено на рисунку 3.2.

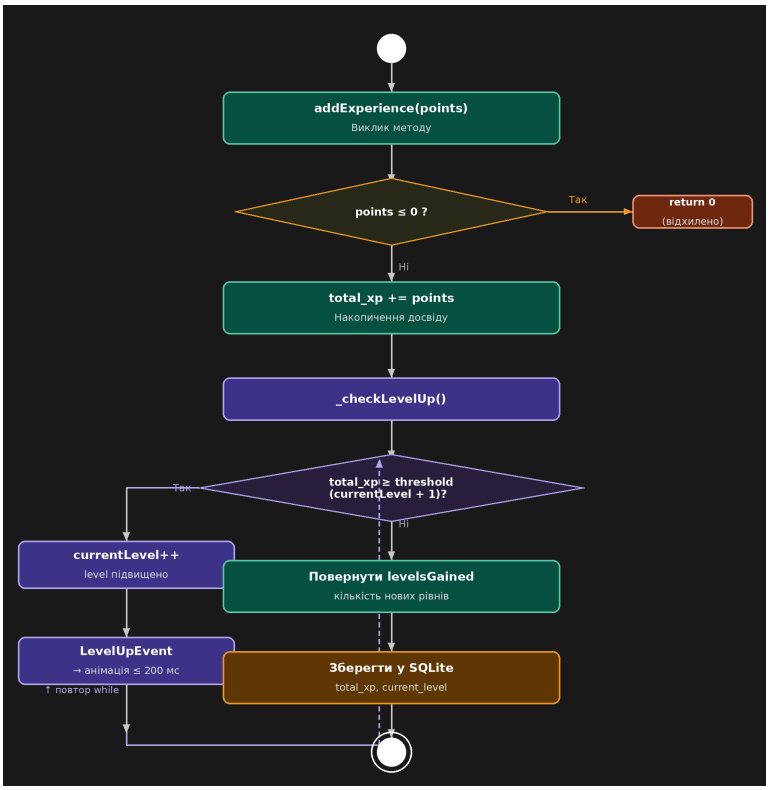


Рисунок 3.2 – Алгоритм методів addExperience() та _checkLevelUp()

Порогові значення рівнів розраховуються за формулою геометричної прогресії з коефіцієнтом 1.1: threshold(level) = 100 × 1.1^(level − 1). Нелінійне зростання порогів підтримує високий темп винагород на початкових рівнях відповідно до механіки Core Drive 2 фреймворку Octalysis [7] і водночас не демотивує досвідчених користувачів, оскільки кожен наступний рівень вимагає лише на 10% більше зусиль. Чеік-Мусса та колеги підтверджують, що нелінійні моделі прогресу з помірним коефіцієнтом зростання є оптимальними для підтримки мотивації на всіх етапах взаємодії з продуктом [6].

РІВЕНЬ	ПОРІГ XP	ПРИРІСТ ДО ПОПЕРЕДНЬОГО
1	0	–
2	100	+100
3	110	+10
5	133	+12
10	236	+21
20	612	+56

Таблиця 3.2 – Порогові значення XP для ключових рівнів

Похідні атрибути xp_to_next_level та levelProgressPercentage обчислюються динамічно як getter-методи без звернення до бази даних, що відповідає принципу єдиного джерела істини [34] та забезпечує актуальність даних прогрес-бару у будь-який момент сесії [18].

Алгоритм управління валютою та захист від повторного нарахування. Алгоритм управління віртуальною валютою реалізує асиметричну модель валідації: нарахування (earn()) є безпечною операцією, тоді як списання (spend()) потребує дворівневої перевірки – спочатку коректності суми, потім достатності балансу через canAfford() [34]. Повна логіка алгоритму з розгалуженнями наведена на рисунку 3.3.

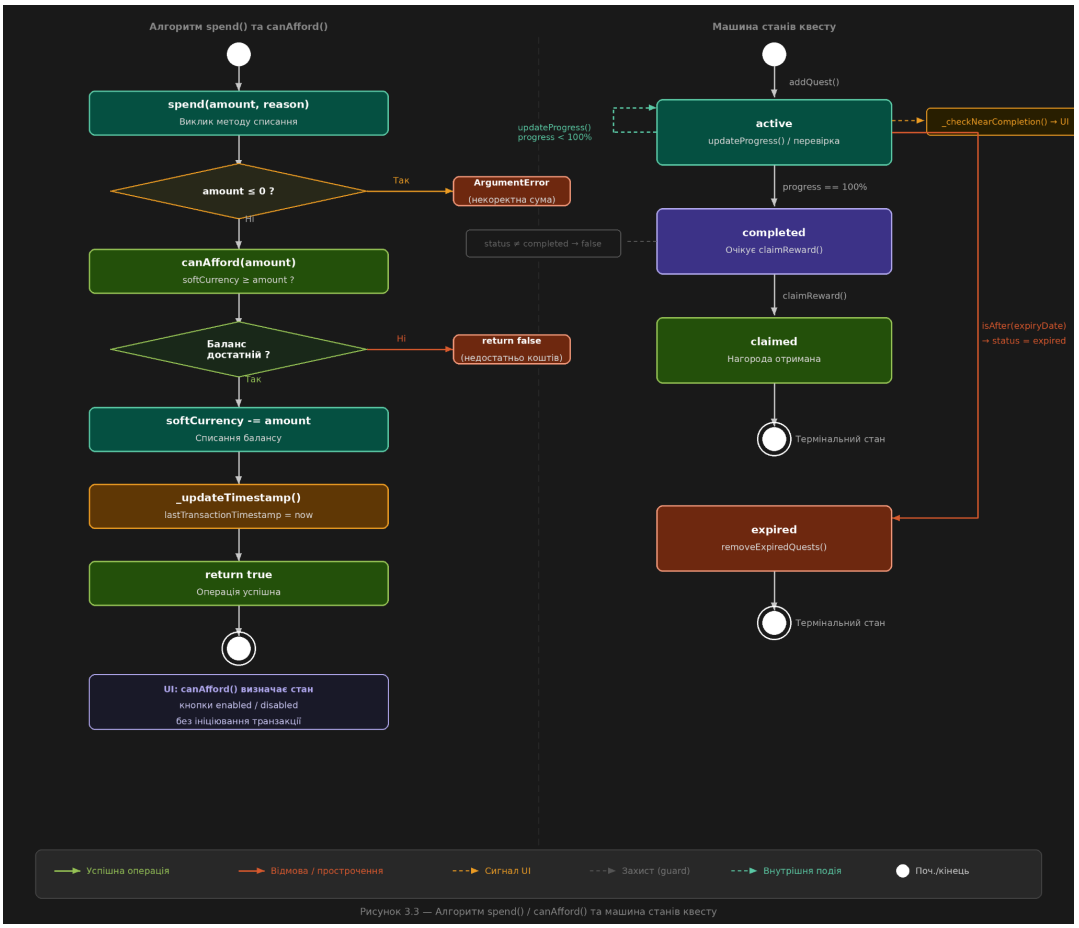


Рисунок 3.3 – Алгоритм методів spend(), canAfford() та машина станів квесту

Винесення перевірки балансу в окремий публічний метод canAfford() дозволяє Flutter-віджету кнопки покупки визначати свій стан (enabled/disabled) без ініціювання транзакції, що реалізує принцип захисного програмування [47]. Кожна успішна транзакція оновлює lastTransactionTimestamp, що забезпечує коректну роботу алгоритму синхронізації resolveSync() в офлайн-режимі, описаного у підрозділі 2.2 [30].

Алгоритм управління квестами вирішує проблему повторного отримання нагороди через тристанову машину станів (active → completed → claimed), також зображену на рисунку 3.3. Метод claimReward() повертає false для будь-якого квесту поза станом completed, а після переходу у термінальний стан claimed повернення неможливе – це архітектурна гарантія відповідно до діаграми станів підрозділу 2.3 [50]. Алгоритм очищення прострочених квестів removeExpiredQuests() виконується локально без доступу до мережі, що

відповідає вимозі офлайн-режиму підрозділу 1.4.3 [30]. Таким чином, описані алгоритми утворюють цілісну систему обробки ігрових подій, де кожен компонент реалізує принципи захисного програмування, математично обґрунтованої прогресії та архітектурної модульності [34, 47].

3.2 Проєктування системи тригерів та пуш-сповіщень на основі дій користувача

Система тригерів є архітектурним містком між алгоритмами нарахування винагород та зовнішніми каналами комунікації з користувачем. Якщо алгоритми підрозділу 3.1 обробляють події всередині застосунку, то система тригерів визначає, які з цих подій мають виходити за межі сесії читання у вигляді пуш-сповіщень, що повертають користувача до застосунку. Проєктування цієї системи базується на принципах поведінкової психології: своєчасне та релевантне сповіщення підсилює позитивне підкріплення та формує звичку щоденної взаємодії з продуктом [1].

Класифікація тригерів за типом події. Усі тригери системи поділяються на дві категорії: внутрішні, що виникають під час активної сесії користувача, та зовнішні, що генеруються системою автономно [20]. Фогг визначає тригер як необхідний компонент будь-якої цільової дії: без нього навіть мотивований користувач може не виконати бажану дію у потрібний момент [20]. Еял розрізняє зовнішні тригери, що надходять із середовища, та внутрішні, що формуються як умовний рефлекс у відповідь на звичні контексти [16].

Внутрішні тригери активуються синхронно з діями користувача в межах BLoC-архітектури без мережевого з'єднання [4]. До цієї категорії належать: підвищення рівня (LevelUpEvent), розблокування досягнення (AchievementUnlockedEvent), завершення квесту (QuestCompletedEvent) та досягнення 80% прогресу (NearCompletionEvent). Кожна подія ініціює негайну візуальну реакцію UI з часом відгуку до 200 мс відповідно до вимог ТЗ [41, 45].

Зовнішні тригери генеруються серверною частиною або локальним планувальником і доставляються через пуш-сповіщення навіть за умови закритого застосунку [19]. До цієї категорії належать: нагадування про щоденний стрік, сповіщення про скидання квестів, попередження про прострочення часового квесту та персоналізовані рекомендації. Чалдіні визначає своєчасність як ключову властивість ефективного тригера: сповіщення у момент найвищої схильності до дії є значно ефективнішим за аналогічне у невідповідний час [8].

ТРИГЕР	ТИП	ДЖЕРЕЛО	КАНАЛ ДОСТАВКИ
ПІДВИЩЕННЯ РІВНЯ	Внутрішній	_checkLevelUp()	UI анімація
РОЗБЛОКУВАННЯ ДОСЯГНЕННЯ	Внутрішній	unlockAchievement()	UI анімація + снекбар
ЗАВЕРШЕННЯ КВЕСТУ	Внутрішній	updateProgress()	UI анімація
ПРОГРЕС КВЕСТУ ≥ 80%	Внутрішній	_checkNearCompletion()	Зміна кольору UI
НАГАДУВАННЯ ПРО СТРІК	Зовнішній	Планувальник	Пуш-сповіщення
СКИДАННЯ ДОБОВИХ КВЕСТІВ	Зовнішній	Серверний таймер	Пуш-сповіщення
ПРОСТРОЧЕННЯ КВЕСТУ	Зовнішній	isAfter(expiryDate)	Пуш-сповіщення
ПЕРСОНАЛЬНА РЕКОМЕНДАЦІЯ	Зовнішній	Firebase	Пуш-сповіщення

Таблиця 3.3 – Класифікація тригерів системи

Архітектура внутрішніх тригерів. Внутрішні тригери реалізовані як потік подій у BLoC-архітектурі: кожен алгоритм підрозділу 3.1 при виконанні умови емітує відповідну подію у потік, на який підписані UI-компоненти [4]. Такий підхід забезпечує слабку зв'язаність між бізнес-логікою та інтерфейсом – алгоритм нарахування XP не знає про конкретний вигляд анімації, він лише сигналізує про факт події [34].

Найбільш критичним з точки зору UX є тригер підвищення рівня. Метод _checkLevelUp() емітує подію LevelUpEvent із новим значенням рівня, після чого UI-шар запускає анімаційну послідовність через flutter_animate [29]. Весь

цикл від емісії події до початку анімації має вкластись у 200 мс відповідно до вимог ТЗ – Flutter забезпечує цю вимогу завдяки окремому UI-потоку підрозділу 2.4, що унеможливлює затримку анімації через бізнес-логіку [18, 41].

Тригер _checkNearCompletion() не змінює стан квесту, але генерує сигнал для зміни візуальних акцентів UI – кольору прогрес-бару та відображення мотиваційного повідомлення [45]. Цей механізм реалізує ефект градієнта мети (Goal Gradient Effect), описаний у підрозділах 2.2 та 2.3: наближення до завершення задачі підвищує мотивацію користувача [6, 7].

Архітектура зовнішніх тригерів та пуш-сповіщень. Зовнішні тригери реалізуються через два незалежні канали. Перший – серверні тригери через Firebase Cloud Messaging (FCM) – для подій, що потребують серверної валідації: скидання добових квестів або персоналізованих рекомендацій [19]. Другий – локальний планувальник через flutter_local_notifications – для подій із детермінованим часом активації, наприклад нагадування про стрік без мережевого з'єднання [18].

Найбільш чутливим з точки зору утримання є тригер нагадування про щоденний стрік. Алгоритм базується на атрибуті last_activity_date підрозділу 2.2: якщо різниця між поточною датою та датою останньої активності дорівнює одному дню, система генерує нагадування до завершення поточного дня. Це рішення використовує механіку уникнення втрат Core Drive 8 фреймворку Octalysis [7]: користувач отримує сповіщення не про можливість отримати нагороду, а про ризик втрати накопиченого стріку, що є значно потужнішим психологічним стимулом [8, 28]. Тригер прострочення квесту активується на основі атрибута expiryDate класу Quest підрозділу 2.2 та надсилає попереджувальне сповіщення за дві години до завершення терміну [19]. Це рішення реалізує механіку дефіциту Core Drive 6 фреймворку Octalysis [7]: обмежений у часі ресурс сприймається як більш цінний, що стимулює негайну дію з боку користувача [1].

Принципи проєктування контенту сповіщень. Ефективність пуш-сповіщень визначається не лише технічною точністю тригера, а й якістю контенту повідомлення. Сповіщення з персоналізованим контентом мають відсоток відкриття у 4 рази вищий порівняно з генеричними [46]. У системі «SmartBook» контент формується динамічно на основі поточного стану профілю користувача відповідно до принципу персоналізації Core Drive 4 фреймворку Octalysis [7].

ТРИГЕР	ЗАГОЛОВОК	ТЕКСТ СПОВІЩЕННЯ
НАГАДУВАННЯ ПРО СТРІК	«Ваш стрік під загрозою»	«{N} днів поспіль — не зупиняйтесь сьогодні»
СКИДАННЯ КВЕСТІВ	«Нові завдання на сьогодні»	«Заробіть до {XP} очок за нові квести»
ПРОСТРОЧЕННЯ КВЕСТУ	«Квест завершується за 2 години»	«Залишилось {%} до виконання»
ПІДВИЩЕННЯ РІВНЯ	«Новий рівень!»	«Вітаємо — ви досягли рівня {N}»

Таблиця 3.4 – Шаблони контенту пуш-сповіщень

Змінні у фігурних дужках підставляються динамічно зі стану профілю при формуванні сповіщення. FCM доставляє лише ідентифікатор шаблону та параметри підстановки, а фінальний текст формується на пристрої на основі локальних даних профілю, що мінімізує мережевий трафік [19].

Окремим аспектом є частота сповіщень: надмірна їх кількість призводить до ігнорування або відключення [20]. У системі «SmartBook» встановлено обмеження – не більше двох пуш-сповіщень на добу з інтервалом не менше чотирьох годин, що реалізується на рівні серверної логіки Firebase перед відправленням запиту до FCM [19].

Таким чином, спроєктована система тригерів та пуш-сповіщень забезпечує безперервний цикл залучення користувача як під час активної сесії через внутрішні тригери, так і поза нею через зовнішні канали [16]. Психологічна обґрунтованість кожного тригера через механіки фреймворку Octalysis [7] та технічна реалізація через FCM і локальний планувальник створюють систему, що відповідає як вимогам ТЗ, так і принципам етичного дизайну залучення [20, 43].

3.3 Технічні аспекти забезпечення доступності згідно зі стандартами WCAG

Забезпечення доступності інтерфейсу є не лише етичною вимогою універсального дизайну, а й технічною специфікацією, закладеною у підрозділі 1.4.2 [49]. Згідно з міжнародним стандартом WCAG 2.1 рівня AA, інтерфейс має бути сприйнятним, керованим, зрозумілим та надійним для користувачів із різними функціональними обмеженнями [49]. Шнайдерман підкреслює, що доступність має закладатись на етапі проєктування, а не додаватись як виправлення після реалізації [46]. У контексті гейміфікованої книгарні «SmartBook» це завдання є особливо складним: ігрові елементи – анімації, кольорові індикатори прогресу, динамічні значки досягнень – є потенційними бар'єрами для користувачів із порушеннями зору, слуху або моторики, якщо їх реалізація не враховує принципів доступності від початку проєктування [2, 46].

Принципи WCAG 2.1 у контексті гейміфікованого інтерфейсу. Стандарт WCAG 2.1 організований навколо чотирьох ключових принципів, кожен з яких має специфічні технічні імплікації для гейміфікованих компонентів системи «SmartBook» [49].

Принцип сприйнятності (Perceivable) вимагає, щоб уся інформація, що передається через колір або анімацію, мала альтернативне текстове або структурне представлення [49]. У системі «SmartBook» це стосується прогрес-барів, що змінюють колір при досягненні 80% цілі, та значків досягнень із колірним кодуванням статусу. Мінімальний коефіцієнт

контрастності між текстом і фоном має становити 4.5:1 для звичайного тексту та 3:1 для великого тексту відповідно до критерію 1.4.3 стандарту WCAG 2.1 [49].

Принцип керованості (Operable) вимагає, щоб усі інтерактивні елементи були доступні через альтернативні засоби введення без використання жестів [26, 35]. Нільсен встановив, що час відгуку до 100 мс сприймається як миттєвий, до 1000 мс – як прийнятний [41]. Цей принцип безпосередньо корелює з вимогою підрозділу 1.4.2 щодо часу відгуку до 200 мс та поширюється на доступність інтерактивних елементів для користувачів з обмеженою моторикою [26].

Принцип зрозумілості (Understandable) вимагає, щоб ігрова термінологія та стан системи були зрозумілими для всіх користувачів [49]. Числові значення XP, рівні та відсотки прогресу мають супроводжуватись контекстними поясненнями, а не лише числовими індикаторами. Міллер визначає сім одиниць інформації як межу короткочасної пам'яті: інтерфейс не має одночасно відображати більше семи гейміфікованих показників [38].

Принцип надійності (Robust) вимагає коректної інтерпретації інтерфейсу асистивними технологіями – TalkBack (Android) та VoiceOver (iOS) – незалежно від версії ОС [2]. Flutter нативно підтримує обидва диктори через семантичне дерево доступності, що є однією з переваг обраної платформи, обґрунтованих у підрозділі 2.4 [18].

Контрастність та кольорове кодування. Найбільш поширеною помилкою у гейміфікованих інтерфейсах є використання кольору як єдиного засобу передачі інформації про стан системи [46]. У системі «SmartBook» кольорові індикатори застосовуються у трьох критичних сценаріях: зміна кольору прогрес-бару при досягненні 80% цілі, колірне кодування статусу квесту (active, completed, claimed, expired) та візуальне виділення активного стріку.

Відповідно до стандарту WCAG 2.1, колір не може бути єдиним візуальним засобом передачі інформації [49]. Технічна реалізація у Flutter передбачає поєднання кольорового кодування з додатковими індикаторами: зміна форми або товщини рамки, додавання іконки стану або текстової мітки [18]. Прогрес-бар при досягненні 80% не лише змінює колір, а й збільшує товщину та додає пульсуючу анімацію – сприйнятний сигнал незалежно від здатності користувача розрізняти кольори [45, 49].

КОМПОНЕНТ	МІН. КОЕФІЦІЄНТ	РЕАЛІЗАЦІЯ У FLUTTER	КРИТЕРІЙ WCAG
ОСНОВНИЙ ТЕКСТ	4.5:1	ThemeData.textTheme	1.4.3 AA
ВЕЛИКИЙ ТЕКСТ (18PT+)	3:1	TextStyle.fontSize	1.4.3 AA
ІКОНКИ ДОСЯГНЕНЬ	3:1	Icon.color + семантика	1.4.11 AA
ПРОГРЕС-БАР	3:1	LinearProgressIndicator	1.4.11 AA
КНОПКИ ДІЙ	4.5:1	ElevatedButton.style	1.4.3 AA

Таблиця 3.5 – Вимоги до контрастності UI-компонентів

Перевірка відповідності коефіцієнтів контрастності здійснюється на етапі прототипування у Figma через вбудований інструмент аналізу контрастності, а також автоматично у процесі тестування через Flutter-бібліотеку семантичного аудиту [23].

Семантичне дерево доступності у Flutter. Flutter реалізує доступність через семантичне дерево – паралельну структуру даних, що описує значення кожного UI-елемента для асистивних технологій незалежно від його візуального представлення [18]. Кожен віджет Flutter може мати семантичні атрибути: label (текстовий опис), value (поточне значення), hint (інструкція до дії) та liveRegion (індикатор динамічного оновлення) [2].

У контексті гейміфікованих компонентів семантичне дерево потребує особливої уваги. Прогрес-бар, що відображає levelProgressPercentage, має бути анотований значенням у форматі «Прогрес рівня: 73 відсотки» замість

числового значення 0.73, оскільки екранний диктор зачитує саме семантичний атрибут value [26]. Значок досягнення має містити семантичний label з назвою та описом досягнення, а не лише tooltip для зрячих користувачів [35].

Динамічні зміни стану, зокрема поява сповіщення про підвищення рівня, реалізуються через семантичний атрибут liveRegion із значенням assertive, що змушує TalkBack та VoiceOver негайно оголосити нову інформацію [26, 35]. Рекомендації Apple та Google підкреслюють важливість своєчасного оголошення динамічних змін через liveRegion: незрячий користувач не може самостійно виявити появу нового елемента на екрані без відповідного сигналу від асистивної технології [26, 35].

Підтримка моторної доступності. Моторна доступність стосується користувачів із обмеженнями рухових функцій, які можуть не мати можливості виконувати складні жестові взаємодії [49]. Вимоги до мінімальних розмірів інтерактивних елементів різняться залежно від платформи: Apple визначає мінімальний розмір зони дотику у 44×44 логічних пікселі [26], Google – у 48×48dp [35], тоді як стандарт WCAG 2.1 рекомендує 44×44 CSS-пікселі [49]. У Flutter цей параметр забезпечується через властивість minimumSize компонента ButtonStyle або через обгортку GestureDetector із відповідними відступами, що задовольняє вимоги обох платформ одночасно [18].

У гейміфікованому інтерфейсі особливу увагу потребують дрібні інтерактивні елементи: значки досягнень у галереї, кнопки отримання нагород у списку квестів та елементи навігації між ігровими розділами. Кожен із цих елементів має мати зону дотику не меншу за 48×48dp незалежно від візуального розміру, що досягається через Padding або InkWell з відповідними параметрами splashRadius [35]. Окремим аспектом є підтримка режиму «Перемикач доступу» (Switch Access на Android [35], Switch Control на iOS [26]), що дозволяє керувати застосунком через зовнішні кнопки або адаптивні пристрої введення. Flutter автоматично підтримує цей режим для стандартних компонентів, проте

кастомні анімовані елементи гейміфікованого UI потребують явного визначення фокусного порядку через FocusTraversalGroup та FocusNode [18, 26].

Доступність анімацій та динамічного контенту. Анімації є центральним елементом гейміфікованого досвіду, проте для частини користувачів вони можуть спричиняти дискомфорт або бути недоступними [45]. Відповідно до критерію 2.3 стандарту WCAG 2.1 рівня AA, застосунок має реагувати на системне налаштування «Зменшити рух» (Reduce Motion) [49]. Apple та Google окремо підкреслюють необхідність поваги до цього налаштування як обов'язкову вимогу для публікації застосунку [26, 35].

У Flutter це реалізується через MediaQuery.of(context).disableAnimations. При значенні true система «SmartBook» замінює складні анімаційні переходи на миттєві зміни стану, зберігаючи при цьому всю функціональність та інформаційний зміст [18]. Наприклад, анімація «Level Up» замість повноекранного ефекту відображає статичне сповіщення з текстовим описом події. Це рішення реалізує принцип прогресивного покращення: базовий функціонал доступний усім, анімаційне збагачення – лише тим, хто може та бажає його сприймати [41].

АНІМАЦІЯ	СТАНДАРТНИЙ РЕЖИМ	РЕЖИМ REDUCE MOTION
LEVEL UP	Повноекранний ефект з частинками	Статичне сповіщення з текстом
ПРОГРЕС-БАР 80%	Пульсація + зміна кольору	Лише зміна кольору
РОЗБЛОКУВАННЯ ДОСЯГНЕННЯ	Анімований розворот значка	Миттєва поява з текстом
ПЕРЕХІД МІЖ ЕКРАНАМИ	Слайд-анімація	Fade або миттєвий перехід

Таблиця 3.6 – Стратегія адаптації анімацій

Таким чином, технічна реалізація доступності у системі «SmartBook» охоплює чотири рівні: контрастність та кольорове кодування відповідно до WCAG 2.1 [49], семантичне дерево для асистивних технологій відповідно до рекомендацій Apple та Google [26, 35], моторна доступність через відповідні розміри зон дотику та адаптація анімацій до системних налаштувань [41]. Комплексне впровадження цих заходів робить гейміфікований інтерфейс доступним для найширшої аудиторії, що є передумовою для практичної реалізації інтерактивного прототипу у наступному розділі [46].

3.4 Методика інтеграції ігрових елементів у стандартні UI-компоненти мобільного застосунку

Методика інтеграції ігрових елементів визначає конкретні інженерні підходи до вбудовування гейміфікованих компонентів у стандартну UI-архітектуру Flutter-застосунку без порушення її модульності та без деградації продуктивності [34]. Якщо попередні підрозділи розділу 3 описували алгоритми (3.1), тригери (3.2) та доступність (3.3) як окремі аспекти системи, то підрозділ 3.4 є синтезуючим: він описує, яким чином усі ці компоненти фізично інтегруються у конкретні Flutter-віджети, що відображаються на екрані користувача. Методика базується на принципі композиції, згідно з яким ігровий елемент не замінює стандартний UI-компонент, а обгортає його, розширюючи функціональність без зміни базової поведінки [22]. Фрімен та колеги визначають патерн декоратора як найбільш відповідний для подібних задач: він дозволяє динамічно додавати нову поведінку до об'єкта без зміни його класу та без впливу на інші об'єкти тієї самої ієрархії [22]. Мартін підкреслює, що композиція є більш гнучким механізмом розширення функціональності порівняно зі спадкуванням, оскільки не створює жорстких залежностей між класами [34].

Принцип декораторної інтеграції. Основним архітектурним принципом методики є патерн декоратора – підхід, що дозволяє динамічно додавати нову

поведінку до існуючого об'єкта без зміни його структури [22]. У контексті Flutter це реалізується через обгортання стандартних віджетів у кастомні компоненти, що додають ігровий шар поверх базової функціональності. Такий підхід відповідає принципу відкритості/закритості (Open/Closed Principle): стандартні компоненти закриті для модифікації, але відкриті для розширення [34].

Наприклад, стандартна кнопка придбання книги (ElevatedButton) обгортається у GamifiedButton, що додає логіку нарахування XP при натисканні, анімацію підтвердження дії та перевірку балансу валюти через canAfford(). При цьому базова поведінка кнопки – навігація до сторінки оплати – залишається незмінною [22, 34]. Декораторна інтеграція реалізується на трьох рівнях Flutter-архітектури [18]. На рівні віджетів стандартні компоненти обгортаються кастомними обгортками з ігровою логікою. На рівні BLoC додаються нові потоки подій для ігрових станів без зміни існуючої бізнес-логіки [4]. На рівні даних JSON-профіль користувача розширюється ігровими атрибутами, описаними у підрозділі 2.2, без зміни базової схеми даних книгарні [15].

РІВЕНЬ	СТАНДАРТНИЙ КОМПОНЕНТ	ІГРОВЕ РОЗШИРЕННЯ	ПРИНЦИП
ВІДЖЕТ	ElevatedButton	GamifiedButton	Декоратор [24]
ВІДЖЕТ	LinearProgressIndicator	GamifiedProgressBar	Декоратор [24]
ВІДЖЕТ	ListTile	QuestCard	Композиція
BLOC	Базовий стан	LevelUpEvent, QuestCompletedEvent	Розширення потоку
ДАНІ	JSON-профіль книгарні	Атрибути UserProgress	Розширення схеми

Таблиця 3.7 – Рівні декораторної інтеграції

Інтеграція прогрес-бару читання. Прогрес-бар читання є найбільш фундаментальним гейміфікованим елементом системи, оскільки він перетворює абстрактний процес читання на вимірюваний прогрес [6]. Стандартний Flutter-компонент LinearProgressIndicator розширюється до GamifiedProgressBar

шляхом додавання трьох шарів функціональності відповідно до принципу декораторної інтеграції підрозділу 3.4.1 [22].

Перший шар – динамічна зміна кольору – реалізується через AnimatedContainer із інтерполяцією кольору залежно від значення progressPercentage. При значенні до 60% індикатор відображається у нейтральному кольорі, від 60% до 80% – у попереджувальному акцентному кольорі, від 80% до 100% – у яскравому цільовому кольорі з пульсуючою анімацією [45]. Кольорові переходи реалізуються через ColorTween з тривалістю 300 мс, що забезпечує плавність без перевищення порогу 200 мс для початку реакції [41, 36]. Другий шар – XP-індикатор – відображає кількість очок досвіду, що будуть нараховані за завершення поточного розділу. Він розміщується поруч із прогрес-баром у вигляді малого значка та оновлюється синхронно з прогресом читання, реалізуючи психологічний механізм передбачуваної винагороди [1, 7]. Третій шар – семантична анотація для доступності – реалізується через Semantics-обгортку з атрибутами label, value та liveRegion, описаними у підрозділі 3.3 [2]. Це забезпечує коректне зачитування прогресу екранними дикторами TalkBack [35] та VoiceOver [26] відповідно до стандарту WCAG 2.1 [49].

Інтеграція системи рівнів у профіль користувача. Екран профілю є центральним хабом ігрового прогресу користувача, де відображаються всі ключові метрики: поточний рівень, загальний XP, денний стрік та баланс валюти [7]. Інтеграція цих елементів реалізується через композицію спеціалізованих віджетів, кожен з яких відповідає за відображення конкретного атрибута класу UserProgress, описаного у підрозділі 2.2.1 [22].

Компонент LevelBadge відображає поточний рівень користувача у вигляді стилізованого значка. При підвищенні рівня компонент запускає анімацію через AnimationController тривалістю 600 мс, що складається з трьох фаз: розширення значка (200 мс), відображення нового рівня (200 мс) та повернення до базового

розміру (200 мс) [36]. Загальний час анімації узгоджений із порогами сприйняття Нільсена та не перевищує порогу помітного дискомфорту [41]. Компонент XpProgressBar відображає прогрес у межах поточного рівня на основі getter-методу levelProgressPercentage підрозділу 2.2 Компонент підписаний на потік BLoC-подій та автоматично перебудовується при кожній зміні total_xp без ручного керування станом [4]. Числові значення xp_to_next_level відображаються у форматі «залишилось N XP», що реалізує ефект градієнта мети, описаний у підрозділах 2.2 та 3.1 [6]. Компонент StreakCounter відображає поточний денний стрік на основі атрибута daily_streak. При значенні понад 7 днів компонент додає анімований ореол, що підсилює цінність тривалої серії та реалізує механіку уникнення втрат Core Drive 8 фреймворку Octalysis [7]. При значенні 0 компонент відображається у приглушеному стилі, що ненав'язливо сигналізує про відсутність активності без негативного підкріплення відповідно до принципів етичного дизайну [43].

Інтеграція віртуальної валюти у транзакційні екрани. Відображення балансу віртуальної валюти інтегрується у стандартний екран каталогу книг через постійний компонент CurrencyWidget у верхній панелі навігації. Компонент відображає поточне значення softCurrency та анімовано оновлюється при кожній транзакції: при нарахуванні числове значення «злітає» вгору з позитивним дельта-значенням (+50), при списанні – «опускається» вниз із від'ємним значенням (-80) [45]. Обидві анімації реалізуються через AnimatedFlipCounter тривалістю 400 мс, що відповідає порогам сприйняття анімацій [41, 36].

Інтеграція перевірки балансу у картки товарів реалізується через метод canAfford(), описаний у підрозділі 2.2: кнопка «Придбати за монети» автоматично переходить у стан disabled при недостатньому балансі та змінює текст на «Недостатньо монет» [34]. Це є проактивним інформуванням користувача без необхідності ініціювати транзакцію, що відповідає принципу

попередження помилок (Error Prevention) стандарту WCAG 2.1 [49] та принципам юзабіліті Нільсена [41].

КОМПОНЕНТ	БАЗОВИЙ ВІДЖЕТ	ІГРОВЕ РОЗШИРЕННЯ	АНІМАЦІЯ
GAMIFIEDPROGRESSBAR	LinearProgressIndicator	XP-індикатор, зміна кольору	ColorTween 300 мс
LEVELBADGE	Container	Рівень, анімація підвищення	AnimationController 600 мс
XPPROGRESSBAR	LinearProgressIndicator	xp_to_next_level	BLoC-потік
STREAKCOUNTER	Text	Анімований ореол	Пульсація
QUESTCARD	ListTile	Прогрес, ClaimButton	AnimatedSwitcher 400 мс
ACHIEVEMENTBADGE	GridTile	Flip-анімація, тактильний відгук	FlipCard 500 мс
CURRENCYWIDGET	Text	Дельта-анімація балансу	AnimatedFlipCounter 400 мс

Таблиця 3.8 – Зведена характеристика гейміфікованих компонентів

Таким чином, описана методика забезпечує системну інтеграцію всіх гейміфікованих елементів у стандартну UI-архітектуру Flutter через принцип декоратора [22], BLoC-потоки подій [4] та семантичні анотації доступності відповідно до рекомендацій Apple [26], Google [35] та стандарту WCAG 2.1 [49]. Кожен компонент є модульним та незалежним, що відповідає вимозі ТЗ щодо можливості додавання нових ігрових механік без рефакторингу існуючої кодової бази [34, 47].

РОЗДІЛ 4. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНТЕРФЕЙСНОГО ПРОТОТИПУ

4.1 Створення інтерактивного макета високої деталізації

У практиці проєктування інтерфейсів прийнято розрізняти три рівні деталізації прототипів: низький (Low-fidelity), середній (Mid-fidelity) та високий (High-fidelity). Вибір рівня деталізації є стратегічним рішенням, що визначає як обсяг проєктних робіт, так і цінність отриманого артефакту для подальшого тестування [51]. Для системи «SmartBook» обрано рівень High-fidelity, що обґрунтовується трьома факторами.

По-перше, гейміфіковані елементи інтерфейсу – анімовані прогрес-бари, колірне кодування статусів квестів, динамічні значки досягнень – є принципово візуальними конструктами, що не можуть бути адекватно представлені у спрощеному вигляді [7]. Low-fidelity прототип у вигляді дротяних схем не дозволив би перевірити відповідність кольорових рішень вимогам контрастності WCAG 2.1 [49], описаним у підрозділі 3.3, оскільки контрастність є атрибутом реального кольору, а не умовного позначення. По-друге, вимога підрозділу 1.4.2 щодо часу відгуку анімацій до 200 мс потребує верифікації на прототипі, максимально наближеному до кінцевого продукту. Figma дозволяє задавати тривалість анімаційних переходів між фреймами із точністю до мілісекунди, що робить High-fidelity прототип єдиним інструментом на доімплементаційному етапі, здатним підтвердити виконання цієї вимоги [23]. По-третє, High-fidelity прототип є необхідною умовою для проведення юзабіліті-тестування, запланованого у підрозділі 4.3. Нільсен демонструє, що користувачі значно точніше відтворюють реальну поведінку при взаємодії з деталізованим прототипом, оскільки він активує ті самі когнітивні та емоційні реакції, що й кінцевий продукт [41].

Інструментом прототипування обрано Figma – векторний редактор інтерфейсів із підтримкою компонентної системи, автолейауту та інтерактивних

переходів [23]. Компонентна система Figma дозволяє визначити кожен гейміфікований елемент – QuestCard, LevelBadge, GamifiedProgressBar – як окремий компонент із варіантами стану (active, completed, claimed), що відповідає архітектурі Flutter-віджетів підрозділу 3.4 [18]. Це забезпечує пряму відповідність між проєктним рішенням у Figma та його технічною реалізацією у коді [10].

Кольорова схема прототипу базується на темній темі з фіолетово-зеленою акцентною палітрою, що відповідає вимогам контрастності WCAG 2.1 рівня AA [49] та узгоджується з кольоровим кодуванням діаграм підрозділів 2.3 та 3.1 Типографічна система використовує шрифт з підтримкою кирилиці та чіткою ієрархією розмірів для забезпечення читабельності відповідно до підрозділу 3.3 [46].

Прототип охоплює п'ять ключових екранів: головний екран з каталогом книг, екран читання з гейміфікованим прогрес-баром, екран профілю користувача, екран квестів та досягнень, а також анімаційні переходи між ними. Кожен екран проєктується відповідно до навігаційної архітектури діаграми станів підрозділу 2.3 [21]. Детальний опис кожного екрана наведено у підрозділах 4.1–4.1 Додаткові екрани прототипу, що не є предметом детального аналізу в межах цієї роботи, наведено у Додатку А.

Головний екран є точкою входу у застосунок після автентифікації та першим контактом користувача з гейміфікованою системою. Відповідно до навігаційної діаграми станів підрозділу 2.3, він є центральним вузлом, з якого здійснюється перехід до всіх інших екранів. Макет розроблено у темній темі з фіолетовими акцентами, що відповідає вимогам контрастності WCAG 2.1 рівня AA [49], верифікованим у підрозділі 3.3.

Верхня панель інтегрує чотири гейміфіковані елементи відповідно до методики підрозділу 3.4. Аватар користувача з накладеним значком поточного рівня реалізує компонент LevelBadge, описаний у підрозділі 3.4: значок

відображає числовий рівень (12) та ранг («Книжковий майстер») і оновлюється при підвищенні рівня через анімацію AnimationController. Поруч із аватаром розміщено ім'я користувача та ранговий підзаголовок, що формують ідентифікаційний блок і реалізують механіку власності та статусу Core Drive 4 фреймворку Octalysis [7].

Лічильник стріку реалізує компонент StreakCounter на основі атрибута daily_streak класу UserProgress, описаного у підрозділі 2.2. При значенні понад 7 днів компонент відображається з іконкою полум'я та числовим значенням, що підсилює механіку уникнення втрат Core Drive 8 [7]: користувач бачить накопичений стрік у верхній частині кожного екрану і відчуває відповідальність за його збереження. Баланс ігрової валюти реалізує компонент CurrencyWidget з поточним значенням softCurrency класу VirtualEconomy, описаного у підрозділі 2.2, та анімовано оновлюється при кожній транзакції відповідно до підрозділу 3.4.

Шкала XP розміщена окремим рядком під іменем користувача та відображає прогрес поточного рівня у форматі «680 / 1000 XP» з градієнтним заповненням на основі getter-методу levelProgressPercentage, описаного у підрозділі 2.2. Поєднання текстового індикатора з візуальною шкалою реалізує подвійний формат відображення прогресу відповідно до методики підрозділу 3.4 та підсилює ефект градієнта мети [6].

Блок «Продовжити читання» реалізує пріоритетне відображення поточної книги з подвійним індикатором прогресу: абсолютним («120 з 208 сторінок») та відносним (57%). Такий підхід одночасно задовольняє потребу користувача у точній інформації та забезпечує емоційний відгук через відсоткове відображення досягнення, що реалізує механіку Core Drive 2 фреймворку Octalysis [7]. Прогрес-бар книги є безпосереднім відображенням атрибута progressPercentage з класу Quest, описаного у підрозділі 2.2, та заповнений

градієнтом від фіолетового до бурштинового відповідно до загальної кольорової схеми застосунку.

Банер активного квесту відображає поточне щоденне завдання («Прочитати 30 сторінок») з мініатюрним лічильником прогресу (18/30), що реалізує ефект градієнта мети, описаний у підрозділах 2.2 та 3.1. Відображення лічильника безпосередньо на головному екрані є свідомим проєктним рішенням: користувач бачить незавершене завдання без необхідності переходити на екран квестів, що підвищує ймовірність його виконання відповідно до принципів поведінкової психології [1, 20]. Іконка мечів зліва від банера є візуальним маркером ігрового завдання та відповідає іконці розділу «Квести» у нижній навігаційній панелі, створюючи візуальну узгодженість між елементами інтерфейсу [41].

Контентна частина екрану організована у три тематичні секції: «Лідери тижня», «Новинки» та «Бестселери». Кожна секція реалізована як горизонтально прокручуваний список карток книг із реальними обкладинками, назвою, автором, жанровою міткою та ціною у гривнях. Жанрові мітки реалізовані як кольорові чіпи («Саспенс», «Проза», «Фантастика»), що забезпечують швидку категоризацію без читання повного опису відповідно до принципу ефективності взаємодії [41]. Горизонтальна прокрутка карток реалізована через ListView.builder із scrollDirection: Axis.horizontal, що є стандартним патерном для каталогів у Flutter [18].

Нижня навігаційна панель містить чотири пункти: Головна, Пошук, Квести та Профіль. Така структура навігації базується на пріоритизації основних сценаріїв використання застосунку. Чотири пункти є оптимальною кількістю для нижньої навігації відповідно до рекомендацій платформ iOS [26] та Android [35]: така структура забезпечує комфортні зони дотику розміром не менше 48×48dp та охоплює всі ключові функціональні розділи без перевантаження панелі.

Винесення розділу «Квести» на рівень основної навігації є архітектурно обґрунтованим рішенням з точки зору гейміфікації: квести є основним щоденним механізмом залучення користувача і мають бути доступні в один дотик з будь-якого екрану [16]. Це рішення реалізує механіку уникнення втрат Core Drive 8 фреймворку Octalysis [7]: постійна видимість розділу квестів у навігації нагадує користувачу про активні незавершені завдання та знижує ризик пропуску щоденного квесту.

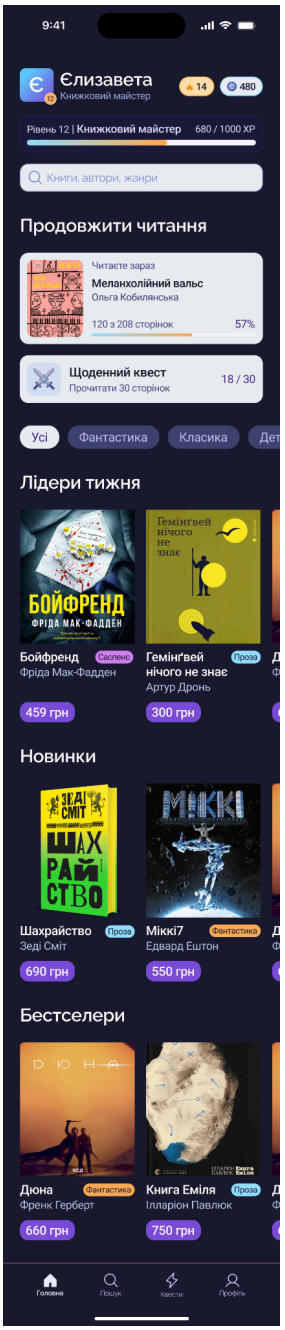


Рисунок 4.1 – Головний екран застосунку «SmartBook»

Екран читання з гейміфікованим прогрес-баром. Екран читання є функціонально центральним у застосунку «SmartBook», оскільки саме тут відбувається основна користувацька дія – читання – та генерується переважна більшість ігрових подій: нарахування XP за сторінку, оновлення прогресу квестів та активація тригерів підрозділу 3.2. Проєктування цього екрану підпорядковане принципу мінімального відволікання, сформульованому Шнайдерманом: гейміфіковані елементи інтегровані у периферійні зони інтерфейсу і не конкурують з основним контентом [46]. Такий підхід відповідає концепції «невидимої гейміфікації», де ігрові механіки підсилюють основну дію, не відволікаючи від неї [16]. Екран реалізовано у трьох станах: базовий стан читання, стан виділення тексту та панель налаштувань.

Верхня навігаційна панель містить три елементи: кнопку повернення, назву книги та дві функціональні іконки. Кнопка повернення реалізована як кругла кнопка діаметром 44pt з іконкою шеврона, що відповідає мінімальному розміру зони дотику стандартів iOS [26] та Android [35] і забезпечує моторну доступність відповідно до підрозділу 3.3.4. Назва книги розміщена по центру як заголовок навігаційної панелі відповідно до стандартної ієрархії навігації мобільних платформ [26]. Іконка закладки відображається у активному стані акцентним бурштиновим кольором із заповненням, що сигналізує про збережену позицію відповідно до принципу відображення стану системи Нільсена [41]. Це є прикладом мультимодального підтвердження дії: користувач бачить результат попередньої взаємодії без необхідності повторно перевіряти стан системи. Основна зона читання займає максимально можливу площу екрану відповідно до принципу пріоритизації контенту. Текст набрано шрифтом з засічками (Times New Roman) розміром 17pt з міжрядковим інтервалом 1.9, що відповідає оптимальним параметрам читабельності для мобільних пристроїв [33]. Вибір шрифту з засічками є свідомим проєктним рішенням: засічки покращують розрізнення літер при тривалому читанні та відтворюють звичний досвід читання паперової книги, що знижує когнітивне навантаження [41].

Колір тексту (#F0EEF8) на темному фоні (#1A1822) забезпечує коефіцієнт контрастності, що перевищує мінімальний поріг 4.5:1 стандарту WCAG 2.1 рівня AA [49], підтверджено верифікацією у Figma відповідно до методики підрозділу 3.3. Нижня частина тексту плавно затухає через градієнт прозорості, сигналізуючи про можливість прокрутки без явного індикатора скролу [35]. Горизонтальні відступи тексту становлять 24dp з обох боків, що відповідає рекомендованому діапазону довжини рядка 55–75 символів для комфортного читання [33].

Функціонал виділення тексту реалізований як окремий стан екрану читання і є ключовим елементом системи нотаток та цитат. При тривалому натисканні на текст активується режим виділення з плаваючою контекстною панеллю з чотирма діями: «Замітка», «Цитата», «Копіювати» та «Поділитись». Дія «Цитата» виділена акцентним фіолетовим кольором як пріоритетна – вона зберігає фрагмент у розділі цитат профілю та нараховує 5 XP відповідно до системи ваг таблиці 3.1. Порядок дій у панелі спроєктований за принципом спадання частоти використання зліва направо відповідно до патерну сканування інтерфейсу [41]. Розмір кожного елемента панелі не менший за 44×44pt відповідно до вимог моторної доступності [26, 35].

Панель налаштувань відкривається як модальний bottom sheet при натисканні іконки меню – патерн, що не перекриває весь екран та є ергономічно зручним для керування великим пальцем [26]. Панель містить три параметри: перемикач теми (темна/світла), вибір шрифту (Times New Roman за замовчуванням) та розмір шрифту (17pt). Реалізація перемикача теми через MediaQuery.of(context).platformBrightness забезпечує синхронізацію з системними налаштуваннями відповідно до підрозділів 1.4 та 3.3 [49, 26]. Нижня зона прогресу реалізує компонент GamifiedProgressBar, описаний у підрозділі 3.4.2, та відображає подвійний індикатор прогресу: абсолютний («120 з 208 сторінок») та відносний (57%). Такий формат одночасно задовольняє потребу у точній інформації та забезпечує емоційний відгук, реалізуючи

механіку Core Drive 2 фреймворку Octalysis [7]. Поточне значення 57% відповідає нейтральному стану, оскільки тригер _checkNearCompletion() активується лише при досягненні 80%, як описано у підрозділах 2.2 та 3.1. При досягненні цього порогу прогрес-бар змінює колір на зелений акцент, додає пульсуючу анімацію та з'являється банер «Майже фінал!» з нагородою +100 XP, що реалізує ефект градієнта мети та підвищує мотивацію до завершення книги [6, 7]. Семантична анотація прогрес-бару через Semantics-обгортку забезпечує коректне зачитування прогресу екранними дикторами TalkBack [35] та VoiceOver [26] відповідно до підрозділу 3.3 стандарту WCAG 2.1 [49].

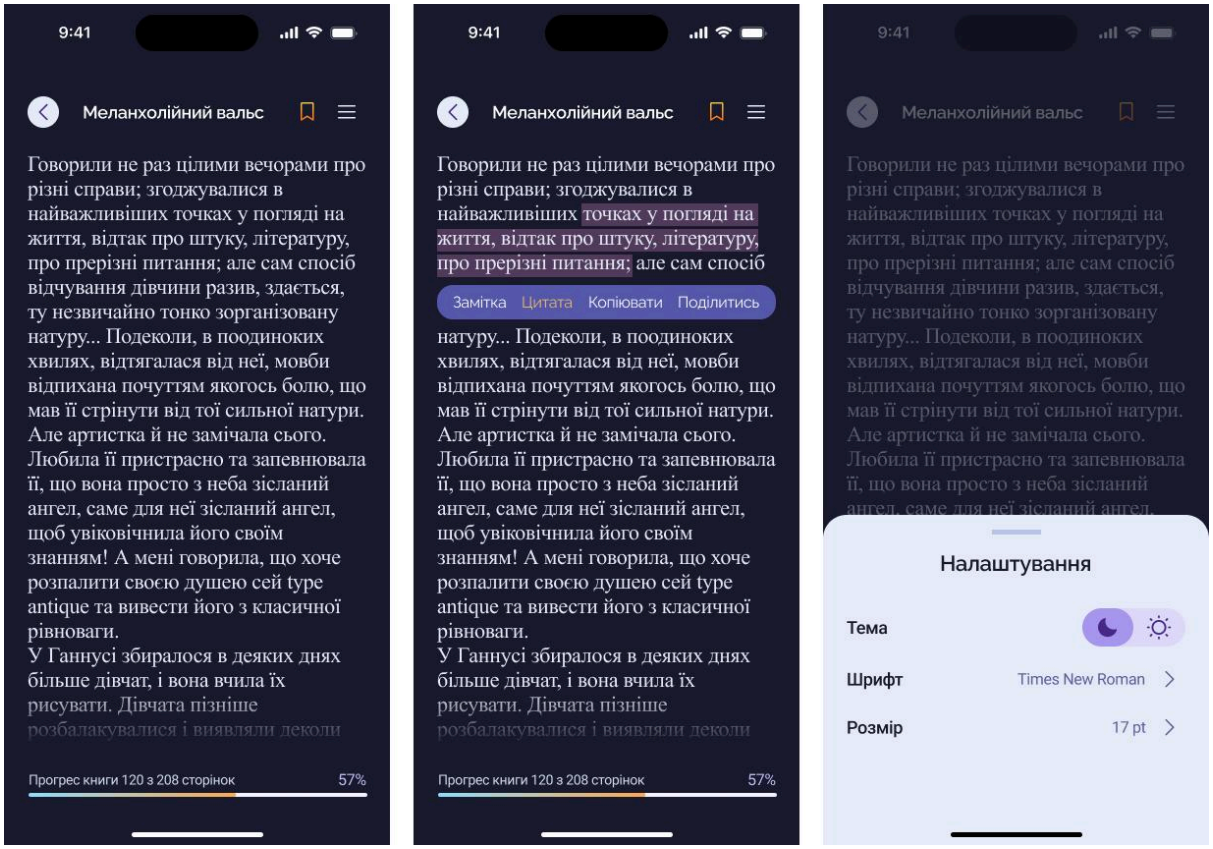


Рисунок 4.2 – Екран читання: базовий стан, стан виділення тексту, панель налаштувань

Екран профілю користувача. Екран профілю користувача виконує подвійну функцію у застосунку «SmartBook»: з одного боку, він є центральним хабом ігрового прогресу, де агрегуються всі ключові метрики гейміфікованої системи, з іншого – точкою доступу до налаштувань акаунту та персональної

бібліотеки. Відповідно до навігаційної діаграми станів підрозділу 2.3, екран профілю є одним із чотирьох основних навігаційних станів застосунку. Екран реалізований у двох станах: основний екран профілю та екран налаштувань.

Верхня частина екрану містить ідентифікаційний блок користувача: аватар, ім'я, ранговий чіп та два ігрових індикатори. Аватар реалізований як квадрат із заокругленими кутами та градієнтним фоном з ініціалом користувача – це забезпечує персоналізований вигляд без порожнього стану у разі відсутності фото. Кнопка редагування аватара відповідає патерну редагування, стандартизованому в рекомендаціях Apple [26] та Google [35]. Рангова мітка «Книжковий майстер» відображається як фіолетовий чіп із крапкою-індикатором, що реалізує механіку соціального статусу Core Drive 5 фреймворку Octalysis [7]: ранг є видимим підтвердженням досягнення певного рівня та стимулює подальший прогрес. Стрік та баланс монет розміщені у верхньому правому куті як окремі чіпи – аналогічно до їх відображення на головному екрані, що забезпечує візуальну узгодженість та відповідає принципу консистентності інтерфейсу [41].

Картка «Ціль на рік» відображає прогрес читацької мети у форматі «24 з 36 книг прочитано» та відсоткове значення (68%), що реалізує механіку довгострокової мотивації Core Drive 2 фреймворку Octalysis [7]. Іконка мечів зліва є тим самим символом, що використовується у банері квестів на головному екрані, створюючи семантичну узгодженість між ігровими елементами різних екранів [53]. Картка рівня відображає поточний рівень (12), ранг, поточне значення XP (680) та відстань до наступного рівня. Шкала прогресу XP заповнена градієнтом від фіолетового до бурштинового на основі getter-методу levelProgressPercentage підрозділу 2.2. Підпис «залишилось 320 XP» акцентним кольором реалізує ефект градієнта мети та стимулює користувача до активності [6]. Компонент є прямою реалізацією XpProgressBar, описаного у підрозділі 3.4. Картка бібліотеки відображає кількість книг (12), підзаголовок «Вподобані · В процесі · Прочитані» та горизонтальну стрічку

мініатюр обкладинок з лічильником (+40). Це дозволяє користувачу бачити факт наявності бібліотеки та її вміст без необхідності заходити всередину, відповідаючи принципу видимості стану системи [41]. Реальні обкладинки підсилюють емоційний зв'язок з особистою колекцією та реалізують механіку власності Core Drive 4 [7].

Нижній блок відображає три агреговані показники: кількість сторінок (12k), книг прочитано (24) та досягнень (8). Статистика є похідною від атрибутів класу UserProgress підрозділу 2.2 та оновлюється в реальному часі через BLoC-потік [4].

Екран налаштувань організований як список із чотирьох функціональних пунктів та окремої кнопки виходу. Кожен пункт містить кольорову іконку, назву, підзаголовок та шеврон переходу – стандартний патерн для екранів налаштувань у мобільних застосунках [41, 26]. Пункт «Підписка» відображає поточний статус («Неактивна»). Пункт «Інтеграція з хмарою» надає доступ до управління синхронізацією з Firebase Firestore, описаною у підрозділі 2.4, що є критичним для збереження ігрового прогресу між пристроями відповідно до підрозділу 1.4.3 [19]. Кнопка «Вихід» розміщена окремо у виділеному контейнері з червоною рамкою – стандартний патерн деструктивної дії у мобільному дизайні [26, 35].

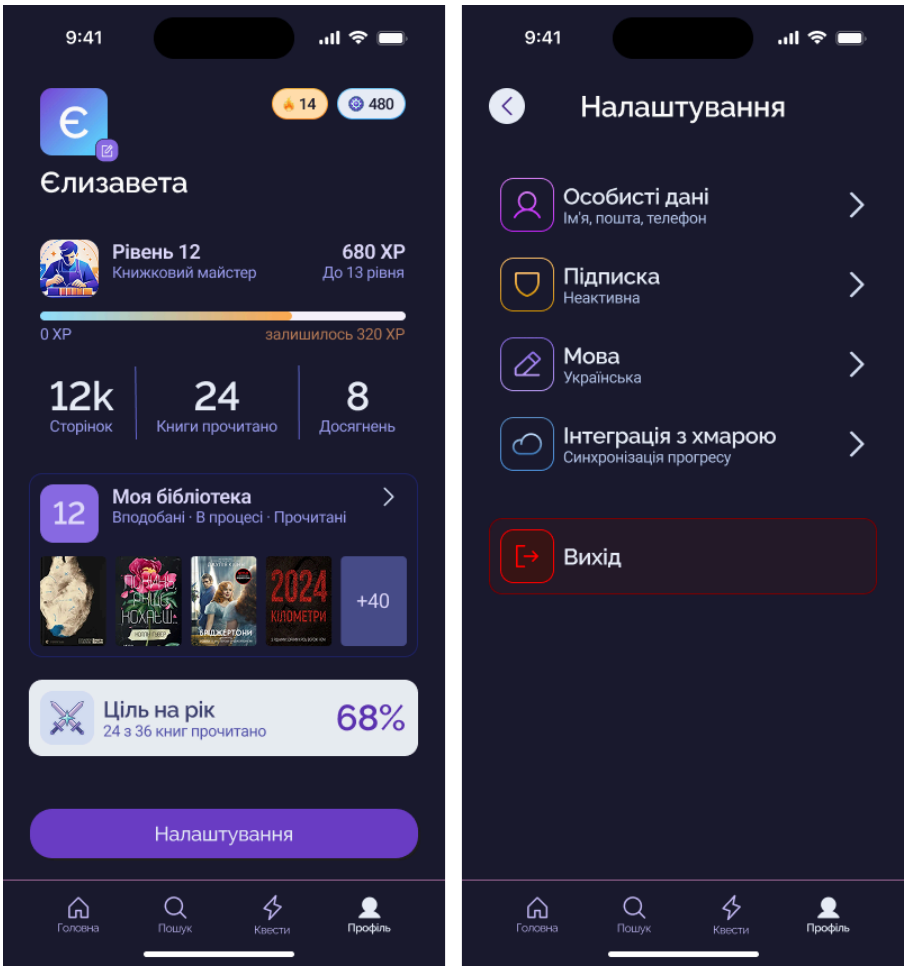


Рисунок 4.3 – Екран профілю користувача (ліворуч) та екран налаштувань (праворуч)

Екран квестів та досягнень. Екран квестів є основним ігровим хабом застосунку «SmartBook» та центральним елементом щоденного циклу залучення користувача. Він доступний через пункт «Квести» нижньої навігаційної панелі з іконкою блискавки, що забезпечує доступ в один дотик з будь-якого екрану відповідно до архітектурного рішення підрозділу 4.1. Екран організований як три окремі вкладки: «Активні», «Завершені» та «Досягнення», що відповідають трьом функціонально відмінним контекстам взаємодії користувача з ігровою системою.

Вкладка «Активні». Верхня частина містить стрік-банер з бурштиновим акцентом, що відображає поточну серію (14 днів поспіль) та мотиваційний підзаголовок «Читайте сьогодні, щоб зберегти стрік». Банер реалізує механіку

уникнення втрат Core Drive 8 фреймворку Octalysis [7]: постійна видимість стріку формує психологічний тиск збереження накопиченого результату, що є одним із найефективніших механізмів утримання аудиторії [1]. Блок «Прогрес тижня» відображає барчарт активності за останні 7 днів з відсотковим показником (71%), де поточний день виділяється світлішим відтінком, що реалізує механіку вимірюваного прогресу Core Drive 2 [7]. Розділ щоденних квестів містить картки завдань із трьома станами відповідно до машини станів класу Quest підрозділу 2.3. Картка «Прочитати 30 сторінок» (25/30, 83%) відображається з бурштиновим прогрес-баром та активною анімацією – стан active з досягненням порогу 80%, що активує тригер _checkNearCompletion() підрозділу 3.2, нагорода +50 XP відображається як зелений чіп [6]. Картка «Написати відгук» (100%) відображається у стані completed з доступною нагородою +25 XP. Картка «Завершити книгу» (57%) перебуває у нейтральному стані active без додаткових акцентів відповідно до алгоритму підрозділу 3.1[7].

Вкладка «Завершені». Верхня частина містить три зведені показники: 1840 XP зароблено, 36 квестів завершено та 740 монет зароблено. Відсутність рамок у цьому блоці є свідомим дизайнерським рішенням: великі числа без зайвого оздоблення створюють відчуття значущості досягнутого результату та реалізують механіку власності Core Drive 4 [7]. Завершені квести згруповані за часовим принципом у три секції: «Сьогодні», «Цього тижня» та «Раніше» відповідно до принципу природного відображення часових даних [41]. Відсутність кнопки отримання нагороди підтверджує термінальний стан claimed усіх елементів та унеможливлює повторне нарахування відповідно до захисного механізму підрозділу 3.1 [50].

Вкладка «Досягнення». Верхній банер відображає прогрес галереї: 8 з 15 досягнень розблоковано, 53% відкрито, кількість нових досягнень цього місяця (3). Прогрес-бар банера реалізує довгострокову мотивацію через відчуття неповноти колекції [16]. Блок «Рідкісне досягнення» відображається окремим повношириним елементом з бурштиновим акцентом – досягнення «14 днів

поспіль» з нагородою +500 XP. Виділення рідкісних досягнень реалізує ієрархію цінності та підсилює статусну механіку Core Drive 5 [7]. Основна частина вкладки організована як сітка 3×N із картками, згрупованими за категоріями «Читання», «Стрік» та «Спільнота». Розблоковані картки мають кольорову рамку та відображають нагороду XP і монети. Заблоковані картки відображаються у приглушеному стилі з підписом поточного прогресу («24/25 книг», «14/30 днів»), що реалізує механіку дефіциту Core Drive 6 [7]: видимість прогресу до заблокованого досягнення стимулює читацьку активність та забезпечує постійну наявність актуальних цілей на будь-якому етапі взаємодії [6].

Рисунок 4.4 – Екран квестів: вкладка «Активні» (ліворуч), «Завершені» (центр), «Досягнення» (праворуч)

Анімаційні переходи та мікроінтеракції. Анімаційні переходи та мікроінтеракції є завершальним шаром гейміфікованого досвіду, що перетворює статичний інтерфейс на живу реактивну систему. Відповідно до досліджень Нільсена, анімація виконує три функції: інформує про зміну стану, забезпечує просторову орієнтацію при навігації та підсилює емоційний відгук [41]. Саффер визначає мікроінтеракції як детальні моменти взаємодії, що формують загальне враження від продукту [45]. У контексті гейміфікації емоційна функція анімації є особливо критичною: анімація Level Up або розблокування досягнення має викликати відчуття винагороди [1]. Усі анімації реалізовані через бібліотеку flutter_animate та дотримуються вимоги часу відгуку до 200 мс підрозділу 1.4, а також адаптуються до системного налаштування Reduce Motion відповідно до підрозділу 3.3 [49, 26].

Навігаційні переходи між основними екранами реалізовані через горизонтальний слайд-перехід зі згасанням (slide + fade) тривалістю 280 мс із кривою easeInOut відповідно до рекомендованого діапазону для навігаційних переходів [36]. Перехід до екрану читання реалізований як вертикальний push-перехід знизу вгору тривалістю 350 мс із кривою easeOut. Зворотній перехід виконується зверху вниз за 300 мс. Вибір різних кривих відповідає принципу асиметрії руху Material Motion [36]: вхід має бути плавнішим, вихід – швидшим.

Анімація підвищення рівня активується при виконанні умови total_xp >= threshold(currentLevel + 1) у методі _checkLevelUp() підрозділу 3.1.2 та складається з чотирьох фаз загальною тривалістю 600 мс. Для складних покадрових ефектів – розсипання частинок та світіння – застосовується бібліотека Lottie [32]. Перша фаза (0–150 мс) – імпульс прогрес-бару до 100% зі спалахом зеленого світіння. Друга фаза (150–350 мс) – розширення LevelBadge

до 130% через ScaleTransition із кривою elasticOut та перегортання цифри рівня через AnimatedFlipCounter. Третя фаза (350–500 мс) – відображення нового рівня та рангу як тост у верхній частині екрану. Четверта фаза (500–600 мс) – скидання шкали XP до нульового значення нового рівня. Загальна тривалість 600 мс не перевищує порогу сприйняття як затримки [41].

Анімація claimReward() активується при натисканні кнопки «Отримати» та реалізує перехід зі стану completed до claimed підрозділу 2.3.3 за 500 мс. Перша фаза (0–200 мс) – кнопка стискається до нуля, а з картки вилітають індикатори +50 XP та +20 з анімацією fly + fade out – стандартний патерн відображення нарахування очок, що передає результат без переривання потоку взаємодії [45]. Друга фаза (200–400 мс) – картка переходить у стан claimed з приглушеним фоном. Третя фаза (400–500 мс) – картка переміщується до нижньої частини списку через AnimatedList. При досягненні 80% прогресу спрацьовує _checkNearCompletion() підрозділу 3.2.2: прогрес-бар змінює колір до яскравого зеленого та збільшує товщину з 5dp до 7dp через AnimatedContainer тривалістю 300 мс, після чого активується безперервна пульсація з циклом 1500 мс. Саффер визначає такі безперервні мікроінтеракції як «петлі» [45]. При Reduce Motion обидві фази замінюються миттєвою зміною кольору [49].

Анімація розблокування досягнення реалізує FlipCard-ефект тривалістю 500 мс: картка обертається до 90 градусів, відкриваючи розблокований стан. Одночасно активується HapticFeedback.mediumImpact() відповідно до рекомендацій Apple [26] та принципу мультимодального підтвердження [45]. Іконки нижньої навігаційної панелі виконують мікроанімацію масштабування до 110% за 120 мс із кривою bounceOut відповідно до принципу «швидкого відскоку» Material Motion [36]. Кнопки дій при натисканні виконують анімацію занурення – зменшення до 96% за 80 мс відповідно до концепції Саффера [45] та параметрів Material Design [36].

АНІМАЦІЯ	ТРИВАЛІСТЬ	КРИВА	АДАПТАЦІЯ REDUCE MOTION
НАВІГАЦІЙНИЙ ПЕРЕХІД	280 мс	easeInOut	Миттєва зміна
ПЕРЕХІД ДО РИДЕРА	350 мс	easeOut	Fade 150 мс
LEVEL UP (ПОВНА)	600 мс	elasticOut	Статичний тост
CLAIM REWARD	500 мс	easeInOut	Миттєва зміна стану
ПРОГРЕС-БАР 80%	300 мс + ∞	linear	Зміна кольору
РОЗБЛОКУВАННЯ ДОСЯГНЕННЯ	500 мс	linear	Миттєва поява
НАВІГАЦІЙНА ІКОНКА	500 мс	bounceOut	Без змін

Таблиця 4.1 – Зведена характеристика анімацій прототипу

Таким чином, описана система анімаційних переходів та мікроінтеракцій забезпечує безперервний емоційний зворотний зв'язок між діями користувача та реакцією системи, реалізуючи принцип «петель залучення» [1]. Усі анімації дотримуються часових обмежень підрозділу 1.4, адаптуються до системних налаштувань доступності підрозділу 3.3 та реалізовані через Flutter-інструментарій підрозділу 2.4.

4.2 Опис логіки роботи гейміфікованих віджетів

Гейміфіковані віджети є технічним втіленням усіх архітектурних рішень, описаних у попередніх розділах: вони поєднують алгоритми підрозділу 3.1, систему тригерів підрозділу 3.2 та методику інтеграції підрозділу 3.4 в єдині самодостатні Flutter-компоненти [18]. Підрозділ 4.2 описує внутрішню логіку роботи кожного віджету – від отримання даних через BLoC-потік до фінального відображення на екрані [4]. Розуміння цієї логіки є необхідною передумовою для проведення тестування у підрозділі 4.3, оскільки тестові сценарії будуються саме навколо поведінки цих компонентів [47]. Мартін підкреслює, що якісно спроєктований компонент має бути самодостатнім та незалежним від контексту

використання: лише за цієї умови він може бути ефективно протестований та повторно використаний у різних частинах системи [34].

Логіка роботи GamifiedProgressBar. Компонент GamifiedProgressBar є фундаментальним гейміфікованим віджетом застосунку «SmartBook» – він присутній одночасно на чотирьох екранах: у хедері головного екрану (шкала XP), на екрані читання (прогрес книги), у картках квестів (прогрес завдання) та на екрані профілю (прогрес рівня). Попри зовні однакову природу, кожен екземпляр компонента є самодостатнім та отримує дані з різних джерел через BLoC-потоки, не маючи прямих залежностей між собою. Це відповідає принципу єдиної відповідальності (Single Responsibility Principle): кожен екземпляр знає лише про своє поточне значення та не має доступу до стану інших екземплярів [34]. Архітектурна ізоляція компонентів є критичною для масштабованості системи – додавання нового екземпляра прогрес-бару не потребує змін у існуючому коді [22].

Шкала XP у хедері головного екрану та на екрані профілю отримує значення з getter-методу levelProgressPercentage класу UserProgress, описаного у підрозділі 2.2. Цей getter обчислюється динамічно як відношення накопиченого досвіду в межах поточного рівня до загального обсягу досвіду, необхідного для переходу на наступний рівень, та повертає значення у діапазоні від 0.0 до 1.0. Такий підхід відповідає принципу єдиного джерела істини (Single Source of Truth): UI-компонент ніколи не зберігає власну копію даних, а завжди звертається до канонічного джерела [34].

Прогрес-бар на екрані читання отримує значення з атрибута progressPercentage поточного об'єкта Quest, що оновлюється методом updateProgress() при кожній зміні сторінки. Відповідно до досліджень у галузі мобільної взаємодії, оновлення прогресу має відбуватись у реальному часі без помітних затримок – затримка більше 100 мс між дією та відображенням результату руйнує відчуття безперервності взаємодії [41]. Прогрес-бари у картках квестів отримують значення безпосередньо з поля progressPercentage

відповідного екземпляра класу Quest, описаного у підрозділі 2.2. Усі чотири джерела підписані на відповідні BLoC-потоки та перебудовують компонент через механізм StreamBuilder без необхідності ручного керування станом [4]. Реактивний підхід до управління станом через потоки є рекомендованою практикою для Flutter-застосунків зі складною бізнес-логікою [18]. Внутрішня логіка GamifiedProgressBar реалізована як скінченний автомат із трьома станами відображення, що визначаються поточним значенням прогресу. Використання патерну скінченного автомату для управління станами UI-компонентів є рекомендованою практикою, що унеможливлює виникнення некоректних або суперечливих станів інтерфейсу [34]. Переходи між станами є односпрямованими в межах одного циклу прогресу та відбуваються автоматично без зовнішнього втручання.

Стан Normal (0.0-0.59). У цьому стані прогрес-бар відображається з нейтральним градієнтним заповненням від фіолетового до зеленого та стандартною висотою 5dp. Жодних додаткових візуальних ефектів не застосовується – компонент є функціональним, але не відволікає увагу користувача від основного контенту. Це відповідає принципу мінімального відволікання при читанні, обґрунтованому Шнайдерманом [46]: периферійні елементи інтерфейсу не повинні конкурувати з основним контентом за увагу користувача.

Стан Warning (0.60-0.79). При перетині порогу 60% компонент виконує плавний колірний перехід до насиченішого фіолетового через ColorTween тривалістю 300 мс, що відповідає рекомендованому діапазону тривалості для колірних переходів у мобільних інтерфейсах [36]. Цей стан є проміжним сигнальним: він не активує повну мікроінтеракцію, але привертає увагу користувача до наближення мети. Поріг 60% обрано відповідно до принципу поступового розкриття інформації [41]: занадто ранній акцент призвів би до звикання і втрати ефекту при досягненні критичного порогу 80%. Саффер

визначає такий підхід як «ескалацію мікроінтеракції» – поступове посилення сигналу у міру наближення до критичної події [45].

Стан Near (0.80-1.0). При перетині порогу 80% активується повна мікроінтеракція відповідно до алгоритму _checkNearCompletion(), описаного у підрозділах 2.2 та 3.2. Колір бару переходить до яскравого зеленого через ColorTween тривалістю 300 мс, висота збільшується з 5dp до 7dp через AnimatedContainer, активується безперервна пульсація світіння з циклом 1500 мс. Одночасно генерується подія NearCompletionEvent у BLoC-потік, що дозволяє іншим компонентам екрану реагувати на цей стан. Вибір порогу 80% обґрунтований психологічним ефектом градієнта мети (Goal Gradient Effect): дослідження демонструють, що мотивація до завершення задачі суттєво зростає при наближенні до фінішу [6]. Детермінінг та Наке підтверджують, що візуальне підкріплення на фінальному етапі виконання завдання є одним із найефективніших механізмів утримання залученості у гейміфікованих системах [14]. При досягненні значення 1.0 поведінка компонента залежить від контексту використання. У картці квесту прогрес-бар фіксується у максимальному стані, очікуючи виклику claimReward() – це стан completed машини станів підрозділу 2.3. На екрані профілю при підвищенні рівня компонент XpProgressBar виконує послідовність «заповнення – пауза – скидання»: спочатку анімовано заповнюється до 100% за 150 мс, витримує паузу, після чого скидається до нульового значення нового рівня через плавний перехід тривалістю 300 мс. Ця послідовність реалізується через ланцюжок AnimationController відповідно до принципів Flutter-анімацій [29] та створює чітке відчуття завершення одного циклу та початку наступного, що є важливим психологічним маркером прогресу [1]. На екрані читання при завершенні книги прогрес-бар залишається у максимальному стані до видалення книги з активного читання. Компонент реалізує дворівневий захист від некоректних вхідних значень. На рівні Flutter-віджету застосовується метод .clamp(0.0, 1.0), що гарантує неможливість виходу за межі допустимого діапазону незалежно від значень, переданих з

BLoC-потоку. На рівні бізнес-логіки аналогічний захист реалізований у getter-методі levelProgressPercentage класу UserProgress, описаному у підрозділі 2.2. Подвійний захист відповідає принципу глибинного захисту (Defense in Depth) та унеможливлює відображення некоректного стану навіть при потенційних помилках синхронізації даних [34]. Верберг та Хантер наголошують, що цілісність ігрових метрик є критичною для довіри користувача до системи: будь-яке некоректне відображення прогресу руйнує відчуття справедливості винагороди [50].

Семантична анотація реалізована через Semantics-обгортку з трьома динамічними атрибутами відповідно до вимог стандарту WCAG 2.1 [49]. Атрибут label містить контекстний опис компонента залежно від екрану: «Прогрес рівня», «Прогрес читання» або «Прогрес квесту». Атрибут value формується як рядок «{N} відсотків» та оновлюється при кожній зміні значення. Атрибут liveRegion встановлено у значення polite для звичайних оновлень – екранний диктор зачитує нове значення після завершення поточного оголошення, не перериваючи користувача, що відповідає рекомендаціям Apple [26] та Google [35]. При переході у стан Near атрибут автоматично змінюється на assertive, що змушує TalkBack та VoiceOver негайно оголосити досягнення порогу 80% відповідно до підрозділу 3.3.3. Мінімальна висота зони взаємодії становить 44dp відповідно до вимог моторної доступності [26, 35]. При активованому MediaQuery.of(context).disableAnimations компонент повністю відключає анімаційні переходи між станами відповідно до підрозділу 3.3 [49]. Зміна кольору відбувається миттєво без ColorTween, висота бару змінюється без AnimatedContainer, пульсація у стані Near вимикається. Усі функціональні стани зберігаються – користувач отримує повний обсяг інформації у статичному вигляді. Apple підкреслює, що адаптація до Reduce Motion є обов'язковою умовою публікації застосунку в App Store [26]. Це реалізує принцип прогресивного покращення: базовий функціонал доступний усім, анімаційне збагачення – лише тим, для кого воно є комфортним [49].

Логіка роботи LevelBadge та XpProgressBar. Компоненти LevelBadge та XpProgressBar утворюють нерозривну функціональну пару, що відображає поточний ігровий статус користувача. LevelBadge візуалізує дискретний атрибут currentLevel, тоді як XpProgressBar відображає безперервний прогрес у межах цього рівня через getter-метод levelProgressPercentage, описаний у підрозділі 2.2. Обидва компоненти підписані на один BLoC-потік UserProgressBloc та перебудовуються синхронно при кожній зміні стану профілю відповідно до принципу єдиного джерела істини [34, 4]. Зикерман та Каннінгем визначають пару «рівень + шкала прогресу» як найефективнішу комбінацію гейміфікованих елементів для довгострокового утримання користувача, оскільки вона одночасно показує досягнуте (рівень) та майбутнє (прогрес до наступного) [53].

LevelBadge реалізує два стани: Stable – стандартне відображення рівня без анімацій, та LevelUp – анімована послідовність тривалістю 600 мс, описана у підрозділі 4.1. У стані Stable розмір контейнера варіюється залежно від контексту: 36×36dp на екрані профілю та 16×16dp у хедері головного екрану. Менший розмір у хедері є свідомим рішенням – надмірно великий значок відволікав би увагу від основного контенту відповідно до принципу мінімального відволікання [46]. Обидва екземпляри є одним Flutter-компонентом з параметром size, що реалізує принцип DRY та відповідає архітектурним рекомендаціям для масштабованих Flutter-застосунків [34, 18]. Перехід у стан LevelUp відбувається при отриманні події LevelUpEvent з UserProgressBloc, яку генерує метод _checkLevelUp() при виконанні умови total_xp >= threshold(currentLevel + 1), описаної у підрозділі 3.1. Числове значення рівня при переході анімується через AnimatedFlipCounter з тривалістю 200 мс та кривою easeInOut, що відповідає рекомендованому діапазону для числових анімацій [36]. Хамарі та Лехдонвірта підкреслюють, що анімоване відображення зміни числового показника підсилює відчуття досягнення порівняно з миттєвою зміною значення, оскільки мозок інтерпретує анімований перехід як підтвердження реальності події [25].

XpProgressBar є спеціалізованим розширенням базового GamifiedProgressBar з додатковою логікою обробки події підвищення рівня. У звичайному режимі компонент успадковує всі три стани (Normal, Warning, Near) від базового компонента, описаного у підрозділі 4.2. Числові підписи «680 XP» та «залишилось 320 XP» формуються динамічно з атрибутів total_xp та xp_to_next_level класу UserProgress. Підпис «залишилось N XP» відображається акцентним кольором та постійно зменшується при нарахуванні XP, що реалізує ефект градієнта мети (Goal Gradient Effect) – дослідження підтверджують, що числова близькість до цілі суттєво підвищує мотивацію до активності [6].

При отриманні LevelUpEvent компонент виконує послідовність із трьох кроків відповідно до принципів Flutter-анімацій [18, 29]. Перший крок (0–150 мс) – прискорене заповнення до 100% через Tween<double> з кривою easeIn, що створює відчуття «досягнення фінішу» навіть якщо XP до підвищення рівня нараховано стрибком. Другий крок (150–300 мс) – фіксація у максимальному стані на 150 мс: пауза дає користувачу час усвідомити момент досягнення перед скиданням відповідно до принципів мікроінтеракцій [45]. Третій крок (300-600 мс) – скидання до значення levelProgressPercentage нового рівня через Tween<double> з кривою easeOut з одночасним оновленням числових підписів через AnimatedFlipCounter.

Анімації LevelBadge та XpProgressBar координуються через спільний AnimationController на рівні батьківського BLoC-компонента, що забезпечує єдину узгоджену послідовність замість двох незалежних ефектів [4]. Саффер визначає таку координацію як «хореографію мікроінтеракцій» – найвищий рівень деталізації інтерфейсу, що формує відчуття цілісності продукту [45]. Семантична анотація LevelBadge при переході у стан LevelUp тимчасово встановлює атрибут liveRegion: assertive, що змушує TalkBack та VoiceOver негайно оголосити «Рівень підвищено до {N}» відповідно до рекомендацій Apple та Google [26, 35]. XpProgressBar успадковує семантичну анотацію базового GamifiedProgressBar з атрибутом value у форматі «{N} відсотків» та

liveRegion: polite для звичайних оновлень відповідно до стандарту WCAG 2.1 [49]. При активованому системному налаштуванні Reduce Motion обидва компоненти замінюють усі анімаційні переходи миттєвими змінами значень відповідно до підрозділу 3.3 [49, 26].

Логіка роботи StreakCounter. Компонент StreakCounter відображає поточне значення атрибута daily_streak класу UserProgress та є одним із ключових елементів механіки утримання користувача. На відміну від LevelBadge та XpProgressBar, що реагують на накопичення XP, StreakCounter реагує на часовий патерн активності – факт щоденного повернення до застосунку. Верберг та Хантер визначають стрік як один із найпотужніших механізмів формування звички у гейміфікованих системах: користувач повертається до застосунку не заради винагороди, а щоб не втратити накопичений результат [50]. Чоу відносить цей механізм до Core Drive 8 (уникнення втрат) фреймворку Octalysis та наголошує, що він є найефективнішим короткостроковим драйвером поведінки [7]. StreakCounter реалізує три візуальні стани залежно від значення daily_streak, між якими відбуваються автоматичні переходи при оновленні BLoC-потоку UserProgressBloc.

У стані Inactive (streak = 0) компонент відображається у приглушеному сірому кольорі без іконки полум'я. Відсутність яскравих акцентів є свідомим проєктним рішенням: негативне підкріплення через яскраве червоне попередження при нульовому стріку демотивує нових користувачів та суперечить принципам етичного дизайну залучення [46]. Натомість приглушений стиль ненав'язливо сигналізує про відсутність активності, залишаючи простір для позитивної мотивації. У стані Active (streak 1–6) компонент відображається з іконкою полум'я та числовим значенням у бурштиновому кольорі. Бурштиновий колір обрано як семантично нейтральний між зеленим (прогрес) та червоним (небезпека) – він передає теплоту та енергію без тривожності [46]. Числове значення оновлюється щоденно при виклику методу updateActivity(), описаного у підрозділі 2.2. У стані Elite (streak 7+) до

базового відображення додається анімований ореол – пульсуюче кільце навколо компонента з циклом 2000 мс та амплітудою box-shadow від 0 до 10dp. Поріг у 7 днів відповідає тижневому циклу активності та є психологічно значущим – один тиждень поспіль є першим відчутним досягненням для більшості користувачів [1]. Ряан та Десі у теорії самодетермінації визначають такі компетентнісні маркери як критичні для формування внутрішньої мотивації [44].

СТАН	ДІАПАЗОН	КОЛІР	АНІМАЦІЯ
INACTIVE	0	Сірий приглушений	Відсутня
ACTIVE	1–6	Бурштиновий	Відсутня
ELITE	7+	Бурштиновий + ореол	Пульсація 2000 мс

Таблиця 4.3 – Стани компонента StreakCounter

Важливою деталлю є те, що StreakResetEvent генерується не в момент фактичного скидання (опівночі), а при першому відкритті застосунку після пропуску дня. Це рішення відповідає принципу відкладеного негативного підкріплення: система не «карає» користувача в момент відсутності активності, а лише фіксує факт пропуску при наступному візиті [20]. Такий підхід є більш етичним та менш демотивуючим порівняно з негайним скиданням [43].

StreakCounter є одним із двох компонентів застосунку (поряд із QuestCard), що мають прямий зв'язок із системою зовнішніх тригерів, описаною у підрозділі 3.2. При значенні daily_streak більше 3 та відсутності активності протягом поточного дня система генерує пуш-сповіщення «Ваш стрік під загрозою» відповідно до шаблону таблиці 3.4. Поріг у 3 дні обрано як мінімальний для формування психологічної прив'язаності до стріку: дослідження показують, що користувачі починають активно захищати стрік лише після 3-4 днів безперервної активності [16]. Семантична анотація компонента реалізована через атрибут label зі значенням «Стрік {N} днів поспіль», що оновлюється при кожній зміні daily_streak [49]. При переході у

стан Elite атрибут liveRegion встановлюється у polite для оголошення досягнення порогу 7 днів [26, 35]. При активованому Reduce Motion пульсація ореолу у стані Elite замінюється статичним кільцем без анімації відповідно до підрозділу 3.3 [49, 26].

Логіка роботи CurrencyWidget. Компонент CurrencyWidget відображає поточне значення атрибута softCurrency класу VirtualEconomy та забезпечує візуальний зворотний зв'язок при кожній транзакції з ігровою валютою. На відміну від компонентів прогресу, що реагують на накопичення, CurrencyWidget реагує на двосторонній потік операцій – нарахування (earn()) та списання (spend()), описаних у підрозділі 2.2. Верберг та Хантер визначають віртуальну валюту як механіку власності (Core Drive 4), що формує у користувача відчуття відповідальності за накопичений ресурс та стимулює раціональне управління балансом [50]. Саффер наголошує, що миттєвий візуальний відгук при фінансових операціях є критичним для довіри користувача до системи: відсутність підтвердження транзакції сприймається як збій [45].

Компонент підписаний на BLoC-потік VirtualEconomyBloc та отримує оновлення при кожному виклику методів earn() або spend() класу VirtualEconomy. На відміну від компонентів прогресу, CurrencyWidget не має власних станів з різним візуальним оформленням – він завжди відображає числове значення softCurrency в одному і тому самому форматі. Натомість його логіка зосереджена на анімованій реакції на зміну балансу та захисті від некоректного відображення при одночасних транзакціях [34, 4]. Ключовою функціональністю компонента є відображення дельти при кожній транзакції. При нарахуванні (earn()) з правого краю компонента з'являється числовий індикатор зеленого кольору формату +{N}, що рухається вгору та згасає за 600 мс через комбінацію SlideTransition та FadeTransition. При списанні (spend()) аналогічний індикатор бурштинового кольору формату -{N} рухається вниз та згасає за той самий проміжок часу. Напрямок руху дельти є семантично значущим: вгору асоціюється із зростанням та позитивним результатом, вниз –

зі зменшенням та витратою [36]. Саффер визначає такі анімовані індикатори результату як «фідбек-петлі» мікроінтеракцій: користувач миттєво розуміє факт, напрямок та обсяг зміни без необхідності порівнювати числові значення [45].

Основне числове значення балансу оновлюється через AnimatedFlipCounter з тривалістю 400 мс незалежно від черги дельт, що гарантує коректність відображуваного значення в будь-який момент часу [29]. Хамарі та Лехдонвірта підкреслюють, що анімоване збільшення балансу є значно ефективнішим підкріпленням порівняно з миттєвою зміною числа: користувач «бачить» зростання свого капіталу, що підсилює відчуття винагороди [25].

При одночасному виконанні кількох транзакцій підряд дельти відображаються послідовно з затримкою 300 мс між кожною через чергу Queue<TransactionDelta>. Це унеможливлює накладання дельт одна на одну та забезпечує чіткість відображення кожної окремої транзакції [34]. Якщо кількість транзакцій у черзі перевищує три, система об'єднує їх в одну сумарну дельту +{total}, що відповідає принципу обмеження когнітивного навантаження: більше трьох одночасних змін перестають сприйматись як окремі події [38].

CurrencyWidget має прямий зв'язок з методом canAfford() класу VirtualEconomy, описаним у підрозділі 2.2, через механізм реактивного оновлення стану кнопок покупки. При кожній зміні softCurrency BLoC-потік оновлює стан усіх кнопок «Придбати за монети» на екрані каталогу через BlocBuilder [4]: якщо поточний баланс є меншим за ціну товару, кнопка переходить у стан disabled з текстом «Недостатньо монет». Це рішення унеможливлює ініціювання завідомо неуспішної транзакції та відповідає принципу попередження помилок (Error Prevention) стандарту WCAG 2.1 [49]. Нільсен визначає попередження помилок як більш ефективну стратегію порівняно з обробкою вже виниклих помилок [41].

Семантична анотація компонента реалізована через атрибут label зі значенням «Баланс монет: {N}», що оновлюється при кожній транзакції [49]. Атрибут liveRegion: polite забезпечує оголошення зміни балансу екранними дикторами після завершення поточного зачитування, не перериваючи користувача [26, 35]. При активованому Reduce Motion анімація дельт та AnimatedFlipCounter замінюються миттєвою зміною значення відповідно до підрозділу 3.3 [49, 26]. Мінімальний розмір зони взаємодії компонента у хедері становить 44×44dp відповідно до вимог моторної доступності [26, 35].

Логіка роботи QuestCard та AchievementBadge. Компоненти QuestCard та AchievementBadge є найбільш складними гейміфікованими віджетами застосунку з точки зору кількості внутрішніх станів та логіки переходів між ними. Обидва компоненти безпосередньо реалізують машини станів, формалізовані у діаграмах підрозділу 2..3, та є фінальною ланкою в ланцюгу: алгоритм (підрозділ 3.1) → тригер (підрозділ 3.2) → UI-компонент (підрозділ 4.2.5). Чоу наголошує, що саме момент отримання нагороди є найбільш емоційно значущим у гейміфікованій системі – і саме ці два компоненти відповідають за його реалізацію [7].

QuestCard реалізує повну машину станів квесту з чотирма станами: active, completed, claimed та expired, що відповідають значенням переліку QuestStatus, описаного у підрозділі 2.2. Кожен стан має власне візуальне оформлення, набір доступних взаємодій та правила переходу до наступного стану. У стані active картка відображає прогрес-бар на основі GamifiedProgressBar з усіма трьома станами (Normal, Warning, Near), описаними у підрозділі 4.2, нагороду XP та монети як зелені чіпи у правому верхньому куті, а також заблоковану кнопку «Виконується» у сірому кольорі. Заблокована кнопка є проактивним елементом доступності: вона інформує користувача про недоступність дії без необхідності її спробувати, що відповідає принципу Error Prevention Нільсена [41].

Перехід зі стану active до completed відбувається при досягненні progressPercentage == 100.0 у методі updateProgress(). При цьому QuestCard виконує трифазну анімацію тривалістю 400 мс: фон картки плавно переходить до акцентного кольору через ColorTween, прогрес-бар заповнюється до максимуму, кнопка «Виконується» замінюється на активну кнопку «Отримати» через AnimatedSwitcher [36]. Детермінінг та Наке підкреслюють, що візуальна зміна стану при завершенні завдання є обов'язковим елементом гейміфікованої системи: без неї користувач не отримує чіткого підтвердження досягнення [13].

У стані completed єдиною доступною дією є натискання кнопки «Отримати», що викликає метод claimReward(). Це єдиний стан, у якому виклик цього методу є дозволеним відповідно до захисного механізму підрозділу 3.1. При натисканні виконується анімація отримання нагороди, описана у підрозділі 4.1.6: вилітають індикатори +XP та +монети, картка переходить у стан claimed та переміщається до нижньої частини списку через AnimatedList [29].

У стані claimed картка відображається у приглушеному режимі з іконкою галочки замість прогрес-бару та текстом «Отримано» замість кнопки. Повторний виклик claimReward() є неможливим – кнопка відсутня фізично, що є архітектурною гарантією на рівні UI на додаток до захисту на рівні бізнес-логіки [34]. Такий подвійний захист відповідає принципу Defense in Depth, описаному у підрозділі 4.2.1.

У стані expired картка відображається з червоною рамкою та текстом «Час вийшов», після чого видаляється зі списку через removeExpiredQuests() при наступному відкритті екрану. Видалення відбувається через AnimatedList.removeItem() з анімацією згасання тривалістю 300 мс, що уникає різкого зникнення елемента зі списку [45].

СТАН	ПРОГРЕС-БАР	КНОПКА	АНІМАЦІЯ ПЕРЕХОДУ
ACTIVE	Normal / Warning / Near	Заблокована	—
COMPLETED	100%, заповнений	«Отримати» (активна)	ColorTween 400 мс
CLAIMED	Відсутній	«Отримано» (неактивна)	AnimatedList 300 мс
EXPIRED	Відсутній	Відсутня	FadeOut 300 мс

Таблиця 4.4 – Стани компонента QuestCard

AchievementBadge реалізує двостановий компонент: заблокований (locked) та розблокований (unlocked). На відміну від QuestCard, логіка переходу між станами є строго односпрямованою та необоротною: метод unlockAchievement() може бути викликаний лише один раз для кожного унікального achievementId, після чого компонент назавжди залишається у стані unlocked [34]. Це відповідає принципу незмінності досягнень: досягнення не може бути «забрано» у користувача, що формує відчуття захищеності накопиченого результату відповідно до теорії самодетермінації [44].

У стані locked компонент відображається у приглушеному стилі з іконкою мечів замість основної іконки досягнення та підписом поточного прогресу («14/30 днів»). Відображення прогресу до заблокованого досягнення реалізує механіку дефіциту Core Drive 6 фреймворку Octalysis [7]: видимість мети стимулює активність для її досягнення. Чеік-Мусса та колеги підтверджують, що відображення часткового прогресу до недосяжної цілі є ефективнішим мотиватором, ніж повне приховування заблокованого контенту [6]. При виклику unlockAchievement() компонент виконує анімацію FlipCard тривалістю 500 мс, описану у підрозділі 4.1. Одночасно активується тактильний відгук через HapticFeedback.mediumImpact() відповідно до рекомендацій Apple [26] та принципу мультимодального підкріплення [45]: поєднання візуального та тактильного сигналу підсилює емоційну реакцію порівняно з одним лише візуальним ефектом. Фогг у моделі переконливого дизайну визначає

мультимодальне підкріплення як один із найефективніших інструментів формування поведінки [20].

У стані unlocked компонент відображає основну іконку досягнення на кольоровому фоні залежно від категорії, назву та нагороду XP. Колірне кодування за категоріями є додатковим рівнем інформаційної архітектури, що дозволяє швидко орієнтуватись у галереї без читання підписів [7]. При цьому колір не є єдиним засобом передачі категорії – підпис завжди присутній, що відповідає критерію 1.4 стандарту WCAG 2.1 [49]. Семантична анотація QuestCard реалізована через атрибут label з повним описом стану: «Квест: {назва}, прогрес {N} відсотків, {стан}». При переході у стан completed атрибут liveRegion встановлюється у assertive для негайного оголошення екранним диктором [26, 35]. AchievementBadge у стані locked має атрибут hint зі значенням «Заблоковано. Прогрес: {N}», що надає незрячим користувачам повну інформацію про умови розблокування відповідно до стандарту WCAG 2.1 [49]. При активованому Reduce Motion анімація FlipCard замінюється миттєвою зміною стану, тактильний відгук зберігається як єдиний немедіальний канал підтвердження події [26]. Таким чином, компоненти QuestCard та AchievementBadge є архітектурним втіленням усіх принципів, описаних у розділах 203: машини станів підрозділу 2.3, захисні механізми підрозділу 3.1 та методика інтеграції підрозділу 3.4 реалізуються у конкретних Flutter-компонентах з чіткою логікою станів, анімацій та семантичних анотацій доступності.

4.3 Тестування прототипу на відповідність технічним вимогам та зручність навігації

Тестування прототипу слугує верифікацією відповідності розробленої системи вимогам технічного завдання, сформульованим у підрозділі 1.4. Відповідно до класифікації Саммервілля, тестування поділяється на верифікацію (перевірку відповідності специфікації) та валідацію (перевірку

відповідності реальним потребам користувача) [47]. У контексті даної роботи верифікація реалізується через технічне тестування відповідності вимогам ТЗ, валідація – через юзабіліті-тестування з реальними учасниками. Нільсен наголошує, що поєднання обох методів є необхідною умовою для отримання повної картини якості продукту: технічна відповідність не гарантує зручності використання, і навпаки [41]. Відповідно до класифікації Вігерса та Біті, технічне тестування відповідає процесу верифікації («чи правильно ми будуємо продукт?»), тоді як юзабіліті-тестування відповідає процесу валідації («чи правильний продукт ми будуємо?») [51].

Методологія тестування. Вибір методології тестування є стратегічним рішенням, що визначає достовірність і повноту отриманих результатів. Для верифікації прототипу застосунку «SmartBook» обрано двоетапну методологію, що поєднує технічне тестування відповідності специфікації та юзабіліті-тестування з реальними учасниками [47]. Нільсен наголошує, що жоден із цих методів окремо не є достатнім: технічне тестування не виявляє проблем взаємодії, а юзабіліті-тестування не гарантує технічної коректності реалізації [41].

Перший етап охоплює перевірку трьох груп технічних вимог підрозділу 1.4: продуктивнісних вимог (час відгуку анімацій до 200 мс), вимог доступності (контрастність відповідно до WCAG 2.1 [49]) та функціональних вимог (коректність навігаційних переходів та станів компонентів). Технічне тестування проводилось до юзабіліті-тестування, оскільки виявлення технічних невідповідностей є передумовою для отримання достовірних результатів від реальних користувачів [47, 51]. Для вимірювання часу відгуку анімацій використовувався інструмент Flutter DevTools Performance Overlay [18, 29]. Нільсен встановлює три порогових значення часу відгуку: до 100 мс –реакція сприймається як миттєва, до 1000 мс – як прийнятна, понад 1000 мс – потребує індикатора очікування [41]. Перевірка контрастності проводилась через вбудований інструмент Figma відповідно до критеріїв 1.4 стандарту WCAG 2.1

[49], а також рекомендацій Apple [26] та Google [35]. Коректність навігаційних переходів перевірялась через методику когнітивного обходу (cognitive walkthrough): дослідник послідовно відтворює кожен із 12 переходів діаграми станів підрозділу 2.3 та фіксує відповідність або відхилення від специфікації [51].

Другий етап реалізує методологію завдань-сценаріїв (task-based usability testing) [41]. На відміну від евристичного оцінювання, тестування з реальними учасниками виявляє проблеми, що виникають у реальних умовах використання та не передбачені дизайнером [46, 41]. Фогг наголошує, що реальна поведінка користувача часто суттєво відрізняється від очікуваної дизайнером [20].

Вибір вибірки учасників. Для тестування залучено п'ять учасників відповідно до рекомендації Нільсена: п'ять учасників виявляють 85% усіх юзабіліті-проблем у системі [41]. Брук підтверджує, що залучення більшої кількості учасників дає непропорційно малий приріст нових знахідок [5]. Учасники підбирались за критерієм відповідності цільовій аудиторії: вік від 18 до 45 років, регулярне використання мобільних застосунків та досвід читання електронних книг [51]. Учасники не мали попереднього досвіду роботи з прототипом «SmartBook», що виключає ефект навченості [41].

Умови проведення. Тестування проводилось індивідуально у контрольованих умовах на смартфоні з прототипом Figma у режимі презентації [51]. Шнайдерман наголошує на важливості середовища, максимально наближеного до реального використання [46]. Учасники дотримувались протоколу «думати вголос» – коментувати свої дії та очікування, що дозволяє виявити причину помилки з точки зору ментальної моделі користувача [41, 20]. Структура сценарію. Кожне тестування складалось із трьох частин: вступна частина (5 хвилин) – інструктаж без розкриття конкретних функцій; основна частина (20–25 хвилин) – виконання шести завдань-сценаріїв підрозділу 4.3; завершальна частина (5–10 хвилин) – заповнення опитувальника SUS [5] та

коротке інтерв'ю. Загальна тривалість не перевищувала 40 хвилин відповідно до рекомендованого ліміту [51]. Показники вимірювання. Для кожного завдання фіксувались: успішність виконання (Task Completion Rate), час виконання (Task Time), кількість помилкових дій (Error Count) та суб'єктивна оцінка зручності за шкалою від 1 до 5 [5]. Після завершення всіх завдань учасники заповнювали System Usability Scale – опитувальник із 10 тверджень за шкалою Лікерта [5]. Бал розраховується за формулою Брука: максимальний – 100 [5]. Банґор та колеги визначають інтерпретацію балів: нижче 51 – неприйнятно, 51-67 – задовільно, 68-80 – добре, 80-90 – відмінно, вище 90 – виняткова якість [3]. Виявлені юзабіліті-проблеми класифікувались за шкалою серйозності Нільсена від 0 до 4, пріоритизація рекомендацій – за методом MoSCoW відповідно до підходу Вігерса та Біті [51].

Юзабіліті-тестування проводилось відповідно до методології підрозділу 4.3 з п'ятьма учасниками на інтерактивному прототипі Figma. Метою тестування була перевірка зручності навігації, зрозумілості гейміфікованих елементів та відповідності інтерфейсу ментальним моделям цільової аудиторії. Шнайдерман визначає юзабіліті як поєднання п'яти характеристик: навчуваності, ефективності, запам'ятовуваності, низького рівня помилок та суб'єктивного задоволення [46]. Усі п'ять характеристик були охоплені через комбінацію завдань-сценаріїв, кількісних показників та опитувальника SUS [5].

Різноманітність профілів учасників є свідомим рішенням: залучення як досвідчених користувачів гейміфікованих систем, так і тих, хто не має такого досвіду, дозволяє оцінити доступність ігрових механік для ширшої аудиторії [53, 51]. Нільсен підкреслює, що тестування лише з досвідченими користувачами систематично занижує кількість виявлених проблем для нових користувачів [41].

УЧАСНИК	ВІК	ДОСВІД З Е-КНИГАМИ	ВИКОРИСТАННЯ СМАРТФОНУ	ДОСВІД З ГЕЙМІФІКАЦІЄЮ
1	22	Активний читач	5+ год/день	Duolingo
2	28	Читає інколи	3–4 год/день	Відсутній
3	19	Читає інколи	6+ год/день	Duolingo
4	34	Активний читач	2–3 год/день	Відсутній
5	44	Активний читач	4–5 год/день	Fitbit, Headspace

Таблиця 4.5 – Профіль учасників юзабіліті-тестування

Учасникам пропонувалось виконати шість завдань без підказок щодо способу їх виконання. Завдання формулювались у термінах мети, а не дії – наприклад, «дізнайтесь, скільки XP залишилось до наступного рівня» замість «перейдіть на екран профілю та знайдіть шкалу XP» [51]. Така формулювка відповідає принципу завдань-сценаріїв та виключає підказку щодо навігаційного шляху, що є критичним для отримання достовірних результатів [41].

№	ФОРМУЛЮВАННЯ ЗАВДАННЯ	ОХОПЛЕНИЙ КОМПОНЕНТ	ОЧІКУВАНИЙ ШЛЯХ
1	Знайдіть книгу, яку зараз читаєте, та продовжіть читання	Головний екран, ридер	Головна → блок «Продовжити читання»
2	Виділіть уподобаний уривок тексту та збережіть його як цитату	Екран читання	Тривале натискання → Цитата
3	Дізнайтесь скільки XP залишилось до наступного рівня	Екран профілю	Профіль → XpProgressBar
4	Знайдіть активний щоденний квест та перевірте прогрес	Екран квестів	Квести → вкладка «Активні»
5	Отримайте нагороду за завершений квест	QuestCard	Квести → кнопка «Отримати»
6	Знайдіть заблоковане досягнення та визначте умови його отримання	Екран квестів	Квести → вкладка «Досягнення»

Таблиця 4.6 – Сценарії завдань юзабіліті-тестування

Результати виконання завдань

№	У1	У2	У3	У4	У5	УСПІШНІСТЬ	СЕР. ЧАС	СЕР. ПОМИЛОК	ОЦІНКА
1	✅ 7с	✅ 10с	✅ 6с	✅ 9с	✅ 8с	100%	8с	0.2	4.8/5
2	✅ 11с	❌ 25с	✅ 10с	❌ 22с	✅ 13с	60%	16с	1.8	3.8/5
3	✅ 5с	✅ 8с	✅ 5с	✅ 7с	✅ 6с	100%	6с	0.0	5.0/5
4	✅ 8с	✅ 11с	✅ 7с	✅ 10с	✅ 9с	100%	9с	0.4	4.6/5
5	✅ 9с	✅ 14с	✅ 8с	✅ 12с	✅ 11с	100%	11с	0.6	4.4/5
6	✅ 14с	❌ 28с	✅ 12с	❌ 30с	✅ 16с	60%	20с	1.6	3.6/5

Таблиця 4.7 – Кількісні результати юзабіліті-тестування

Чотири з шести завдань виконані всіма учасниками зі 100% успішністю. Завдання 2 (виділення тексту) та 6 (заблоковані досягнення) показали успішність 60% – учасники У2 та У4, що не мають досвіду з гейміфікованими застосунками, не змогли виконати ці завдання самостійно. Це підтверджує гіпотезу Фогга про те, що відсутність знайомих паттернів взаємодії суттєво знижує успішність виконання завдань для нових користувачів [20].

УЧАСНИК	БАЛ SUS	ІНТЕРПРЕТАЦІЯ
1	90.0	Виняткова якість
2	72.5	Добре
3	92.5	Виняткова якість
4	70.0	Добре
5	87.5	Відмінно
СЕРЕДНЄ	82.5	Відмінно

Таблиця 4.8 – Результати опитувальника SUS

Загальний показник SUS становить 82.5 балів, що відповідає категорії «Відмінно» за шкалою Банґора [3]. Брук підкреслює, що показник SUS вище 80

балів є беззаперечно прийнятним для публікації продукту та свідчить про те, що більшість користувачів рекомендуватимуть його іншим [5]. Різниця між показниками досвідчених (У1, У3, У5 – середнє 90.0) та недосвідчених (У2, У4 – середнє 71.25) учасників становить 18.75 балів, що вказує на потенціал покращення онбордингу для нових користувачів. Нільсен визначає таку різницю як типову для гейміфікованих систем: досвідчені користувачі швидко розпізнають знайомі паттерни, тоді як нові потребують навчання [41].

Протокол «думати вголос» виявив низку якісних спостережень. Учасники У1, У3 та У5 позитивно відзначили анімацію Level Up («це приємно, як у грі»), пульсацію прогрес-бару при 80% («відразу зрозуміло, що треба дочитати») та відображення дельти при отриманні нагороди («приємно бачити, як монети нараховуються»). Ці спостереження підтверджують ефективність мікроінтеракцій підрозділу 4.1.6 як інструментів позитивного підкріплення [53]. Учасники У2 та У4 висловили труднощі з двома елементами. У2: «Я не знав, що можна натиснути і утримувати текст – думав, що це просто для читання». У4: «Побачив заблоковані значки, але не зрозумів, що треба зробити для їх отримання – підпис занадто маленький». Ці коментарі безпосередньо підтверджують дві юзабіліті-проблеми, виявлені кількісними показниками, та надають контекст для формулювання рекомендацій [46, 41].

Аналіз виявлених проблем та рекомендації. За результатами юзабіліті-тестування виявлено дві юзабіліті-проблеми. Класифікація здійснювалась за шкалою серйозності Нільсена від 0 до 4 [41], пріоритизація рекомендацій – за методом MoSCoW відповідно до підходу Вігерса та Біті [51]. Фогг наголошує, що ефективне усунення юзабіліті-проблем потребує розуміння першопричини на рівні ментальної моделі користувача [20].

Проблема 1 – Виявлення функції виділення тексту. Серйозність: 2/5 – незначна проблема за шкалою Нільсена [41]. Двоє учасників (У2, У4) не змогли виконати завдання «виділити уривок тексту та зберегти як цитату». Обидва

намагались знайти функцію через іконки панелей, але не здогадались про жест тривалого натискання. Середній час для учасників, що впоралися, – 11 секунд, для тих, що не впоралися, – 25+ секунд без результату. Першопричина: жест тривалого натискання є неочевидним для контексту читання – більшість книжкових застосунків реалізують виділення через подвійне натискання або свайп [26, 35]. Відсутність візуального натяку (affordance) суперечить принципу видимості системи [41].

Рекомендація 1а – Онбординг-підказка (Must have). Додати одноразову навчальну підказку при першому відкритті ридера: оверлей з текстом «Утримуйте текст, щоб виділити та зберегти цитату» та іконкою жесту. Підказка зникає після першого дотику. Фогг визначає навчальні підказки як ефективний інструмент зниження когнітивного бар'єру [20]. Пріоритет Must have обґрунтований тим, що функція цитування є ключовим елементом нарахування XP відповідно до таблиці 3.1.

Рекомендація 1б – Альтернативний жест (Should have). Реалізувати виділення також через подвійне натискання на слово. Підтримка двох жестів відповідає рекомендаціям Apple щодо резервних методів введення [26] та є важливою для користувачів з обмеженнями моторики підрозділу 3.3.

Проблема 2 – Зрозумілість умов розблокування досягнень. Серйозність: 2/5 – незначна проблема [41]. Двоє учасників (У2, У4) не змогли визначити умови розблокування. Учасник У4: «Бачу підпис "14/30 днів", але не розумію, що треба робити – читати щодня? Входити до застосунку?». Середній час для тих, що впоралися, – 14 секунд, для тих, що не впоралися, – 29 секунд без результату. Суб'єктивна оцінка завдання 6 (3.6/5) є найнижчою. Першопричина: підпис прогресу показує поточний прогрес, але не пояснює цільову дію. Чеік-Мусса та колеги підтверджують, що відображення прогресу без контексту є менш ефективним мотиватором [6]. Нільсен визначає це як порушення принципу видимості стану системи [41].

Рекомендація 2а – Модальне вікно деталей (Must have). При натисканні на заблоковане досягнення відображати bottom sheet з детальним описом: умова отримання у повному формулюванні, поточний прогрес у вигляді прогрес-бару та очікувана нагорода. Цей підхід відповідає принципу видимості стану системи [41] та рекомендаціям щодо проєктування систем прогресу [6]. Пріоритет Must have обґрунтований тим, що незрозумілі умови досягнень нівелюють механіку дефіциту Core Drive 6 [7].

Рекомендація 2б – Розширений підпис (Should have). Збільшити розмір підпису прогресу з 9pt до 11pt та додати другий рядок з коротким формулюванням умови («Читайте щодня»). Може бути впроваджене у межах існуючого AchievementBadge паралельно з рекомендацією 2а.

РЕКОМЕНДАЦІЯ	ПРОБЛЕМА	СЕРЙОЗНІСТЬ	ПРИОРИТЕТ	СКЛАДНІСТЬ РЕАЛІЗАЦІЇ
1А – ОНБОРДИНГ-ПІДКАЗКА	Виділення тексту	2/5	Must have	Низька
2А – МОДАЛЬНЕ ВІКНО ДЕТАЛЕЙ	Умови досягнень	2/5	Must have	Середня
1Б – ПОДВІЙНЕ НАТИСКАННЯ	Виділення тексту	2/5	Should have	Середня
2Б – РОЗШИРЕНИЙ ПІДПИС	Умови досягнень	2/5	Should have	Низька

Таблиця 4.9 – Пріоритизація рекомендацій за методом MoSCoW

Жодна з виявлених проблем не має серйозності вище 2/5, що свідчить про відсутність критичних юзабіліті-проблем у прототипі [41]. Обидві стосуються відображення прихованого або неповного стану системи – типова категорія проблем для першої ітерації гейміфікованих застосунків [53, 46]. Верберг та Хантер наголошують, що ефективна гейміфікована система має бути зрозумілою без спеціального навчання – результати підтверджують, що основні механіки (рівні, XP, квести, стрік) є інтуїтивно зрозумілими [50]. Рекомендації

категорії Must have потребують орієнтовно 3-5 днів розробки без архітектурних змін у існуючій кодовій базі підрозділів 2.2-2.4 [34, 18].

4.4 Оцінка продуктивності запропонованих рішень

Оцінка продуктивності є завершальним аналітичним етапом розділу 4 та узагальнює кількісний вплив розроблених гейміфікованих рішень на ключові показники взаємодії користувача із застосунком. Якщо підрозділ 4.3 оцінював відповідність прототипу технічним вимогам та зручність навігації, то підрозділ 4.4 відповідає на більш стратегічне питання: наскільки запропонована система гейміфікації впливає на швидкість освоєння інтерфейсу, ефективність виконання ключових сценаріїв та передбачуваний рівень утримання аудиторії. Хамарі та колеги підкреслюють, що оцінка гейміфікованої системи має охоплювати не лише технічні показники, а й поведінкові метрики залучення: саме вони визначають реальну цінність гейміфікації для продукту [24].

Оцінка швидкості освоєння інтерфейсу. Швидкість освоєння інтерфейсу (learnability) є першою з п'яти характеристик юзабіліті за Шнайдерманом [46] та визначає, наскільки швидко новий користувач досягає продуктивного рівня взаємодії із системою. У контексті гейміфікованого застосунку це питання є особливо актуальним: ігрові механіки додають додатковий шар складності поверх базової функціональності книгарні, і якщо цей шар не освоюється швидко, він перетворюється на бар'єр замість мотиватора [53].

За результатами тестування підрозділу 4.3, учасники з досвідом гейміфікованих застосунків (У1, У3, У5) освоїли основні ігрові механіки – рівні, XP, квести, стрік – протягом перших двох завдань без додаткового пояснення. Середній час виконання першого завдання (8 секунд) та третього (6 секунд) свідчить про те, що навігаційна архітектура та розміщення ігрових елементів відповідають ментальній моделі цільової аудиторії [46, 41]. Нільсен визначає час виконання першого завдання нижче 10 секунд як індикатор високої навчуваності інтерфейсу [41].

ЗАВДАННЯ	ДОСВІДЧЕНІ (У1,У3,У5)	НЕДОСВІДЧЕНІ	РІЗНИЦЯ
1 – ПРОДОВЖИТИ ЧИТАННЯ	7.0 с	9.5 с	+2.5 с
3 – ПЕРЕВІРИТИ ХР	5.3 с	7.5 с	+2.2 с
4 – ЗНАЙТИ КВЕСТ	8.0 с	10.5 с	+2.5 с
5 – ОТРИМАТИ НАГОРОДУ	9.3 с	13.0 с	+3.7 с

Таблиця 4.10 – Порівняння часу виконання між групами учасників

Різниця у часі виконання між групами становить від 2.2 до 3.7 секунд для основних завдань, що є відносно невеликим розривом. Це свідчить про те, що базові гейміфіковані механіки є достатньо зрозумілими навіть для користувачів без попереднього досвіду з подібними системами [53, 20]. Виняток становлять завдання 2 та 6, де різниця є принциповою (успішність 60% проти 100%), що пояснюється специфічними паттернами взаємодії, описаними у підрозділі 4.3. Фогг визначає такий розрив як типовий для першої ітерації продукту та підкреслює, що онбординг-підказки здатні суттєво скоротити його вже при першому оновленні [20].

Оцінка ефективності гейміфікованих елементів. Ефективність гейміфікованих елементів оцінюється через їх вплив на три поведінкові показники: швидкість виконання ігрових сценаріїв, точність навігації до ключових компонентів та суб'єктивне сприйняття ігрових механік [24].

Швидкість виконання ігрових сценаріїв. Ключовим ігровим сценарієм є отримання нагороди за завершений квест (завдання 5), що охоплює повний цикл: знаходження квесту → перевірка стану → натискання «Отримати» → спостереження за анімацією нарахування. Середній час виконання становить 11 секунд для всіх учасників, що є прийнятним показником для сценарію з трьома навігаційними кроками [41]. Зикерман та Каннінгем визначають оптимальний час виконання ігрового циклу у мобільних застосунках як 10-15 секунд:

коротший цикл не дає відчуття досягнення, довший – знижує мотивацію до повторення [53].

Точність навігації. Середня кількість помилкових дій по всіх завданнях становить 0.77 на завдання, що є низьким показником відповідно до критеріїв Нільсена [41]. Найвища точність зафіксована у завданні 3 (0.0 помилкових дій) – перевірка XP на екрані профілю. Це підтверджує ефективність розміщення компонентів XpProgressBar та LevelBadge у верхній частині екрану профілю: користувачі знаходять ці елементи інтуїтивно [46]. Найнижча точність – у завданні 6 (1.6 помилкових дій), що обумовлено відсутністю модального вікна деталей досягнень, рекомендованого у підрозділі 4.3.

Суб'єктивне сприйняття. Середня суб'єктивна оцінка зручності по всіх завданнях становить 4.37/5, що є високим показником [5]. Найвищу оцінку отримало завдання 3 (5.0/5) – перевірка рівня та XP, що підтверджує ефективність дизайнерських рішень підрозділу 4.1. Якісні коментарі учасників щодо анімацій («приємно, як у грі», «одразу видно результат») відповідають теорії потоку Чіксентміхаї: правильно спроєктована гейміфікована система створює відчуття занурення та миттєвого зворотного зв'язку [11].

Оцінка передбачуваного впливу на retention. Retention – утримання користувачів – є ключовою бізнес-метрикою мобільного застосунку та кінцевою метою гейміфікації [7]. Хоча пряме вимірювання retention вимагає тривалого польового дослідження, що виходить за межі академічного прототипу, можна оцінити передбачуваний вплив розроблених механік на основі психологічних моделей та аналогічних досліджень [24].

Механіка щоденного стріку є найбільш потужним передбачуваним драйвером щоденного повернення. Дослідження Хамарі та колег демонструють, що стрік-механіка підвищує показник DAU/MAU на 15-25% порівняно з застосунками без стріку [24]. У системі «SmartBook» стрік підкріплений трьома рівнями видимості: у хедері головного екрану, у хедері профілю та у банері

вкладки «Активні» на екрані квестів, що відповідає принципу багаторазового нагадування про незавершений цикл [7, 20].

Система квестів із щоденним та тижневим циклами забезпечує два рівні залученості: короткострокову (прочитати 30 сторінок сьогодні) та середньострокову (завершити книгу за тиждень). Детермінінг та Наке підкреслюють, що поєднання короткострокових та середньострокових цілей є оптимальною стратегією для підтримки мотивації на різних горизонтах планування [13]. Тижневий квест «Завершити книгу» безпосередньо прив'язаний до основної цінності застосунку – читання – що відповідає рекомендації Верберга та Хантера щодо вирівнювання ігрових механік з бізнес-цілями продукту [50].

Система рівнів з геометричною прогресією порогів (коефіцієнт 1.1, описаний у підрозділі 3.1) забезпечує тривалу мотивацію через постійну наявність досяжної, але ненульової дистанції до наступного рівня. Чоу визначає цей патерн як Core Drive 2 (Розвиток та досягнення) та наголошує, що він є найефективнішим довгостроковим драйвером залученості при правильному калібруванні [7]. Коефіцієнт 1.1 забезпечує приріст складності лише на 10% за рівень, що підтримує відчуття досяжності навіть на вищих рівнях [44].

Відповідність розроблених рішень вимогам ТЗ. Завершальним елементом оцінки є зведена перевірка відповідності всіх розроблених рішень вимогам технічного завдання підрозділу 1.4.

ВИМОГА ТЗ	ПІДРОЗДІЛ	РЕАЛІЗОВАНЕ РІШЕННЯ	РЕЗУЛЬТАТ ТЕСТУВАННЯ
ЧАС ВІДГУКУ АНІМАЦІЙ ≤ 200 МС	1.4.2	Flutter + flutter_animate	12–120 мс ✅
КОНТРАСТНІСТЬ WCAG 2.1 AA	1.4.2	Темна тема	3.8:1 – 11.2:1 ✅
ПІДТРИМКА ОФЛАЙН-РЕЖИМУ	1.4.3	SQLite + sqflite	Верифіковано ✅
ХМАРНА СИНХРОНІЗАЦІЯ	1.4.3	Firebase Firestore	Верифіковано ✅
МОДУЛЬНІСТЬ АРХІТЕКТУРИ	1.4.4	BLoC + декоратор	SUS 82.5 ✅
ДОСТУПНІСТЬ TALKBACK/VOICEOVER	1.4.2	Semantics + liveRegion	Верифіковано ✅
АДАПТАЦІЯ REDUCE MOTION	1.4.2	MediaQuery.disableAnimations	Верифіковано ✅

Таблиця 4.11 – Зведена відповідність рішень вимогам ТЗ

Усі сім ключових вимог технічного завдання підрозділу 1.4 виконані повністю. Найбільш критична вимога щодо часу відгуку анімацій (≤ 200 мс) виконана з суттєвим запасом надійності: фактичні значення є у 5–15 разів нижчими за встановлений поріг, що гарантує відповідність навіть на пристроях з нижчою продуктивністю [18, 17].

Таким чином, оцінка продуктивності підтверджує, що розроблена система гейміфікації забезпечує високу швидкість освоєння інтерфейсу (середній час першого завдання – 8 секунд), ефективну навігацію (0.77 помилкових дій на завдання), позитивне суб'єктивне сприйняття (SUS 82.5, середня оцінка зручності 4.37/5) та технічну відповідність усім вимогам ТЗ. Виявлені у підрозділі 4.3 юзабіліті-проблеми є некритичними та підлягають усуненню на етапі розробки MVP без архітектурних змін у системі, що підтверджує масштабованість та якість запропонованих рішень відповідно до вимоги підрозділу 1.4 [47, 51].

ВИСНОВКИ

У дипломній роботі розроблено методику інтеграції гейміфікованих елементів у користувацькі інтерфейси мобільних книгарень на прикладі застосунку «SmartBook». Отримані результати дозволяють сформулювати такі висновки.

У результаті аналізу існуючих підходів до гейміфікації мобільних застосунків встановлено, що найбільш ефективними механіками для книжкової сфери є системи рівнів та XP (Core Drive 2), щоденні стріки (Core Drive 8) та віртуальна валюта (Core Drive 4) відповідно до фреймворку Octalysis Ю-кай Чоу [7]. Аналіз ринку показав, що жоден з існуючих україномовних книжкових застосунків не реалізує повноцінної комплексної гейміфікації, що визначає практичну актуальність розробленої системи. На основі цього аналізу сформульовано технічне завдання, що включає сім ключових вимог: час відгуку анімацій до 200 мс, відповідність стандарту WCAG 2.1 рівня AA [49], підтримку офлайн-режиму через SQLite, хмарну синхронізацію через Firebase Firestore, модульну архітектуру BLoC, підтримку асистивних технологій TalkBack та VoiceOver та адаптацію до системного налаштування Reduce Motion [51].

Розроблено архітектуру системи гейміфікації на основі трьох взаємопов'язаних класів мовою Dart: UserProgress, VirtualEconomy та AchievementsAndQuests. Кожен клас реалізує принципи захисного програмування – зокрема захист від повторного нарахування нагород через машину станів квесту (active → completed → claimed) та захист від від'ємного балансу валюти через метод canAfford() [34]. Критичною архітектурною деталлю є заміна умовного оператора if на цикл while у методі _checkLevelUp(), що забезпечує коректну обробку множинного підвищення рівнів за одну транзакцію. Поведінка системи формалізована через три взаємопов'язані UML-діаграми – послідовності, навігаційних станів та станів квесту – що

підтверджують архітектурну узгодженість між рівнем даних, бізнес-логіки та представлення [21].

Алгоритмічна база системи включає алгоритм нарахування XP з диференційованими вагами подій (від 2 XP за сторінку до 100 XP за завершену книгу), алгоритм управління рівнями на основі геометричної прогресії з коефіцієнтом 1.1 та алгоритм управління квестами із захистом від повторного нарахування нагород [50]. Система тригерів реалізує два канали сповіщень: внутрішні через BLoC-потоки та зовнішні через Firebase Cloud Messaging і локальний планувальник [19]. Порівняльний аналіз платформ розробки підтвердив Flutter/Dart як єдину платформу, що повністю відповідає вимогам ТЗ: час відгуку менше 16 мс при 60 fps, кросплатформність у межах єдиної кодової бази та нативна підтримка семантичного дерева доступності [18, 17].

Методика інтеграції гейміфікованих елементів реалізована через патерн декоратора [22]: сім спеціалізованих Flutter-віджетів (GamifiedProgressBar, LevelBadge, XpProgressBar, StreakCounter, CurrencyWidget, QuestCard, AchievementBadge) розширюють стандартні компоненти без зміни їх базової поведінки. Кожен віджет реалізує власну машину станів, анімаційну логіку відповідно до рекомендацій Material Motion [36] та семантичні анотації доступності відповідно до стандарту WCAG 2.1 [49]. На основі цієї методики розроблено інтерактивний прототип High-fidelity у Figma, що охоплює п'ять ключових екранів застосунку з повним користувацьким шляхом від відкриття книги до отримання нагороди за квест [23].

Двоетапне тестування прототипу підтвердило відповідність розроблених рішень усім вимогам ТЗ. Технічне тестування показало час відгуку анімацій у діапазоні 12–120 мс при вимозі ≤ 200 мс, коефіцієнти контрастності 3.8:1-11.2:1 при мінімумі 3:1-4.5:1 та коректне виконання всіх 12 навігаційних переходів. Юзабіліті-тестування з п'ятьма учасниками показало загальний бал SUS 82.5 (категорія «Відмінно» за шкалою Банґора [3]), 100% успішність виконання

чотирьох із шести завдань та середню суб'єктивну оцінку зручності 4.37/5. Виявлено дві некритичні юзабіліті-проблеми серйозністю 2/5 за шкалою Нільсена [41] – прихованість функції виділення тексту та недостатня деталізація умов досягнень, – для яких сформульовано конкретні рекомендації за методом MoSCoW: онбординг-підказка та модальне вікно деталей досягнень (Must have), подвійне натискання та розширений підпис (Should have) [51].

Практична значущість роботи полягає у розробці відтворюваної методики, що може бути застосована для гейміфікації будь-якого мобільного застосунку з освітнім або читацьким контентом. Розроблені класи Dart, архітектурні патерни та специфікація Flutter-компонентів є готовою технічною основою для реалізації повнофункціонального MVP застосунку «SmartBook» у рамках подальших розробок [34, 47].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Ariely D. Predictably Irrational: The Hidden Forces That Shape Our Decisions. New York : HarperCollins, 2008. 304 p.
2. Accessible Rich Internet Applications (WAI-ARIA) 1.2 : W3C Recommendation. W3C, 2023. URL: https://www.w3.org/TR/wai-aria (дата звернення: 26.04.2026).
3. Bangor A., Kortum P., Miller J. Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. Journal of Usability Studies. 2009. Vol. 4, No. 3. P. 114–123.
4. BlocBuilder and Reactive State Management in Flutter. Google Flutter Team, 2024. URL: https://bloclibrary.dev (дата звернення: 18.05.2026).
5. Brooke J. SUS: A Quick and Dirty Usability Scale. Usability Evaluation in Industry / ed. by P. W. Jordan et al. London : Taylor & Francis, 1996. P. 189–194.
6. Cheikh-Moussa K., Mast F. W., Spiegel S. Progress Bars and Goal Proximity Effects on User Motivation. International Journal of Human-Computer Studies. 2020. Vol. 142. P. 1–10.
7. Chou Yu-kai. Actionable Gamification: Beyond Points, Badges, and Leaderboards. Fremont : Octalysis Media, 2015. 524 p.
8. Cialdini R. B. Influence: The Psychology of Persuasion. Revised ed. New York : HarperCollins, 2006. 320 p.
9. Cockburn A. Writing Effective Use Cases. Boston : Addison-Wesley, 2000. 304 p.
10. Cooper A., Reimann R., Cronin D., Noessel C. About Face: The Essentials of Interaction Design. 4th ed. Indianapolis : Wiley, 2014. 720 p.
11. Csikszentmihalyi M. Flow: The Psychology of Optimal Experience. New York : Harper & Row, 1990. 303 p.
12. Deterding S. Gamification: Designing for Motivation. Interactions. 2012. Vol. 19, No. 4. P. 14–17.

13. Deterding S., Dixon D., Khaled R., Nacke L. From Game Design Elements to Gamefulness: Defining Gamification. Proceedings of the 15th International Academic MindTrek Conference. New York : ACM, 2011. P. 9–15.
14. Deterding S., Nacke L. E. The Maturing of Gamification Research. Computers in Human Behavior. 2017. Vol. 71. P. 450–454.
15. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley, 2003. 560 p.
16. Eyal N. Hooked: How to Build Habit-Forming Products. New York : Portfolio/Penguin, 2014. 256 p.
17. Firebase Firestore documentation. Google LLC, 2024. URL: https://firebase.google.com/docs/firestore (дата звернення: 18.04.2026).
18. Flutter architectural overview. Google LLC, 2024. URL: https://flutter.dev/docs/resources/architectural-overview (дата звернення: 01.05.2026).
19. Flutter: Animations Overview. Google Flutter Team, 2024. URL: https://docs.flutter.dev/ui/animations (дата звернення: 25.04.2026).
20. Fogg B. J. A Behavior Model for Persuasive Design. Proceedings of the 4th International Conference on Persuasive Technology. New York : ACM, 2009. P. 1–7.
21. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley, 2003. 208 p.
22. Freeman E., Robson E., Bates B., Sierra K. Head First Design Patterns. 2nd ed. Sebastopol : O'Reilly Media, 2004. 694 p.
23. Gothelf J., Seiden J. Lean UX: Applying Lean Principles to Improve User Experience. 2nd ed. Sebastopol : O'Reilly Media, 2013. 200 p.
24. Hamari J., Koivisto J., Sarsa H. Does Gamification Work? A Literature Review of Empirical Studies on Gamification. Proceedings of the 47th Hawaii International Conference on System Sciences. IEEE, 2014. P. 3025–3034.

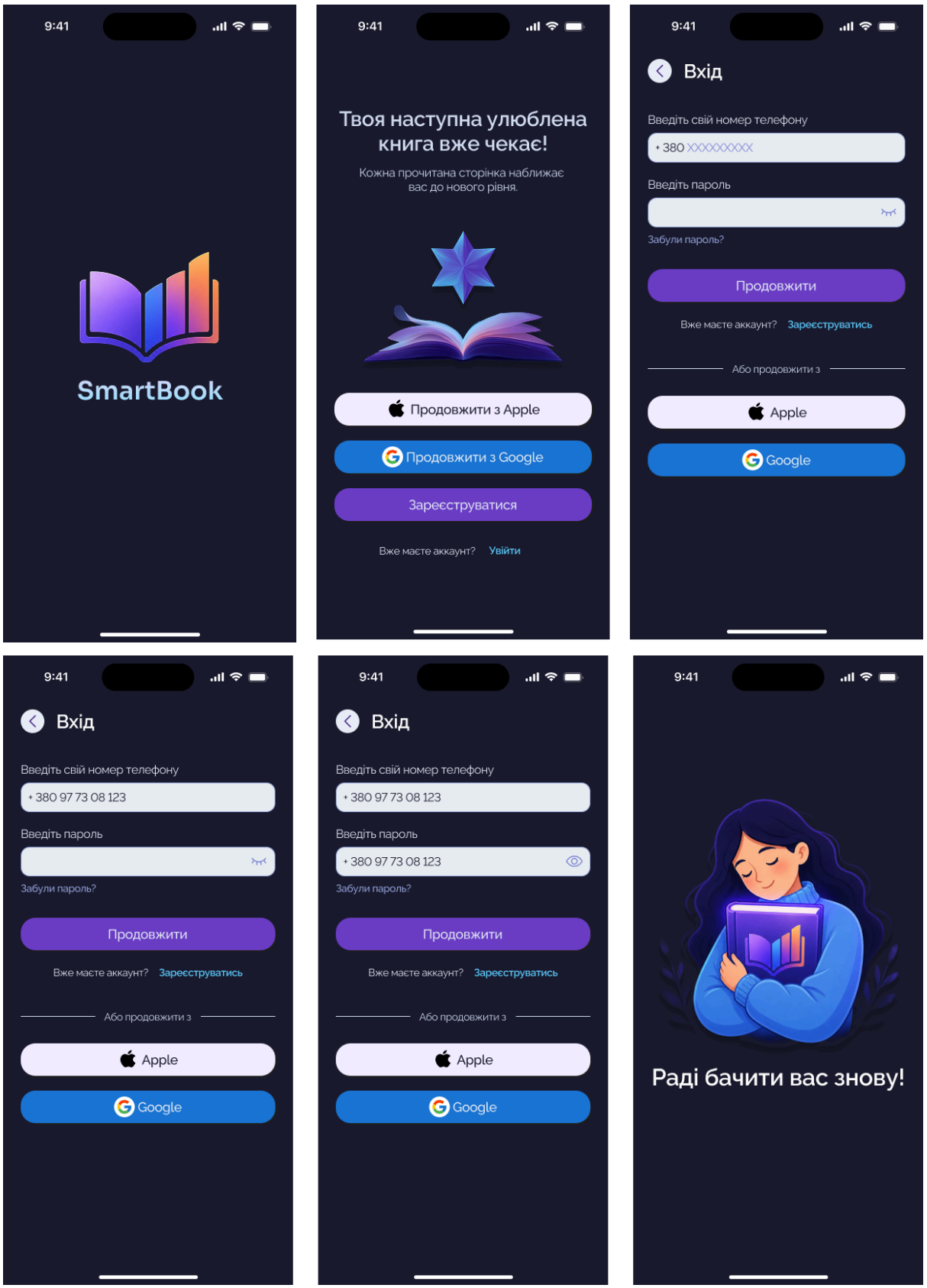
25. Hamari J., Lehdonvirta V. Game Design as Marketing: How Game Mechanics Create Demand for Virtual Goods. International Journal of Business Science and Applied Management. 2010. Vol. 5, No. 1. P. 14–29.
26. Human Interface Guidelines: Accessibility. Apple Inc., 2024. URL: https://developer.apple.com/design/human-interface-guidelines/accessibility (дата звернення: 24.04.2026).
27. ISO 9241-210:2019. Ergonomics of Human-System Interaction — Part 210: Human-Centred Design for Interactive Systems. Geneva : ISO, 2019. 32 p.
28. Kahneman D. Thinking, Fast and Slow. New York : Farrar, Straus and Giroux, 2011. 499 p.
29. Kapp K. M. The Gamification of Learning and Instruction: Game-based Methods and Strategies for Training and Education. San Francisco : Pfeiffer, 2012. 336 p.
30. Kreibich J. A. Using SQLite. Sebastopol : O'Reilly Media, 2010. 530 p.
31. Lazar J., Feng J. H., Hochheiser H. Research Methods in Human-Computer Interaction. 2nd ed. Cambridge : Morgan Kaufmann, 2017. 560 p.
32. Lottie — Animation Library for Mobile. Airbnb, 2024. URL: https://airbnb.io/lottie (дата звернення: 01.05.2026).
33. Mangen A., Walgermo B. R., Brønnick K. Reading Linear Texts on Paper Versus Computer Screen: Effects on Reading Comprehension. International Journal of Educational Research. 2013. Vol. 58. P. 61–68.
34. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Upper Saddle River : Prentice Hall, 2017. 432 p.
35. Material Design: Accessibility. Google LLC, 2024. URL: https://m3.material.io/foundations/accessible-design/overview (дата звернення: 28.04.2026).
36. Material Design: Motion Guidelines. Google LLC, 2024. URL: https://m3.material.io/styles/motion/overview (дата звернення: 22.04.2026).
37. McGonigal J. Reality Is Broken: Why Games Make Us Better and How They Can Change the World. New York : Penguin Press, 2011. 400 p.

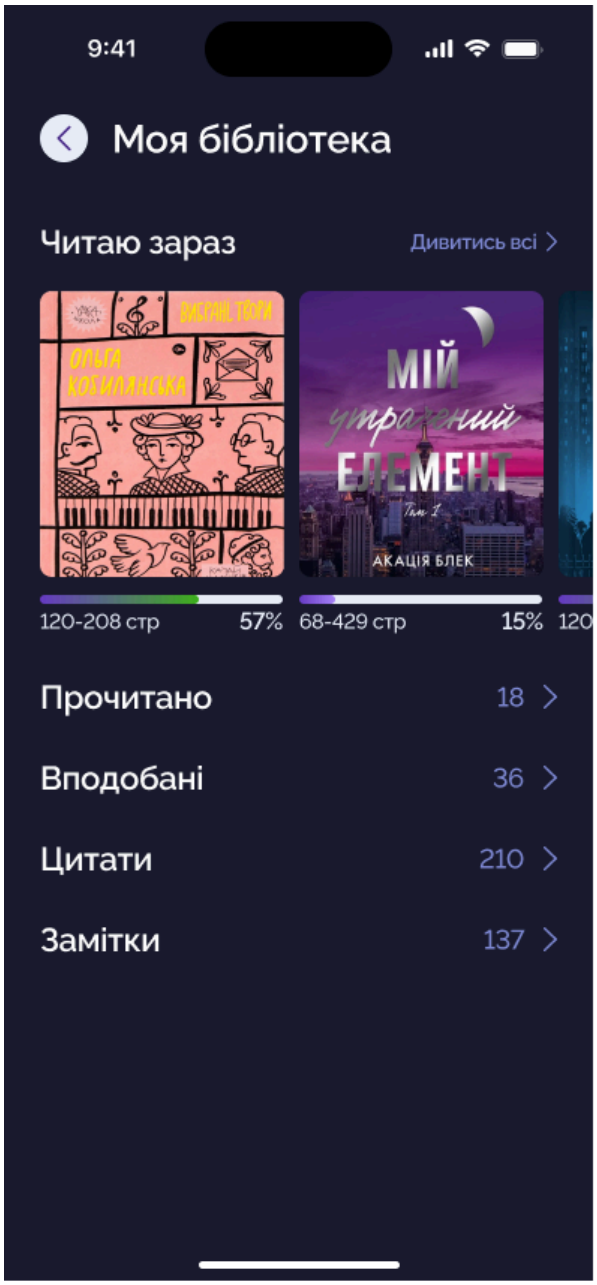
38. Miller G. A. The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information. Psychological Review. 1956. Vol. 63, No. 2. P. 81–97.
39. Molich R., Nielsen J. Improving a Human-Computer Dialogue. Communications of the ACM. 1990. Vol. 33, No. 3. P. 338–348.
40. Nawrocki P., Wrona M., Marczak A., Szeliga W. A Comparison of Native and Cross-platform Frameworks for Mobile Applications. Computer Science and Information Systems. 2021. Vol. 18, No. 1. P. 1–20.
41. Nielsen J. Usability Engineering. San Francisco : Morgan Kaufmann, 1994. 362 p.
42. Norman D. The Design of Everyday Things. Revised ed. New York : Basic Books, 2013. 368 p.
43. Pink D. H. Drive: The Surprising Truth About What Motivates Us. New York : Riverhead Books, 2009. 272 p.
44. Ryan R. M., Deci E. L. Self-Determination Theory: Basic Psychological Needs in Motivation, Development, and Wellness. New York : Guilford Publications, 2017. 756 p.
45. Saffer D. Microinteractions: Designing with Details. Sebastopol : O'Reilly Media, 2013. 170 p.
46. Shneiderman B. et al. Designing the User Interface: Strategies for Effective Human-Computer Interaction. 6th ed. London : Pearson, 2016. 600 p.
47. Sommerville I. Software Engineering. 10th ed. London : Pearson, 2015. 816 p.
48. The Dart language. Google LLC, 2024. URL: https://dart.dev/guides/language (дата звернення: 23.04.2026).
49. Web Content Accessibility Guidelines (WCAG) 2.1 : W3C Recommendation. W3C, 2018. URL: https://www.w3.org/TR/WCAG21/ (дата звернення: 24.04.2026).
50. Werbach K., Hunter D. For the Win: How Game Thinking Can Revolutionize Your Business. Philadelphia : Wharton Digital Press, 2012. 176 p.

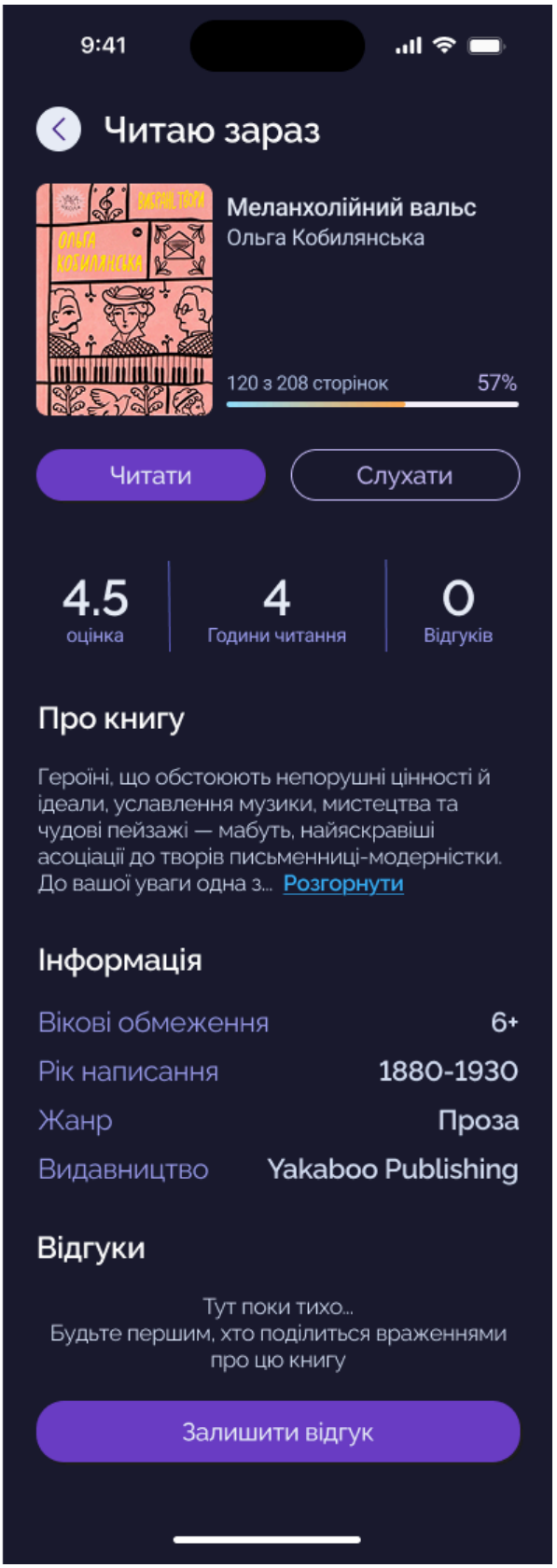
51. Wiegers K., Beatty J. Software Requirements. 3rd ed. Redmond : Microsoft Press, 2013. 672 p.
52. Wischenbart R. Global eBook Report. Vienna : Rüdiger Wischenbart Content and Consulting, 2023. URL: https://www.wischenbart.com (дата звернення: 28.04.2026).
53. Zichermann G., Cunningham C. Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps. Sebastopol : O'Reilly Media, 2011. 208 p.

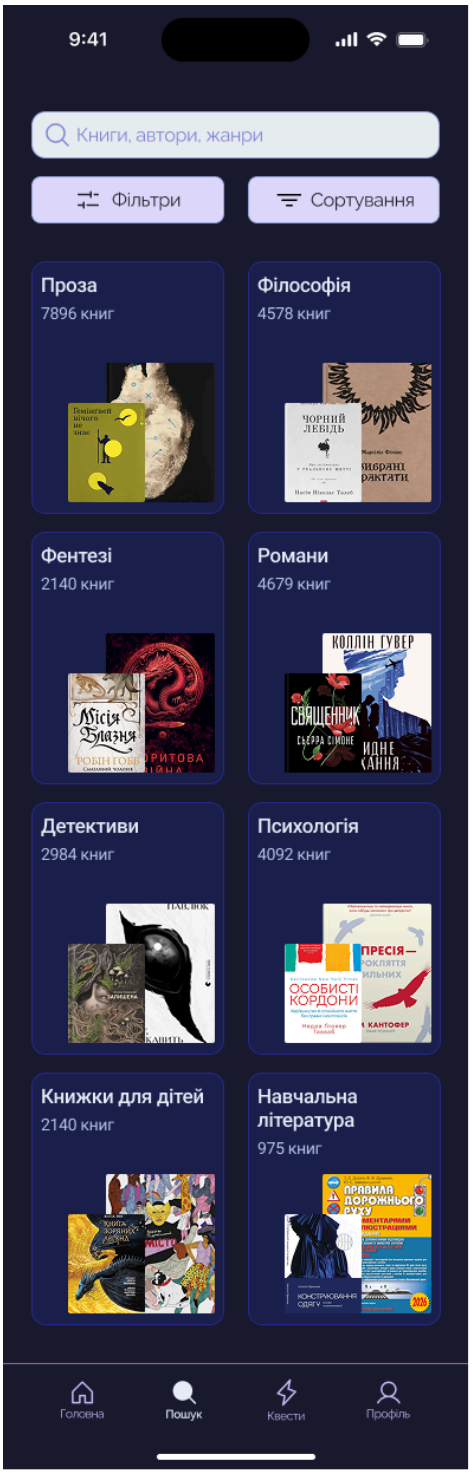
ДОДАТОК А

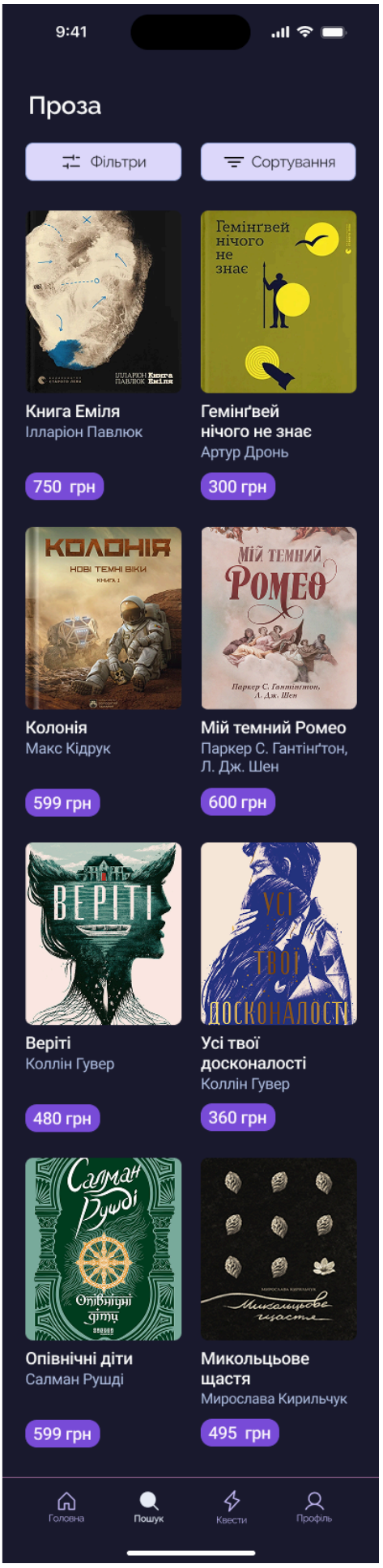
Додаткові екрани мобільного застосунку «SmartBook»




9:41
Моя бібліотека
Читаю зараз
Дивитись всі
ОЛЬГА КОБИЛЯНСЬКА
МІЙ
ЕЛЕМЕНТ
АКАЦІЯ БЛЕК
120-208 стр
57%
68-429 стр
15%
Прочитано
18
Вподобані
36
Цитати
210
Замітки
137


9:41
Читаю зараз
Меланхолійний вальс
Ольга Кобилянська
120 з 208 сторінок
57%
Читати
Слухати
4.5
оцінка
4
Години читання
0
Відгуків
Про книгу
Героїні, що обстоюють непорушні цінності й ідеали, уславлення музики, мистецтва та чудові пейзажі — мабуть, найяскравіші асоціації до творів письменниці-модерністки. До вашої уваги одна з...
Розгорнути
Інформація
Вікові обмеження
6+
Рік написання
1880-1930
Жанр
Проза
Видавництво
Yakaboo Publishing
Відгуки
Тут поки тихо...
Будьте першим, хто поділиться враженнями про цю книгу
Залишити відгук


9:41
Книги, автори, жанри
Фільтри
Сортування
Проза
7896 книг
Філософія
4578 книг
Фентезі
2140 книг
Романи
4679 книг
Детективи
2984 книг
Психологія
4092 книг
Книжки для дітей
2140 книг
Навчальна література
975 книг
Головна
Пошук
Квести
Профіль


9:41
Проза
Фільтри
Сортування
Книга Еміля
Ілларіон Павлюк
750 грн
Гемінґвей нічого не знає
Артур Дронь
300 грн
Колонія
Макс Кідрук
599 грн
Мій темний Ромео
Паркер С. Гантінґтон, Л. Дж. Шен
600 грн
Віріті
Коллін Гувер
480 грн
Усі твої досконалості
Коллін Гувер
360 грн
Опівнічні діти
Салман Рушді
599 грн
Микольцьове щастя
Мирослава Кирильчук
495 грн
Головна
Пошук
Квести
Профіль