Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп’ютерних наук

**Допущено до захисту**

Завідувач кафедри комп’ютерних наук

доктор технічних наук, професор

Андрій БОНДАРЧУК

«01» червня 2026 р.

# КВАЛІФІКАЦІЙНА РОБОТА

**на здобуття освітнього ступеня «бакалавр»**

**спеціальність 122 Комп’ютерні науки**

**освітня програма 122.00.01 Інформатика**

**Тема:** АВТОНОМНИЙ ЗАСТОСУНОК ОБРАХУНКУ ДАНИХ ДЛЯ ПРИЦІЛУ НА ОСНОВІ ВВЕДЕНИХ ПОПРАВОК

Виконав
студент групи ІНб-2-22-4.0д
Даценко Владислав
Дмитрович

_____
(підпис)

Науковий керівник
Доктор технічних наук,
професор
Бондарчук Андрій Петрович

_____
(підпис)

Київ - 2026

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

**«Затверджую»**
Завідувач кафедри комп'ютерних наук, доктор технічних наук, професор
Андрій БОНДАРЧУК
«31» жовтня 2025 р.

**ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА**
**«Автономний застосунок обрахунку даних для прицілу на основі введених поправок»**

Виконавець – студент групи ІНб-2-22-4.0д,
спеціальності 122 Комп'ютерні наук,
освітньої програми 122.00.01 Інформатика
Даценко Владислав Дмитрович

1. Вихідні дані: Законодавство України в галузі інформаційних технологій, Статути Збройних сил України, армійські настанови з високоточної стрільби, навчальні посібники та література зі снайпінгу, технічні документації зі створення мобільних застосунків, керівні документи до обраних технологій (React Native, SQLite).
2. Основним завданням кваліфікаційної роботи: опрацювати літературні джерела, нормативні акти, та керівні документи присвячені темі дослідження, проаналізувати сучасні застосунки-конкуренти; проєктування архітектури майбутнього; оформлення пояснювальної записки згідно вимог стандарту кафедри.
3. Пояснювальна записка: Обсяг – до 66 стор. Формату А4 комп'ютерного набору, згідно вимог стандарту кафедри.
4. Графічні матеріали: скріншоти застосунку, макети роботи, графіки.
5. Додатки: відсутні.
6. Термін подання роботи: «01» червня 2026 р.

Науковий керівник
д.т.н., професор
Бондарчук А.П.
_____ дата

Виконавець:
Даценко В. Д.

_____ дата

# КАЛЕНДАРНИЙ ПЛАН

| № | Етапи виконання роботи | Термін виконання | Примітка |
|---|---|---|---|
| 1 | Затвердження теми кваліфікаційної роботи | 31.10.25 | Виконано |
| 2 | Аналіз предметної області автономного застосунку обрахунку даних для прицілу | 01.11.25 – 11.01.26 | Виконано |
| 3 | Аналіз основ прикладної балістики | 26.01.26 – 15.03.26 | Виконано |
| 4 | Дослідження практичної складової високоточної стрільби | 16.03.26 – 31.03.26 | Виконано |
| 5 | Розробка застосунку для обрахунку коригувальних поправок, написання обчислювального ядра та бази даних застосунку | 01.04.26 - 15.04.26 | Виконано |
| 6 | Польове тестування розробленого застосунку | 16.04.26 – 30.04.26 | Виконано |
| 7 | Впровадження програмних корекцій на основі результатів польового тестування | 01.05.26 – 15.05.26 | Виконано |
| 8 | Підготовка пояснювальної записки, графічних матеріалів та додатків | 25.05.26 – 12.06.26 | Виконано |
| 9 | Захист кваліфікаційної роботи | 16.06.2026 | |

**«Затверджую»**

Науковий керівник

д.т.н., професор,

_____ Бондарчук А.П.

Індивідуальний план склав

_____ Даценко В.Д.

# РЕФЕРАТ

**Кваліфікаційна робота:** 66 с., 18 рис., 19 табл., 61 посилань.

**Актуальність роботи:** потреба у створенні автономних програмних засобів, здатних виконувати локальні коригувальні розрахунки в умовах обмеженого або нестабільного мережевого доступу. Для мобільних оптико-вимірювальних систем важливими є точність, швидкість, відтворюваність результатів і зменшення помилок користувача.

**Мета роботи:** підвищення точності та швидкості розрахунку коригувальних параметрів шляхом проєктування автономного застосунку для обрахунку коригувальних поправок.

**Об'єкт дослідження:** процес обробки вхідних параметрів у автономних обчислювальних системах.

**Предмет дослідження:** методи, моделі та програмні засоби розрахунку коригувальних параметрів з урахуванням умов середовища, профільних коефіцієнтів і внутрішніх правил валідації.

**Завдання дослідження:**

1. проаналізувати предметну область коригувальних розрахунків;
2. дослідити наявні методи та програмні засоби;
3. визначити функціональні й нефункціональні вимоги до майбутньої системи;
4. розробити математичну модель універсального коригувального перерахунку;
5. спроєктувати структуру даних і алгоритм роботи програмного застосунку;
6. сформувати вимоги до інтерфейсу та сценаріїв взаємодії користувача з системою.

**Основні результати:** було побудовано математичну модель для точного обрахунку коригувальних поправок, спроєктовано відповідні функціональні елементи, алгоритми роботи, розроблено інтерфейс, а також проведено опис підходів до реалізації, тестування та оцінювання якості застосунку.

**Практичне значення:** полягає в тому, що запропоновані підходи можуть бути використані під час створення автономного прикладного застосунку для польових, виробничих, дослідницьких та навчальних сценаріїв, де потрібен швидкий локальний перерахунок параметрів без залежності від мережевих сервісів. Запропонована архітектура забезпечує можливість збереження профілів, повторного використання шаблонів даних, журналювання результатів, локальної валідації та поступового масштабування функціональності без руйнування базової структури застосунку.

**Ключові слова:** автономний застосунок, коригувальні параметри, offline-first, локальне сховище, SQLite, валідація, інтерфейс, тестування, мобільна система.

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

API - програмний інтерфейс прикладної взаємодії.
Offline-first - підхід, за якого ключові функції застосунку доступні без постійного мережевого з’єднання.
SQLite - вбудована реляційна система керування базами даних.
UI - користувацький інтерфейс.
UX - користувацький досвід.
WCAG - рекомендації з доступності веб-контенту.
Валідація - перевірка коректності введених даних перед обчисленням.
Високоточна стрільба – вид стрільби, що ведеться по статичних та рухомих цілях на гранично великих дистанціях із високою точністю та купчастістю.
MRAD (MIL, міліradіан) – одиниця вимірювання кутів, 0.001 радіана, призначена для вимірювання відстаней та коригувальних поправок в снайпінгу; має метричну прив’язку.
Прицільний комплекс – система механічних та/або цифрових оптичних засобів, призначених для ведення прицільного вогню з високоточної зброї в денний чи нічний час доби.
Набій (патрон) – боєприпас для стрілецької зброї, який складається з усіх компонентів, необхідних для здійснення пострілу.
Балістичний калькулятор – пристрій або застосунок, призначений для обрахунку коригувальних поправок на основі введених в нього даних про гвинтівку, прицільний комплекс, набій, та середовище.
Калібр зброї – основна характеристика, що означає внутрішній діаметр її ствола (має чітко збігатися з діаметром кулі).
Крок нарізів (твіст) – крок в дюймах, через який куля робить повний оберт по своїй осі. Тобто запис 1:10 означає що куля робить оберт кожні 10 дюймів довжини ствола.
Фокальна площина – положення прицільної сітки всередині прицілу, що визначає її поводження при зміні кратності.
Драг-функція – математична функція, що описує зміну коефіцієнту аеродинамічного опору кулі залежно від швидкості її польоту та форми.
Грейни (грани, grain) – одиниця вимірювання маси пороху та кулі. 1 грейн = 0.06 грама.
Пресет (профіль) – в цьому контексті, заздалегідь збережена конфігурація елементу, який може бути використаний повторно.

# ЗМІСТ

# ВСТУП

У сучасних умовах цифрової трансформації прикладних технічних систем особливої ваги набуває локалізація розрахунків, від яких залежить оперативність прийняття рішень, стабільність вимірювання та відтворюваність результатів. У мобільних оптико-вимірювальних системах користувач працює не в лабораторній тиші, а в мінливому середовищі, де параметри температури, освітленості, положення сенсора, похибка введення, калібрувальні зсуви та обмеження апаратної платформи здатні змінювати кінцевий результат за лічені секунди. Саме тому традиційний підхід, за якого оператор виконує перерахунок вручну, спирається на статичні таблиці або зовнішні довідкові матеріали, вже не відповідає вимогам до швидкості, точності та надійності сучасного програмного забезпечення [1-4].

Проблема ускладнюється тим, що значна частина прикладних сценаріїв пов’язана з роботою в умовах нестабільного зв’язку або повної відсутності мережевого доступу. У такій ситуації централізовані веб-сервіси, хмарні калькулятори та зовнішні довідники втрачають практичну цінність. Натомість зростає потреба в автономних застосунках, які можуть локально зберігати вхідні дані, профілі користувача, коефіцієнти перерахунку, історію обчислень і забезпечувати повторне використання результатів без синхронізації з віддаленими серверами. Саме ідеологія offline-first, яку нині підтримують офіційні рекомендації для мобільних застосунків, розглядає локальне сховище як базову ланку надійності та безперервності роботи [11-18; 29-39].

Додаткового значення темі надає і загальна тенденція переходу від великих централізованих систем до edge- та on-device-обробки, коли критично важливі обчислення виконуються безпосередньо поблизу джерела даних. Такий підхід зменшує затримки, знижує залежність від каналу зв’язку, дає змогу краще контролювати якість даних і робить систему передбачуванішою в реальній експлуатації. Для прикладного користувача це означає дуже просту річ: результат має з’явитися швидко, бути зрозумілим і пояснюваним, а сам

інтерфейс не повинен змушувати людину боротися з програмою замість роботи із завданням [6-10; 43-46].

Актуальність теми полягає у необхідності розроблення автономного програмного засобу, який поєднує точний розрахунок коригувальних параметрів, стійкість до неповних або помилкових вхідних даних, зручний інтерфейс та незалежність від постійного доступу до мережі. Для інженерної практики це не модна забаганка, а нормальна вимога зрілої системи: якщо застосунок не працює локально, то в критичний момент він працює тільки в презентації, а не в реальному житті.

Метою роботи є підвищення точності та швидкості розрахунку коригувальних параметрів шляхом проєктування автономного застосунку для обрахунку коригувальних поправок. Для досягнення поставленої мети у роботі сформульовано такі завдання:

1. проаналізувати предметну область коригувальних розрахунків;
2. дослідити наявні методи та програмні засоби;
3. визначити функціональні й нефункціональні вимоги до майбутньої системи;
4. розробити математичну модель універсального коригувального перерахунку;
5. спроєктувати структуру даних і алгоритм роботи програмного застосунку;
6. сформувати вимоги до інтерфейсу та сценаріїв взаємодії користувача з системою.

Об’єктом дослідження є процес обробки вхідних параметрів у автономних обчислювальних системах мобільного типу. Предметом дослідження є методи, моделі та програмні засоби розрахунку коригувальних параметрів з урахуванням умов середовища, профільних коефіцієнтів і внутрішніх правил валідації. У роботі використано методи системного аналізу, порівняльного аналізу програмних рішень, математичного моделювання,

алгоритмізації, структурного проєктування даних та елементів людино-машинної взаємодії [3-10; 40-47].

Практичне значення одержаних результатів полягає в тому, що запропоновані підходи можуть бути використані під час створення автономного прикладного застосунку для польових, виробничих, дослідницьких та навчальних сценаріїв, де потрібен швидкий локальний перерахунок параметрів без залежності від мережевих сервісів. Запропонована архітектура забезпечує можливість збереження профілів, повторного використання шаблонів даних, журналювання результатів, локальної валідації та поступового масштабування функціональності без руйнування базової структури застосунку [11-18; 19-28].

# РОЗДІЛ 1
# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1 Особливості обрахунку коригувальних параметрів у мобільних оптико-вимірювальних системах

Коригувальний розрахунок у мобільній оптико-вимірювальній системі доцільно розглядати як задачу перетворення первинних вхідних даних на узгоджений набір вихідних параметрів, придатних для подальшої інтерпретації оператором або суміжним програмним модулем. На відміну від статичних довідкових систем, де значення беруться з наперед заданих таблиць, сучасний автономний застосунок має враховувати, що частина параметрів змінюється динамічно, частина вводиться користувачем вручну, а частина походить із профілю пристрою чи попереднього калібрування. Отже, обчислення ніколи не є «чистою математикою»; це завжди поєднання моделі, оцінювання невизначеності, правил перевірки даних і програмної логіки обробки помилок [1; 2; 11; 12].

У найзагальнішому вигляді формування коригувального значення можна подати як послідовність етапів: фіксація первинних параметрів; перевірка їхньої повноти та допустимості; нормалізація одиниць вимірювання; застосування коефіцієнтів перерахунку; оцінка сумарної невизначеності; формування кінцевого результату та його пояснення для користувача. Така структура цілком узгоджується з сучасними уявленнями про надійні вимірювальні процедури, де важливе не лише отримане число, а й контекст його одержання, тобто які саме припущення, обмеження та правила були використані під час обчислення [1; 2; 47].

Особливістю мобільних систем є високий вплив середовищних чинників. Якщо в стаціонарній системі багато змінних можна вважати відносно сталими, то в мобільній платформі температура, стан освітлення, положення сенсора, кутове відхилення, вібрація, похибка ручного введення або навіть часовий лаг між вимірюванням та обчисленням можуть суттєво

впливати на якість результату. Саме тому проєктування математичної моделі повинно передбачати не тільки обчислювальне ядро, а й процедури оцінювання довіри до вхідних даних. Інакше система буде швидкою, але несерйозною - а в інженерії це майже завжди означає проблеми на етапі експлуатації [1; 2; 6; 47].

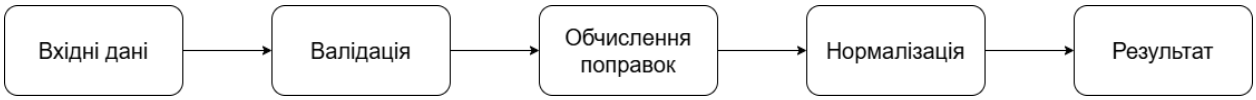


Рис. 1.1 - Узагальнена схема формування коригувальних параметрів (розроблено автором)

Таблиця 1.1

Основні фактори, що впливають на точність коригувального розрахунку

| **Фактор** | **Сутність впливу** | **Прояв у системі** |
|---|---|---|
| Похибка введення | Некоректне або неповне введення значень | Відхилення підсумкового результату |
| Умови середовища | Температура, освітленість, вібрація, часовий лаг | Зміна коректності часткових поправок |
| Стан калібрування | Зсув коефіцієнтів профілю або сенсора | Систематична помилка |
| Архітектура застосунку | Невірна організація логіки та даних | Нестійкість і слабка відтворюваність |
| Інтерфейс користувача | Нечіткі поля, помилки форм, перевантаження екрана | Підвищення кількості помилок оператора |

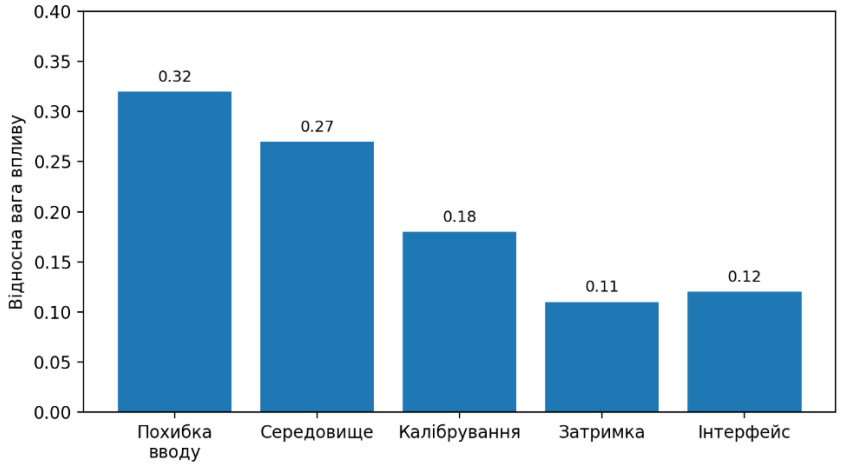


Рис. 1.2 - Умовний розподіл ваги чинників впливу на якість результату (розроблено автором)

Окрему роль відіграє людський фактор. Практика проєктування інтерфейсів показує, що значна частина помилок виникає не через складність самої формули, а через некоректне введення, непомітний формат поля, незрозумілий порядок дій або нечіткі повідомлення про винятки. За рекомендаціями з human-centred design та сучасними UX-підходами, система має мінімізувати ймовірність помилки ще до моменту розрахунку: підказувати формат даних, перевіряти межі значень, підтримувати локальний контроль одиниць вимірювання, зберігати останні сценарії вводу та надавати користувачу зворотний зв’язок зрозумілою мовою [6; 8; 43-46].

Формування корекцій доцільно розглядати не як одну монолітну формулу, а як композицію часткових поправок. Універсальна логіка тут проста: первинний параметр послідовно уточнюється з урахуванням умов середовища, профільних коефіцієнтів, індивідуальних налаштувань системи та локальної історії калібрування. Така модульність важлива не лише математично, а й програмно: вона дозволяє змінювати один механізм перерахунку без переписування всієї системи, що напряму відповідає принципам розділення відповідальності та масштабованої архітектури [11-15; 47].

Ще одна суттєва особливість полягає у потребі пояснюваності результату. Для прикладного користувача недостатньо просто побачити підсумкове значення; важливо розуміти, з яких складових воно сформоване, які поправки були застосовані автоматично, які параметри взято з профілю, а які введено вручну. Вимога до пояснюваності безпосередньо впливає на структуру даних застосунку: разом із числовими значеннями треба зберігати метадані про джерело, час, версію профілю та стан валідації. Саме на цьому етапі математична модель перетинається зі структурою сховища та журналом подій [16-18; 19-27].

Таким чином, особливості обрахунку коригувальних параметрів визначаються трьома групами чинників: математичними, інформаційними та

ергономічними. До математичних належать точність формул, стійкість алгоритму та коректне поєднання поправок; до інформаційних - повнота та надійність вхідних даних, їх локальне зберігання і відтворюваність; до ергономічних - зрозумілий інтерфейс, запобігання помилкам та можливість швидко інтерпретувати результат. Ігнорування хоча б однієї з цих груп руйнує загальну якість системи, тому подальше проєктування має здійснюватися комплексно, без вузького зведення задачі лише до написання калькулятора.

### 1.2 Аналіз існуючих методів та засобів обрахунку поправок

Історично у прикладних вимірювальних задачах використовувалися ручні методи розрахунку, коли оператор послідовно виконував перерахунки на основі формул, таблиць або графіків. Перевага такого підходу очевидна: він не потребує спеціального програмного забезпечення, легко пояснюється в навчальному процесі та може бути реалізований буквально «на папері». Проте його головним недоліком є низька стійкість до людської помилки. Навіть проста операція з округленням, переплутаними одиницями вимірювання або некоректним вибором рядка в таблиці здатна змінити кінцевий результат настільки, що він втрачає практичну цінність [1; 2; 46].

Табличні підходи стали природним етапом формалізації досвіду: вони дають змогу швидко знайти орієнтовне значення без виконання повного обчислення. Проте таблиця завжди є компромісом між зручністю та гнучкістю. Вона працює добре, коли параметри змінюються в межах передбаченого діапазону і крок дискретизації достатній. Щойно виникає потреба в більш тонкому перерахунку, додаткових коефіцієнтах або нетиповому наборі вхідних даних, таблиця перетворюється з помічника на обмеження. До того ж її важко масштабувати, підтримувати та оперативно оновлювати у польових умовах.

Електронні калькулятори й окремі мікропристрої суттєво прискорили роботу, оскільки перенесли обчислювальне навантаження з людини на технічний засіб. Їх перевагами стали відтворюваність операцій, швидке

виконання формул та менша ймовірність арифметичної помилки. Проте більшість таких рішень виявилися або надто жорстко прив’язаними до конкретного сценарію, або недостатньо прозорими для адаптації. Якщо правило перерахунку змінюється, якщо треба додати новий тип профілю чи змінити формат звіту, вузькоспеціалізований калькулятор часто не витримує цієї еволюції.

Таблиця 1.2

Порівняльна характеристика підходів до розрахунку

| **Підхід** | **Переваги** | **Недоліки** | **Загальна оцінка** |
|---|---|---|---|
| Ручний | Простота, незалежність від ПЗ | Низька швидкість, висока похибка | Базовий резервний |
| Табличний | Швидке орієнтовне визначення | Обмежена гнучкість, груба дискретизація | Допоміжний |
| Електронний | Вища швидкість та повторюваність | Слабка масштабованість сценаріїв | Локально ефективний |
| Програмний автономний | Гнучкість, журналювання, профілі, валідація | Потребує якісного проєктування | Найперспективніший |

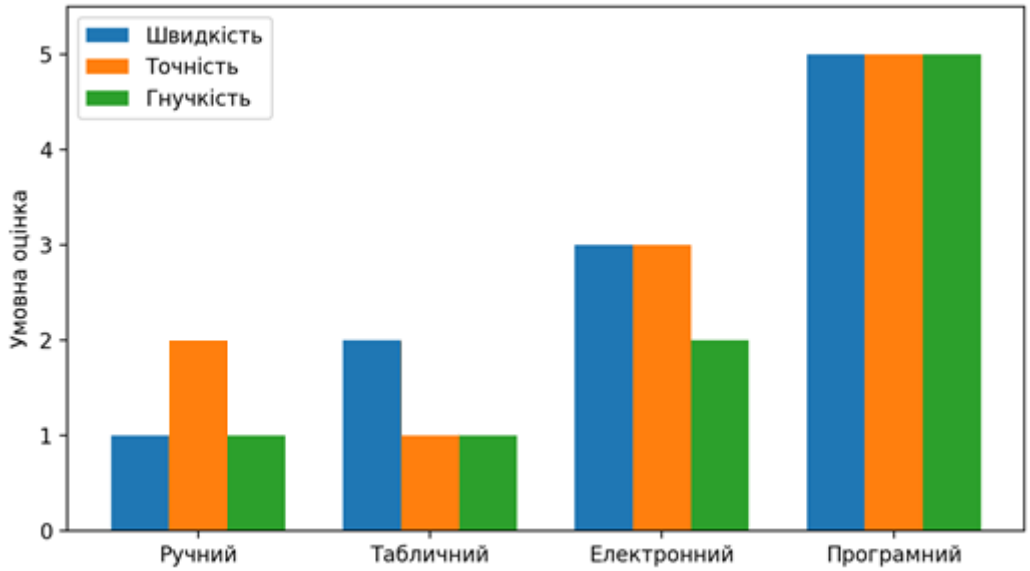


Рис. 1.3. Порівняння основних підходів за швидкістю, точністю та гнучкістю (розроблено автором)

Програмні рішення є логічним розвитком електронних засобів, оскільки поєднують гнучкість алгоритмічної моделі з можливістю накопичення й обробки даних. Саме програмний застосунок дає змогу реалізувати локальне сховище, профілі користувачів, автоматичну валідацію, журналювання операцій, модульну архітектуру і багаторівневий інтерфейс. Інакше кажучи, це вже не просто обчислювач, а робоче середовище, де результат є наслідком узгодженої взаємодії даних, логіки й інтерфейсу [3-6; 11-18].

Разом з тим програмні підходи теж мають свої ризики. По-перше, помилка в алгоритмі масштабується значно швидше, ніж ручна помилка оператора: якщо формула реалізована некоректно, система буде впевнено відтворювати хибний результат. По-друге, залежність від мережевих сервісів знижує надійність у мобільних сценаріях. По-третє, надмірно перевантажений інтерфейс здатний звести нанівець переваги автоматизації. Отже, сама наявність програмного засобу ще не гарантує якості; критично важливими стають питання архітектури, валідації та локальної стійкості [8; 11; 12; 16; 40-45].

У контексті сучасних рекомендацій особливої уваги заслуговує offline-first підхід. Він виходить із того, що ядро прикладної функціональності має бути доступним без мережі, а синхронізація із зовнішніми сервісами повинна бути лише додатковою можливістю, а не умовою працездатності системи. Така позиція добре узгоджується з локальними БД на основі SQLite або Room, з використанням транзакцій, журналювання та механізмів atomic commit/WAL, що підвищують надійність збереження результатів [16-18; 19-28].

Отже, порівняльний аналіз методів та засобів показує, що ручні й табличні способи залишаються корисними як базовий або резервний рівень, але не забезпечують потрібної гнучкості. Електронні калькулятори поліпшують швидкість, однак часто поступаються програмним системам у масштабованості. Саме автономний програмний застосунок поєднує ключові переваги: швидке обчислення, відтворюваність, накопичення історії,

варіативність профілів та можливість подальшого розвитку без радикальної зміни основної логіки.

### 1.3 Аналіз існуючих програмних застосунків для обчислення параметрів

Аналіз існуючих програмних застосунків для розрахунку параметрів у суміжних технічних доменах дає підстави виділити три умовні групи аналогів. До першої належать веб-калькулятори, що працюють у браузері та забезпечують швидкий доступ до формул і конвертацій. До другої - мобільні утиліти, орієнтовані на вузький сценарій використання. До третьої - інженерні або напівпрофесійні застосунки, де поряд із розрахунком реалізовано локальне зберігання даних, шаблони, профілі та історію сеансів. Хоч назви конкретних продуктів у цій роботі не є визначальними, функціональна логіка цих класів рішень достатньо типова для порівняльного аналізу.

Веб-калькулятори зручні тим, що не вимагають інсталяції й легко оновлюються. Проте саме ця перевага має зворотний бік: користувач залежить від браузерного середовища, мережевого доступу, стану кешу й політик безпеки платформи. Навіть за наявності service worker та локального кешування повністю автономна поведінка для складних сценаріїв досягається не завжди, а надійність збереження даних користувача в такому випадку потребує окремого проєктування [29-39].

Таблиця 1.3
Функціональне порівняння класів програмних аналогів

| **Клас рішень** | **Автономність** | **Профілі та історія** | **Пояснюваність результату** | **Типові обмеження** |
|---|---|---|---|---|
| Веб-калькулятори | Низька або часткова | Обмежено | Низька | Залежність від середовища та мережі |
| Мобільні утиліти | Середня | Частково | Середня | Вузький сценарій використання |
| Інженерні автономні застосунки | Висока | Висока | Висока | Більша складність реалізації |

Мобільні утиліти часто мають кращу ергономіку, ніж веб-калькулятори, бо адаптовані під сенсорний сценарій взаємодії. Однак у багатьох із них помітна інша проблема: функціональність зосереджена на швидкому одноразовому обчисленні, тоді як робота з профілями, історією та поясненням результату реалізована поверхово. Користувач отримує число, але не завжди отримує впевненість у його походженні. Для інженерного програмного забезпечення це слабке місце, оскільки без прозорого ланцюга формування результату знижується довіра до системи [6; 11-18; 43-46].

Найбільш зрілими виглядають інженерні застосунки, побудовані за багатошаровою архітектурою: окремо реалізовано рівень інтерфейсу, прикладну логіку, сховище даних і механізми синхронізації. Саме вони найкраще демонструють, чому якісний застосунок має бути не просто математично коректним, а архітектурно дисциплінованим. Рекомендовані підходи Android-архітектури, моделі поділу на UI/data/domain layer, використання Room як локального шару персистентності та принципи unidirectional data flow фактично формують сучасний еталон для побудови автономних обчислювальних систем [11-18].

Втім, навіть добре спроєктовані аналоги нерідко мають обмеження. Серед них можна виокремити жорстку прив’язку до конкретної моделі даних, слабку локальну валідацію, відсутність розгорнутого журналу обчислень,

недостатню доступність інтерфейсу для різних категорій користувачів, а також обмежену пояснюваність помилок введення. Рекомендації WCAG, UX-евристики та підходи OWASP показують, що сучасний застосунок повинен не тільки рахувати, а й запобігати некоректній взаємодії, коректно обробляти винятки та поважати контекст користувача [8-10; 40-45].

З огляду на це перспективним є створення застосунку, який поєднає кращі риси наявних аналогів: автономну роботу, локальне сховище, журналювання, відтворюваність сценаріїв, валідацію даних, зрозумілий інтерфейс і пояснюваний результат. У такому рішенні не повинно бути магії «натисни кнопку й повір». Навпаки, інтерфейс і архітектура мають працювати так, щоб користувач бачив логіку дій системи й міг відтворити попередній сеанс без зайвих маніпуляцій.

### 1.4 Постановка задачі розробки автономного застосунку

На основі проведеного аналізу предметної області, методів обчислення та класів програмних аналогів можна сформулювати постановку задачі розробки автономного застосунку. Майбутній програмний продукт має забезпечувати локальний розрахунок коригувальних параметрів у мобільній оптико-вимірювальній системі на основі введених користувачем значень, збережених профілів та службових коефіцієнтів. При цьому ключовими є не лише точність обчислення, а й відтворюваність, надійність збереження, стійкість до помилок введення та простота практичного використання.

Зміст задачі можна подати як сукупність взаємопов’язаних підзадач. Перша підзадача - організація коректного введення параметрів із перевіркою діапазонів, формату, повноти та одиниць вимірювання. Друга - реалізація математичного ядра, що виконує нормалізацію даних і перерахунок з урахуванням набору поправок. Третя - збереження профілів, шаблонів та історії результатів у локальному сховищі. Четверта - формування інтерфейсу, який дозволяє швидко повторити типовий сценарій і водночас дає

користувачеві пояснення результату та причини відмови у разі помилки [16-18; 19-28; 40-45].

Окремою вимогою є архітектурна стійкість майбутньої системи. Застосунок має бути побудований так, щоб зміна математичної формули або введення нового профілю не вимагали повного перепроєктування всіх модулів. Для цього доцільно відокремити рівень інтерфейсу, рівень прикладної логіки та рівень даних. Таке рішення не тільки полегшує супровід, а й робить систему придатною до подальшого тестування, розширення та аудиту якості [11-15; 47].

Отже, у межах даної роботи ставиться задача проєктування автономного застосунку, який забезпечує: локальне введення та валідацію параметрів; універсальний механізм коригувального перерахунку; роботу з профілями і журналом результатів; збереження даних у локальному надійному сховищі; зручний, зрозумілий і доступний інтерфейс користувача. Саме така постановка задачі задає рамку для наступного розділу, де розглядатимуться вимоги, математична модель, структура даних, алгоритм роботи й інтерфейс застосунку.

Таблиця 1.4

Узагальнення постановки задачі

| **Компонент задачі** | **Зміст** |
|---|---|
| Вхідні дані | Первинні параметри, профілі, коефіцієнти, умови середовища |
| Основна функція | Нормалізація, перерахунок, агрегування та видача результату |
| Супровідні функції | Валідація, журналювання, збереження історії, робота з профілями |
| Вимоги до середовища | Автономність, локальне сховище, стабільна робота без мережі |
| Очікуваний результат | Пояснюваний і відтворюваний розрахунок з мінімізацією помилок |

## Висновки до розділу 1

У першому розділі було проаналізовано предметну область розрахунку коригувальних параметрів у мобільних оптико-вимірювальних системах. Показано, що точність результату визначається не лише формулою, а й якістю введення даних, умовами середовища, правилами валідації та ергономікою інтерфейсу. Проведений огляд ручних, табличних, електронних та програмних засобів дав змогу встановити, що найбільш перспективним є автономний програмний підхід, який поєднує швидкість, гнучкість, локальне збереження та відтворюваність обчислень. На основі цього сформульовано постановку задачі розробки автономного застосунку, здатного працювати без постійного мережевого доступу та забезпечувати пояснюваний, керований і надійний результат.

# РОЗДІЛ 2

# ПРОЄКТУВАННЯ АВТОНОМНОГО ЗАСТОСУНКУ

## 2.1 Формування функціональних та нефункціональних вимог до системи

Формування вимог до автономного застосунку є критичним етапом проєктування, оскільки саме на цьому рівні визначається, що система повинна робити, у яких умовах вона має працювати та за якими критеріями надалі оцінюватиметься її якість. У сучасній практиці розробки ПЗ вимоги розглядаються не як формальна передмова до коду, а як інструмент управління складністю. Якщо вимоги нечіткі, то й архітектура буде випадковою; якщо вимоги сформульовано предметно, система отримує дисципліну вже на старті [3-7; 47].

До функціональних вимог автономного застосунку насамперед належить можливість введення первинних параметрів вручну з контролем типів даних, діапазонів значень та одиниць вимірювання. Застосунок має підтримувати сценарій швидкого одноразового розрахунку, а також сценарій повторного використання збережених профілів. Це означає, що система повинна не просто приймати дані, а й дозволяти користувачу створювати, редагувати та обирати профілі параметрів для типових ситуацій. Такий підхід знижує кількість рутинних дій та зменшує ймовірність повторного ручного введення з помилками [11-18; 40; 41].

Другою ключовою функціональною вимогою є реалізація універсального механізму обчислення, який підтримує послідовну обробку часткових поправок, проміжний перегляд обчислених величин та формування підсумкового результату. Важливо, щоб ядро розрахунку не було жорстко прив’язане до одного набору коефіцієнтів чи однієї конфігурації. Навпаки, модель має дозволяти поступове нарощування параметрів, підключення нових профілів і зміну правил нормалізації без руйнування всього застосунку.

Третьою функціональною вимогою виступає ведення журналу результатів. Кожен сеанс розрахунку повинен залишати структурований запис, що містить вхідні дані, використаний профіль, дату й час обчислення, підсумкові значення та службову інформацію про стан валідації. Такий журнал є важливим як для практичного повторного використання, так і для тестування, аудиту та аналізу поведінки системи. Без історії будь-який застосунок залишається одноразовим інструментом; з історією він стає системою підтримки рішень.

Таблиця 2.1

Функціональні вимоги до застосунку

| № | Вимога | Практичний зміст |
|---|---|---|
| 1 | Введення параметрів та робота з профілями | Ручне введення, перевірка формату, одиниць та діапазонів<br>Створення, редагування, вибір типових конфігурацій |
| 2 | Обчислювальне ядро | Послідовне застосування поправок і видача підсумку |
| 3 | Журнал результатів | Збереження історії обчислень |
| 4 | Локальна валідація | Раннє виявлення помилок і пояснення способу виправлення |
| 5 | Повторне використання сеансів | Відкриття, копіювання та шаблонізація минулих результатів |

Таблиця 2.2
Нефункціональні вимоги до системи

| Категорія | Зміст вимоги | Орієнтир оцінювання |
|---|---|---|
| Автономність | Ключові функції доступні без мережі | Локальне виконання та збереження |
| Швидкодія | Відсутність відчутної затримки для типового сценарію | Операція розрахунку виконується швидко |
| Надійність | Цілісність даних за збоїв або перезапуску | Транзакції, WAL, журналювання |
| Зручність | Зрозумілий сценарій і мінімум зайвих дій | Низька когнітивна складність |
| Підтримуваність | Можливість змінювати формули й профілі без повного переписування | Багатошарова архітектура |
| Доступність | Читабельність, контрастність, зрозумілі повідомлення | Відповідність принципам WCAG/UX |

Четвертою функціональною вимогою є локальна валідація введення. Офіційні рекомендації щодо форм, повідомлень про помилки та перевірки вхідних даних наполягають на тому, що система повинна якомога раніше виявляти некоректні значення, підказувати спосіб виправлення та не допускати «мовчазного» проходження хибних параметрів у розрахункове ядро [32-34; 40-45]. У практичному сенсі це означає перевірку обов’язкових полів, діапазонів, взаємної узгодженості параметрів та логічних обмежень між ними.

Нефункціональні вимоги визначають якість роботи системи як програмного продукту. Насамперед ідеться про автономність: застосунок повинен виконувати ключовий обсяг функціональності без активного мережевого з’єднання. Далі - швидкодія: типовий сценарій розрахунку має виконуватися практично миттєво з погляду користувача, тобто без відчутної затримки інтерфейсу. Не менш важливою є надійність локального збереження, яка досягається правильним вибором сховища, транзакційною моделлю та механізмами відновлення після збою [16-18; 19-28].

До нефункціональних характеристик також належать зручність використання, доступність, підтримуваність, переносимість і тестованість.

Модель якості ISO/IEC 25010 прямо вказує, що якісний програмний продукт оцінюється не лише за коректністю функцій, а і за здатністю бути зрозумілим, ефективним, безпечним і придатним до супроводу [3; 4]. Для автономного прикладного застосунку це особливо важливо, оскільки користувач часто діє в обмеженому часі, а отже інтерфейс повинен вести його за сценарієм, а не змушувати експериментувати всліпу.

Таким чином, система вимог до автономного застосунку охоплює не лише перелік функцій, а й правила поведінки системи в реальних умовах експлуатації. Саме з такої повної постановки - функціональної, нефункціональної та ергономічної - починається не «код заради коду», а повноцінне інженерне проєктування.

## 2.2 Розробка математичної моделі обрахунку коригувальних параметрів

Розробка математичної моделі обрахунку коригувальних параметрів у цій роботі здійснюється в універсальній, нейтральній постановці без прив’язки до конкретного прикладного домену. Такий підхід є принципово важливим, оскільки дає змогу відокремити загальну логіку коригувального перерахунку від особливостей окремої реалізації.

Нехай $X = \{x1, x2, \ldots, xn\}$ - набір первинних параметрів, що вводяться користувачем або надходять із профілю системи. Нехай $C = \{c1, c2, \ldots, cm\}$ - множина коефіцієнтів корекції, які можуть залежати від середовища, стану калібрування, конфігурації пристрою або попередньо збереженого шаблону. Тоді узагальнений результат R може бути подано як функцію $R = F(X, C, P, E)$, де P - профіль застосунку, а E - множина умов середовища. На відміну від статичної формули, така модель прямо показує багатофакторність розрахунку та дозволяє модульно змінювати окремі підсистеми без руйнування цілого.

Перший етап моделі - нормалізація даних. На цьому кроці всі вхідні величини переводяться в узгоджену систему одиниць, перевіряються на допустимість, а за потреби проходять попередню трансформацію. Наприклад,

значення, введені в різних масштабах або форматах, не можуть безпосередньо подаватися до ядра обчислення. З математичного погляду нормалізація формує відображення N: X → X*, де X* - простір стандартизованих параметрів. Без цього кроку навіть коректна формула може працювати з логічно несумісними даними.

Узагальнену модель коригувального перерахунку доцільно подати такими співвідношеннями:

$R = F(X, C, P, E)$,

$X^* = N(X)$,

$\delta i = fi(X^*, C, E)$,

$\Delta = \Sigma \delta i$,

Result = <R, Δ, U, V, M>.

Другий етап - обчислення часткових поправок. Для кожної групи чинників вводиться локальна функція $\delta i = fi(X^*, C, E)$, яка повертає окрему складову загальної корекції. Така декомпозиція дозволяє не змішувати всі впливи в одному виразі, а розглядати систему як набір взаємодіючих модулів. Сумарна корекція може визначатися або як адитивна композиція $\Delta = \Sigma \delta i$, або як послідовне перетворення базового параметра R0 за допомогою операторів T1, T2, …, Tk. Другий підхід краще відповідає реальному програмному дизайну, оскільки природно відображає кроки алгоритму.

Третій етап - оцінювання невизначеності. Згідно з рекомендаціями NIST, при роботі з вимірювальними даними важливо не лише отримати номінальне значення, а й оцінити вплив похибок та меж довіри. Для програмної системи це означає, що разом із підсумковим результатом доцільно зберігати ознаку довіри або індикатор якості обчислення [1; 2].

Четвертий етап - формування пояснюваного результату. Вихід системи має містити не лише кінцеве значення R, а і структуру, з якої воно склалося: базовий параметр, перелік застосованих корекцій, ознаку валідації та довідкову примітку щодо режиму обчислення. У математичному сенсі це означає, що система повертає не скаляр, а кортеж Result = <R, Δ, U, V, M>, де

V - статус валідації, а M - службові метадані. Така форма виходу безпосередньо впливає на структуру класів, записів бази даних та інтерфейсних подань.

Запропонована модель є достатньо загальною, щоб бути придатною для різних прикладних реалізацій, але водночас достатньо конкретною, щоб на її основі можна було спроєктувати алгоритм і схему даних. Її цінність полягає не в надмірній математичній складності, а в чіткому поділі на етапи: нормалізація, часткові поправки, агрегування, оцінка невизначеності, формування пояснюваного виходу. Саме така логіка дає системі міцний каркас і не дозволяє перетворитися на набір хаотичних формул, нашвидкуруч схованих за кнопкою «Обчислити».

Таблиця 2.3
Позначення основних елементів математичної моделі

| **Позначення** | **Зміст** |
|---|---|
| X | Множина первинних вхідних параметрів |
| C | Множина коефіцієнтів корекції |
| P | Профіль користувача або конфігурації |
| E | Множина умов середовища |
| N | Оператор нормалізації даних |
| δi | Часткова поправка |
| Δ | Сумарна корекція |
| U | Індикатор або оцінка невизначеності |
| V | Статус валідації |
| M | Службові метадані |

### 2.3 Проєктування структури даних застосунку

Проєктування структури даних автономного застосунку повинно відображати реальну логіку роботи користувача та обчислювального ядра. Якщо модель даних побудована слабо, застосунок починає накопичувати дублювання, суперечності та «тимчасові поля», які швидко стають постійними. Щоб цього уникнути, дані доцільно поділити на кілька функціональних сутностей: профілі, вхідні параметри, таблиці коефіцієнтів,

результати розрахунків, журнал подій і налаштування інтерфейсу [11-18; 19-28].

Сутність «Профіль» описує набір типових параметрів і службових властивостей, які можуть повторно використовуватися у багатьох сеансах. Профіль містить ідентифікатор, назву, опис, дату створення, дату останнього оновлення, ознаку активності та посилання на пов’язані коефіцієнти. Запровадження профільної моделі має подвійну користь: по-перше, скорочується обсяг ручного введення; по-друге, забезпечується відтворюваність сценаріїв, оскільки користувач працює не з набором випадкових полів, а зі структурованим шаблоном.

Сутність «Сеанс обчислення» фіксує конкретний факт запуску алгоритму. Вона має містити посилання на профіль, набір нормалізованих вхідних значень, службовий статус валідації, версію математичної моделі та часові атрибути. Це важливо для аудиту: навіть якщо через певний час система змінить формулу чи оновить профіль, історичний сеанс має залишитися відтворюваним. Такий підхід відповідає практикам збереження контексту в прикладних системах та полегшує тестування регресій.

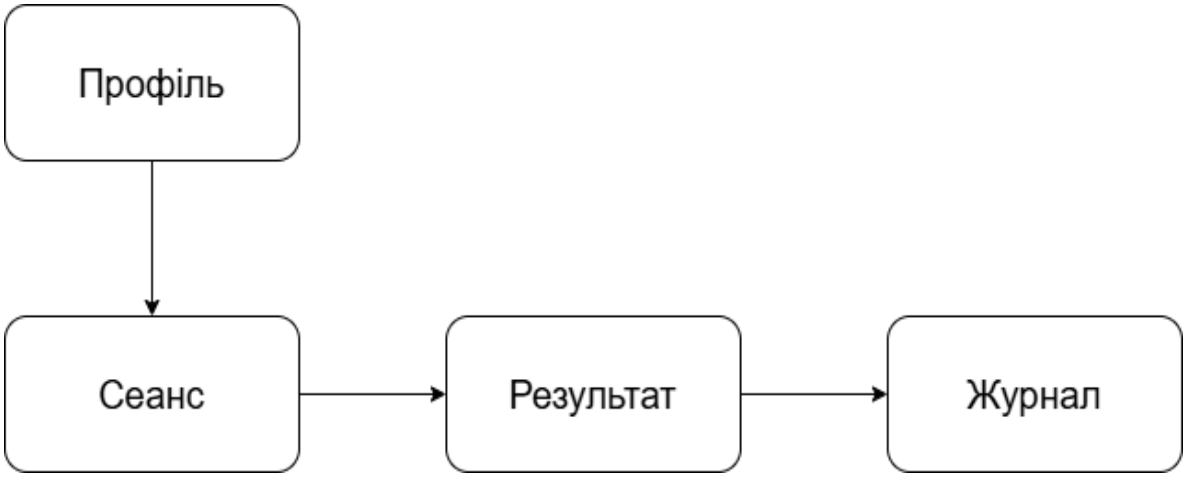


Рис. 2.1 - Логічна схема взаємозв’язків основних сутностей даних (розроблено автором)

Таблиця 2.4
Базові сутності логічної структури даних

| Сутність | Призначення | Ключові поля |
|---|---|---|
| Profile | Опис типового сценарію | id, name, description, created_at, updated_at, is_active |
| Session | Фіксація окремого запуску обчислення | id, profile_id, started_at, validation_status, model_version |
| Result | Підсумок обчислення | id, session_id, result_value, uncertainty, note |
| LogEntry | Журнал подій | id, session_id, event_type, timestamp, payload |

Для автономного застосунку принципово важливим є вибір локального сховища. Документація SQLite та пов’язаних інструментів показує, що для індивідуального мобільного застосунку локальна вбудована БД є раціональним рішенням: вона підтримує транзакції, надійне збереження, і не потребує окремого серверного шару [19-28].

З логічного погляду структура даних повинна забезпечувати баланс між нормалізацією та простотою доступу. Надмірна нормалізація ускладнює виконання типових сценаріїв читання, а надмірна денормалізація веде до дублювання і труднощів оновлення. Тому було б доцільно орієнтуватися на практичну схему: окремі таблиці профілів, сеансів, коефіцієнтів і журналу подій із чіткими зовнішніми ключами та обмеженнями цілісності. Така модель є простою для реалізації, але достатньо строгою для масштабування.

Отже, спроєктована структура даних повинна підтримувати не тільки обчислення як таке, а й життєвий цикл результату: від створення профілю та введення параметрів до повторного відкриття попереднього сеансу. Саме дані утворюють хребет системи: якщо він міцний, інтерфейс і алгоритм працюють спокійно; якщо ні - система починає ламатися в найнеприємніших місцях.

## 2.4 Проєктування алгоритму роботи програмного застосунку

Алгоритм роботи програмного застосунку повинен бути досить простим для практичного використання і водночас достатньо формалізованим для тестування. З огляду на це доцільно будувати його як послідовність детермінованих кроків, де кожен наступний етап запускається лише після успішного завершення попереднього. Така схема не тільки полегшує налагодження, а й зменшує ймовірність прихованих збоїв, коли некоректні дані просочуються в обчислювальне ядро [11-18; 40-45].

Першим кроком алгоритму є ініціалізація застосунку. На цьому етапі відкривається локальне сховище, завантажуються налаштування інтерфейсу, читається список профілів і перевіряється цілісність довідкових таблиць. Якщо сховище недоступне або конфігурація пошкоджена, система повинна не падати мовчки, а коректно інформувати користувача та переходити до безпечного режиму запуску. Це відповідає загальним принципам fault-tolerant-поведінки програмних систем.

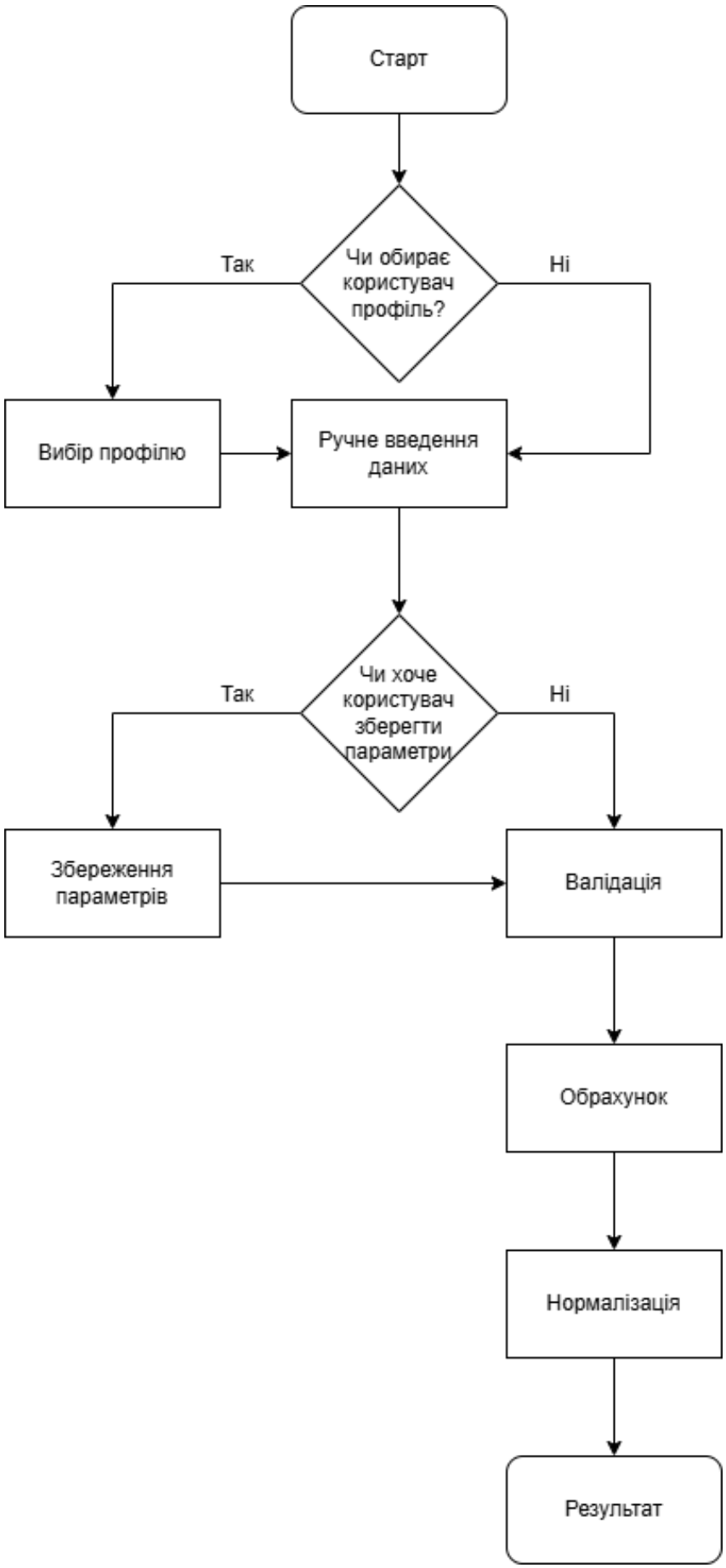


Рис. 2.2 - Узагальнена схема роботи автономного застосунку (розроблено автором)

Таблиця 2.5
Послідовність основних кроків алгоритму

| Крок | Опис дії | Результат кроку |
|---|---|---|
| 1 | Ініціалізація системи та відкриття локального сховища | Готовність конфігурації до роботи |
| 2 | Вибір профілю або створення нового сеансу | Сформований набір вихідних параметрів |
| 3 | Локальна валідація введення | Допуск або відхилення сценарію |
| 4 | Нормалізація даних | Узгоджене внутрішнє представлення параметрів |
| 5 | Виконання розрахунку | Підсумкове значення та часткові поправки |
| 6 | Збереження результату | Запис у БД та журнал подій |
| 7 | Відображення користувачу | Показ підсумку та деталізації |

Другий крок - формування вхідного набору. Користувач або вибирає один із наявних профілів, або створює новий сеанс із ручним введенням параметрів. Після цього система виконує локальну перевірку: наявність обов’язкових полів, правильність типів, межі значень, узгодженість одиниць, а також логічні обмеження між пов’язаними параметрами. Якщо перевірка не пройдена, застосунок повертає користувача до форми з конкретним повідомленням про проблему, не запускаючи ядро розрахунку [32-34; 40-45].

Третій крок - нормалізація та підготовка даних. Значення переводяться до внутрішнього стандарту представлення, з профілю підтягуються коефіцієнти, а в журнал сеансу заноситься факт старту обчислення. На цьому етапі важливо відокремити «сирі» введені дані від нормалізованих, щоб у разі аудиту або повторного відкриття сеансу можна було відновити обидва рівні представлення. Така подвійність здається зайвою лише до першої серйозної перевірки; далі вона стає рятівною.

Четвертий крок - власне виконання моделі: базове значення обробляється набором часткових поправок, після чого формується підсумковий результат, допоміжні проміжні значення та індикатор якості

обчислення. Якщо якась часткова поправка не може бути застосована через відсутність даних або конфлікт значень, система має або перейти до контрольованого спрощеного режиму, або повернути помилку, залежно від правил обраного профілю. Важливо, щоб ці правила були зафіксовані в логіці застосунку, а не виникали ситуативно.

П’ятий крок - збереження та відображення результату. Підсумковий запис разом із метаданими та статусом валідації зберігається транзакційно в локальній базі. Далі користувач бачить підсумок, коротке пояснення ланцюга обчислення і, за потреби, може перейти до розширеного перегляду з деталізацією внеску кожної поправки. Така побудова відповідає принципу progressive disclosure, коли інтерфейс не перевантажує новачка, але дає достатню глибину для досвідченого користувача [43-46].

Шостий крок - післяобробка: сеанс можна повторно використати, експортувати, зберегти як шаблон або видалити. Завдяки цьому застосунок не закінчується на одному натисканні кнопки, а вбудовується в цикл повсякденної роботи. Отже, алгоритм системи логічно зводиться до ланцюга: ініціалізація → вибір/створення профілю → введення → валідація → нормалізація → обчислення → збереження → візуалізація → повторне використання. Саме цю послідовність буде відображено у блок-схемі в межах розділу.

### 2.5 Проєктування користувацького інтерфейсу

Проєктування користувацького інтерфейсу автономного застосунку не можна зводити до «красивих кнопок». Для прикладної обчислювальної системи інтерфейс є частиною самої логіки безпеки, якості й точності, оскільки через нього проходять усі критично важливі дані. Якщо поле введення двозначне, якщо одиниця вимірювання захована, якщо повідомлення про помилку нечітке, то математично правильне ядро не врятує систему від практичної помилки користувача [6-10; 43-46].

Базова структура інтерфейсу доцільно включає чотири екрани або чотири функціональні зони: головну форму параметрів; екран профілів; екран історії розрахунків; довідковий модуль. Головна форма повинна бути побудована за сценарним принципом: користувач бачить не все одразу, а логічно згруповані блоки - вибір профілю, вхідні параметри, службові налаштування, кнопка запуску обчислення та область виведення результату. Така структура мінімізує когнітивне навантаження й відповідає рекомендаціям human-centred design [6; 43-46].


Рис. 2.3 - Концептуальна структура інтерфейсу екрану обрахунку (розроблено автором)

Таблиця 2.6
Призначення основних елементів інтерфейсу

| Елемент | Функція | Вимога до реалізації |
|---|---|---|
| Заголовок і профіль | Поточний контекст роботи | Помітність активного профілю та статусу |
| Поля параметрів | Введення значень | Підказки, одиниці, перевірка форматів |
| Кнопка обчислення | Запуск алгоритму | Блокування при невалідних даних |
| Блок результату | Відображення підсумку | Лаконічність і можливість деталізації |
| Історія | Повторне відкриття сеансів | Швидкий доступ і сортування |
| Довідка | Пояснення полів та режимів | Контекстність і зрозуміла мова |

Важливим елементом є система підказок і перевірок. Для кожного поля слід передбачити коротку назву, одиницю вимірювання, формат вводу та, за потреби, додаткову допомогу у вигляді підказки. Помилки мають відображатися поруч із відповідним елементом, а не ховатися в окремому спливаючому вікні без контексту. При цьому повідомлення повинні бути не каральними, а корисними: не «введено некоректно», а «значення має бути в межах від … до …». Це дрібниця тільки на папері; у реальному застосунку саме такі дрібниці вирішують, чи буде ним зручно користуватися [32-34; 43; 44].

Інтерфейс результату повинен поєднувати лаконічність і пояснюваність. На першому рівні користувач бачить підсумкове значення та статус: обчислено успішно, потрібна перевірка, використано профіль, спрощений режим тощо. На другому рівні - розширену деталізацію: вихідні дані, застосовані коефіцієнти, час обчислення, примітки валідації. Саме такий дворівневий показ дозволяє обслуговувати і швидкий сценарій, і більш уважну роботу з історією або перевіркою.

З позицій доступності інтерфейс повинен дотримуватися базових критеріїв контрастності, масштабованості тексту, видимості фокусу, зрозумілої послідовності навігації та сумісності з поширеними патернами

керування. Хоча йдеться не про публічний вебпортал, підходи WCAG корисні і для локальних застосунків, оскільки вони дисциплінують взаємодію та зменшують кількість прихованих бар’єрів [8-10].

Важливо також враховувати автономний характер системи. Інтерфейс повинен явно показувати, що дані зберігаються локально, профілі доступні офлайн, а історія не зникає через відсутність мережі. У цьому контексті дизайн виконує не лише естетичну, а й комунікативну функцію: він формує довіру до того, що застосунок справді автономний, а не просто тимчасово не може під’єднатися до зовнішнього сервісу.

Отже, проєктування інтерфейсу автономного застосунку слід будувати навколо сценарію користувача, а не навколо випадкового набору елементів керування. Правильний інтерфейс у такій системі - це коли користувач вводить дані, розуміє, що саме він вводить, чому система відреагувала так чи інакше, і може без зайвого клопоту повернутися до попереднього сеансу. Простота тут не означає примітивність; вона означає інженерну чесність щодо людини, яка працює з програмою.

**Висновки до розділу 2**

У другому розділі сформовано вимоги до автономного застосунку, розроблено універсальну математичну модель коригувального перерахунку, спроєктовано логічну структуру даних, алгоритм роботи та засади побудови користувацького інтерфейсу. Встановлено, що якісне автономне рішення повинно спиратися на багатошарову архітектуру, локальне надійне сховище, контрольовану валідацію введення та пояснюваний механізм формування результату. Запропоновані проєктні рішення створюють достатній фундамент для подальшої програмної реалізації, тестування та оцінювання ефективності розробленого застосунку.

# РОЗДІЛ 3.

# РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

## 3.1 Вибір засобів та середовища розробки

Практична частина роботи пов’язана з розробкою автономного мобільного застосунку Precisia, призначеного для локального обчислення коригувальних параметрів на основі набору введених характеристик. На відміну від вебкалькуляторів, які залежать від браузера, стабільності мережі та доступності зовнішнього сервера, мобільний застосунок має працювати безпосередньо на пристрої користувача. Саме тому під час вибору засобів реалізації головними критеріями були автономність, переносимість, швидкодія, зрозумілість структури коду та можливість подальшого супроводу.

Вибір засобів для розробки застосунку з точки зору поставлених вимог та задач майже одразу звужує коло пошуків до Expo/React Native. Даний вибір є достатньо обґрунтованим, оскільки React Native дає змогу створювати мобільний інтерфейс за допомогою компонентної моделі, а Expo спрощує запуск, тестування та збирання застосунку без необхідності одразу занурюватися у складну нативну конфігурацію Android-проєкту.

У структурі застосунку буде одразу викоремлено наступні модулі:

- Головне меню
- Калькулятор (UI-частина)
- Пресети
- Результати
- Налаштування
- Обчислювальне ядро

Така організація обумовлена розподілом системи на інтерфейсні екрани та окреме службове обчислювальне ядро застосунку.

Для реалізації інтерфейсу було використано мобільну компонентну модель. Її перевага полягає в тому, що кожен екран можна сприймати як

окремий функціональний блок: головне меню відповідає за навігацію, калькулятор - за збирання вхідних параметрів, модуль результатів - за відображення підсумку, пресети - за повторне використання конфігурацій, а налаштування - за зміну допоміжних параметрів. Такий підхід суттєво полегшує підтримку застосунку, тому що зміни локалізуються в конкретному модулі. Вибір React Native також обґрунтовано доцільний через можливість адаптації інтерфейсу до різних розмірів екранів. Інтерфейс може бути побудовано у вигляді послідовної блочної структури, що для мобільного застосунку є логічним рішенням.

У межах розробки важливим є також вибір підходу до стилізації. Планується централізована система стилізації з винесенням кольорів, відступів та типових компонентів в окремий тематичний модуль. Це дозволить уникнути дублювання стилів і швидко вносити глобальні зміни (наприклад, зміну кольорової схеми чи розмірів елементів).

Колірна палітра застосунку витримана у спокійних бежево-зелених тонах. В якості джерела колірної бази було взято бойовий камуфляж Збройних сил, що стоїть на озброєнні з 2014-го року, славнозвісний «Малюнок маскувальний зр. 2014» (ММ-14). Вибір мав як практичну, так і символічну сторону: з одного боку, вона не перевантажує очі, з іншого - має достатньо виражений прикладний характер, одразу окреслює призначення застосунку, та додає символічної значущості.

Таким чином, вибір Expo/React Native, TypeScript-орієнтованої структури та модульної організації коду відповідає задачі створення автономного мобільного застосунку. Технологічний стек забезпечує достатній баланс між швидкістю реалізації, якістю інтерфейсу, переносимістю та можливістю подальшого розвитку.

Таблиця 3.1

Обґрунтування вибору засобів реалізації програмного застосунку

| **Компонент** | **Обране рішення** | **Обґрунтування** |
|---|---|---|
| Платформа | Expo | спрощує запуск, тестування та збирання мобільного застосунку |
| Фреймворк | React Native | дає змогу створити мобільний UI з компонентною структурою |
| Мова/типізація | TypeScript / TSX | зменшує ризик помилок типів у числових параметрах |
| Архітектура | модулі app/(tabs) та utils | розділяє екрани інтерфейсу й обчислювальну логіку |
| Стилізація | theme/styles.tsx, constants/theme.ts | централізує оформлення та спрощує супровід |
| Режим роботи | локальне виконання | не потребує постійного мережевого підключення |

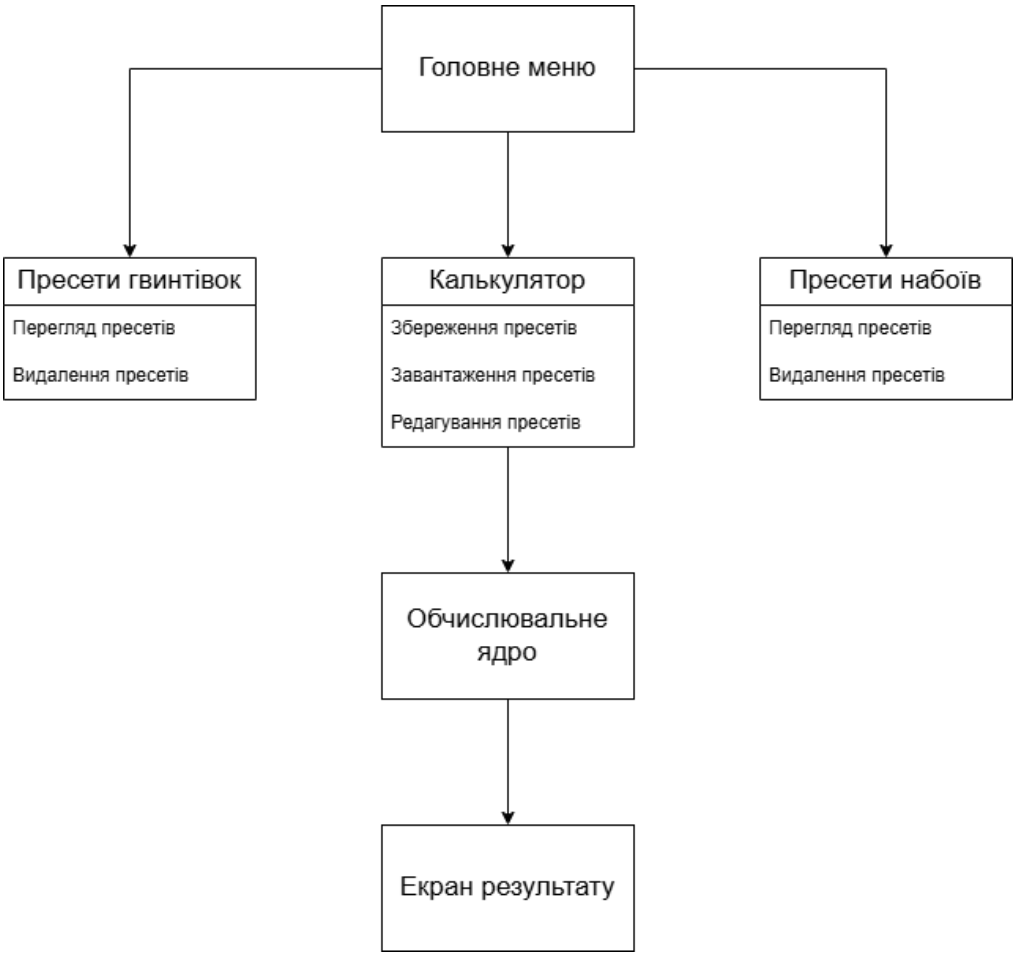


Рис. 3.1 - Логічна архітектура застосунку Precisia (розроблено автором)

Таблиця 3.2
Порівняння можливих технологічних підходів до реалізації

| Підхід | Переваги | Недоліки | Висновок |
|---|---|---|---|
| Нативний Android | максимальний контроль над платформою | більший обсяг коду, складніший старт | доцільно для великого промислового продукту |
| React Native | швидка розробка, компонентність, кросплатформеність | потребує дисципліни у структурі стану | оптимальний варіант для прототипу й розвитку |
| Вебзастосунок | просте оновлення, доступ через браузер | залежність від браузера та мережі | менш придатний для автономного сценарію |
| Desktop-рішення | зручність тестування на ПК | слабка відповідність мобільному сценарію | може бути додатковою версією, але не основною |

### 3.2 Реалізація алгоритмів обчислення в програмному коді

Головною функціональною частиною застосунку є обчислювальний модуль, який перетворює введені користувачем параметри на підсумкові коригувальні значення. Цей модуль розглядається як програмна реалізація математичної моделі, описаної у другому розділі. Його завдання полягає не в тому, щоб замінити спеціалізовані сертифіковані системи, а в тому, щоб показати принцип побудови автономного калькулятора з локальною логікою, контрольованим введенням та зрозумілим виведенням результату.

Зі структури випливає, що для розрахункової логіки передбачено окремий, не пов'язаний із UI-інтерфейсом збору параметрів, файл utils/calc.ts. Такий поділ є принципово правильним: екран calculator.tsx має відповідати за збір параметрів, модальні вікна та взаємодію з користувачем, тоді як функції перерахунку повинні бути ізольовані у обчислювальному ядрі. Це спрощує тестування, тому що функцію обчислення можна викликати окремо, без запуску повного інтерфейсу.

Узагальнено алгоритм роботи можна описати як послідовність п’яти етапів: отримання поточного стану параметрів, перевірка наявності необхідних значень, приведення параметрів до єдиного внутрішнього формату, виконання розрахунку та повернення результату у форматі, придатному для відображення на екрані results. Така схема відповідає класичному підходу до побудови прикладних обчислювальних модулів: окремо дані, окремо перевірка, окремо перерахунок, окремо презентація результату. Обчислення буде подано у вигляді двох величин: вертикального та горизонтального коригування у MRAD. Це означає, що застосунок формує результат не як абстрактне число, а як пару параметрів, які мають окреме смислове призначення.

Окремою частиною логіки є робота з параметрами набою. На екранах буде реалізовано поля “Драг-функція”, “Калібр”, “Старт. швидкість”, “Коефіцієнт G7”, “Довжина кулі”, “Маса”. Частина з них матимуть можливість редагування, а частина буде зафіксована піктограмою замка. З погляду програмної моделі це означає, що параметри поділяються на змінні та фіксовані. Така диференціація є важливою для валідації: користувач не повинен випадково змінити значення, яке належить до пресету або визначено логікою конфігурації.

Налаштування гвинтівки та прицілу також матимуть окремі групи параметрів. Наприклад, екран містить значення калібру, довжини ствола, кроку нарізів, напряму нарізів, висоти оптичного прицілу, бокового виносу, кратності та типу кліка. Ці поля доцільно трактувати як конфігураційні параметри профілю. Тобто вони не повинні вводитися щоразу вручну. Раціональніше зберігати їх у пресеті та використовувати повторно.

Погодні налаштування мають містити швидкість і напрям вітру, відстань до цілі, температуру, вологість та тиск. Для програмного застосунку важливо не лише прийняти ці значення, а й перевірити їх формат, одиниці вимірювання та допустимі межі. Наприклад, порожній рядок, текст замість числа або надмірно велике значення мають бути відхилені до запуску перерахунку.

У реалізації обчислювального ядра доцільно використовувати принцип чистої функції. Це означає, що функція перерахунку має отримувати вхідний об’єкт і повертати результат, не змінюючи глобальний стан застосунку. Такий підхід робить поведінку передбачуваною. Якщо на вхід подано однакові параметри, функція повинна повернути однаковий результат. Саме ця властивість є основою повторюваного тестування. Формат результату повинен містити не тільки числові значення, а й службову інформацію: статус успішності, повідомлення про помилку, список використаних припущень і, за потреби, ознаку спрощеного режиму. Якщо система повертає лише число, то користувач не бачить, чи всі параметри були враховані. Якщо ж результат містить контекст, застосунок стає прозорішим. Під час реалізації важливо уникати прихованих перетворень одиниць вимірювання. Якщо користувач бачить сантиметри, метри, градуси, гектопаскалі або MRAD, то внутрішня модель повинна чітко знати, у яких одиницях зберігається кожне значення. Найкращий варіант - на етапі нормалізації переводити всі параметри до внутрішнього стандарту, а при відображенні назад форматувати їх у зручному для користувача вигляді. Це зменшує кількість помилок і спрощує подальше тестування.

Застосунок матиме кілька типів редагування: перемикач драг-функції, числовий слайдер дистанції, рядкові поля параметрів, кнопки переходу та кнопка запуску розрахунку. Кожен тип введення має власну модель перевірки. Для перемикача достатньо обмежити вибір дозволеними варіантами. Для числових полів потрібні межі, крок і перевірка формату. Для заблокованих полів важливо гарантувати, що вони не змінюються випадково через інтерфейс. Для мобільного застосунку важливим є також керування станом. Дані, які користувач бачить на екрані, мають бути узгоджені з даними, що передаються в обчислювальний модуль. Якщо користувач змінив драг-функцію в модальному вікні, стан основного екрана повинен оновитися одразу після закриття діалогу. Якщо цього не зробити, на екрані може відображатися одне значення, а в розрахунок піде інше. Під час тестування особливу увагу

потрібно приділяти граничним значенням: нульова дистанція, від’ємні значення, дуже великі числа, порожні поля, зміна одиниць вимірювання, перемикання між G1 та G7, повторне відкриття модального вікна, запуск розрахунку після зміни пресету. Саме на таких сценаріях найчастіше проявляються помилки стану або неповна валідація.

Узагальнюючи, реалізація алгоритмів обчислення в коді повинна базуватися на чотирьох правилах: структуроване введення, рання валідація, чиста функція перерахунку та пояснюваний результат. Якщо ці правила виконуються, застосунок стає не лише візуально зручним, а й технічно контрольованим.

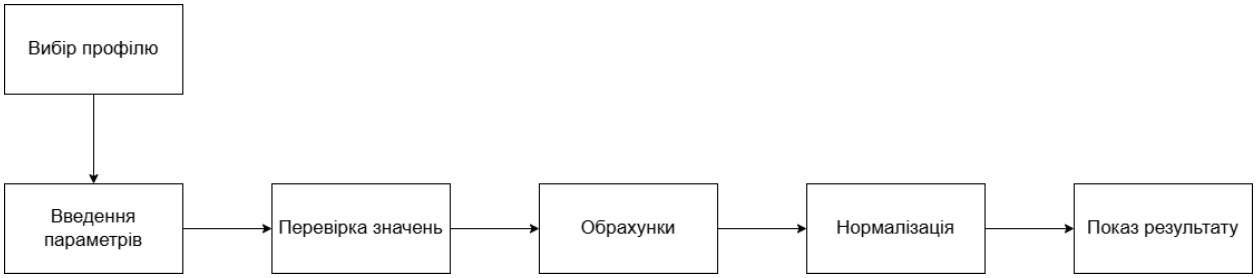


Рис. 3.2 - Узагальнений алгоритм обробки одного розрахункового сеансу (розроблено автором)

Таблиця 3.3

Основні групи параметрів, що використовуються у застосунку

| **Група параметрів** | **Приклади полів з інтерфейсу** | **Призначення у програмній моделі** |
|---|---|---|
| Параметри профілю | калібр, довжина ствола, крок нарізів | опис базової конфігурації |
| Параметри набою | драг-функція, стартова швидкість, маса | опис обраного пресету |
| Параметри прицілу | висота, боковий винос, клік | налаштування способу виведення результату |
| Параметри середовища | температура, тиск, вологість, вітер | умови для навчальної моделі |
| Результат | vertical, horizontal, unit | вихідні значення для екрана результатів |

### 3.3 Реалізація програмного інтерфейсу застосунку

Інтерфейс застосунку Precisia є однією з найсильніших частин практичної реалізації застосунку. Його структура побудована навколо зрозумілих функціональних блоків: головне меню, налаштування гвинтівки, налаштування прицілу, налаштування набою, налаштування погоди, модальні вікна редагування та екран результату. Основний замисел полягає в тому, щоб досягти фінальної точки за якомога менше кліків, слайдів і переходів між екранами. Така композиція відповідає принципу сценарної взаємодії: користувач не шукає потрібне поле в хаосі, а рухається від загальної конфігурації до конкретного обчислення.

Головний екран містить назву застосунку, та короткий мотиваційний напис, присвячений ідейному натхненнику цієї роботи, майору Збройних сил України, Олексію «Рекруту» Чубашеву, що загинув 10 червня 2022 року у боях за Сєвєродонецьк. Також на екрані наявні чотири основні кнопки: “Балістичний калькулятор”, “Пресети гвинтівок”, “Пресети набоїв”, “Налаштування”. З погляду UX це вдале рішення, оскільки користувач одразу бачить три головні сценарії: виконати розрахунок, працювати з готовими конфігураціями або змінити системні параметри. Немає перевантаження, немає зайвих меню, немає дрібного тексту. Також це вкладається в загальну логіку – «в потрібну точку за мінімум дій».

Як вже було згадано раніше, колірна палітра була взята з українського загальноармійського камуфляжа «ММ-14». Це рішення також має певний символічний ефект: висловлює повагу та захоплення українським піхотинцем, його вчинками та героїзмом. З іншого боку, це також має і практичну складову: застосунок усім своїм виглядом показує мету свого існування та сферу пріоритетного застосування.

Екран вводу даних побудований у вигляді великих карток. Кожна картка має ілюстративний силует, заголовок, і перелік параметрів. Такий патерн повторюється для гвинтівки, прицілу, набою та погоди. Повторюваність є важливою: після першого блоку користувач уже розуміє, як читати наступні.

Це класичний принцип консистентності інтерфейсу. Піктограми замка та олівця навпроти параметрів виконують важливу функцію. За їх допомогою користувач може візуально орієнтуватися в програмі та визначити, які з полів є редагованими (піктограма «Олівець»), а які – строго зафіксованими (піктограма «Замок»). Це дозволяє не пояснювати кожне поле окремим реченням та не створювати контекстні підказки на кожному кроці. Візуальний знак працює швидше за текст. Модальні вікна редагування реалізують локальну зміну параметра без переходу на інший екран. Наприклад, вибір драг-функції здійснюється через діалог із перемикачем G1/G7, а зміна напрямку вітру - через обертове колесо. Це правильний мобільний патерн: користувач не втрачає контекст основного екрана, а лише тимчасово відкриває мале вікно для конкретної дії. Після закриття модального вікна він повертається на той самий екран, на якому вже відображається змінене значення. Слайдер для визначення напрямку вітру має перевагу в швидкості введення, але має й обмеження. Якщо користувачу потрібне дуже точне значення, слайдер може бути менш зручним за текстове поле. Тому для зручності використання застосунку передбачено комбінований режим: швидке перетягування та можливість ручного введення числа.

Екран результату є максимально інформативним та наочним: на ньому виводяться вертикальна та горизонтальна поправки, обраховані у MRAD, а також зображення прицільної сітки TMR, на якій користувачеві фізично буде продемонстровано відхилення кулі від центру прицілу під впливом навколишнього середовища, та йому буде краще зрозуміло, яким виносом необхідно цілитись в мішень, якщо він не використовує барабани поправок на самому прицілі.

У застосунку також простежується ідея пресетів. Галерея пресетів у головному меню означає, що конфігурації можна зберігати або повторно використовувати. Це особливо важливо для автономних застосунків, де повторення типових сценаріїв має бути швидким. Пресет зменшує кількість ручного введення та знижує ризик помилки. Пресети за своєю структурою

були розбиті на 2 окремих екрани: пресети з гвинтівками, та пресети з кулями (на даному етапі передбачено використання лише одного прицільного комплексу). Дане рішення було прийнято з огляду на обрану раніше стратегію та спрощення навігації по застосунку. Користувачеві не потрібно проходити шлях «Пресети» - «Пресети гвинтівок/набоїв» для того, аби отримати галерею. Достатньо просто натиснути ту чи іншу кнопку і от ви тут. Візуально кожен пресет оформлено у стилі, схожому на картки на екрані обчислень. Подібна структура створює єдиний візуально впізнаваний контур та спрощує орієнтування в застосунку. Користувачеві одразу зрозуміло де і які дані шукати, адже вони мають ідентичний вигляд та розташування.

З огляду на структуру застосунку, основними інтерфейсними модулями є mainMenu.tsx, calculator.tsx, riflePresets.tsx, ammoPresets.tsx, results.tsx та settings.tsx. Їх можна описати як окремі компоненти навігаційної системи. mainMenu відповідає за початковий вибір сценарію, calculator - за роботу з параметрами, кожен з файлів presets - за збережені набори, results - за показ результату, settings - за загальні налаштування. Такий поділ добре читається і в коді та не перевантажує інтерфейс. Функціонально інтерфейс підтримує три типи дій: навігаційні, редакційні та обчислювальні. Навігаційні дії переводять користувача між екранами. Редакційні дії змінюють значення параметрів. Обчислювальна дія запускає локальний перерахунок. Саме така класифікація корисна для тестування: для кожної групи дій можна створити окремі сценарії перевірки.

Загалом інтерфейс можна оцінити як зрозумілий і логічний. Його найбільші переваги - проста навігація, великі елементи керування, повторювана структура карток, зрозумілі піктограми та чітке і зрозуміле відображення результату.

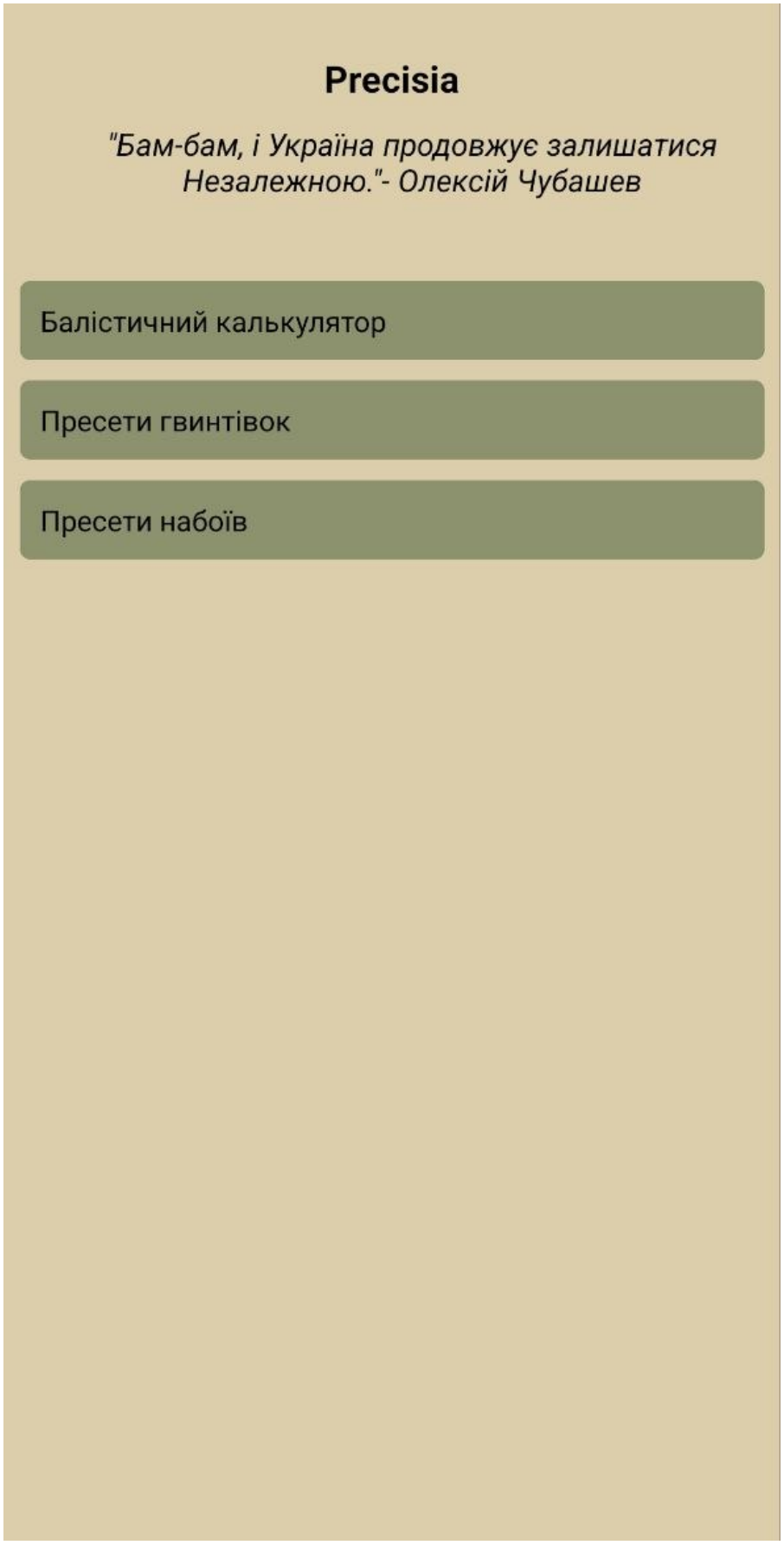


Рис. 3.3 - Головне меню застосунку Precisia

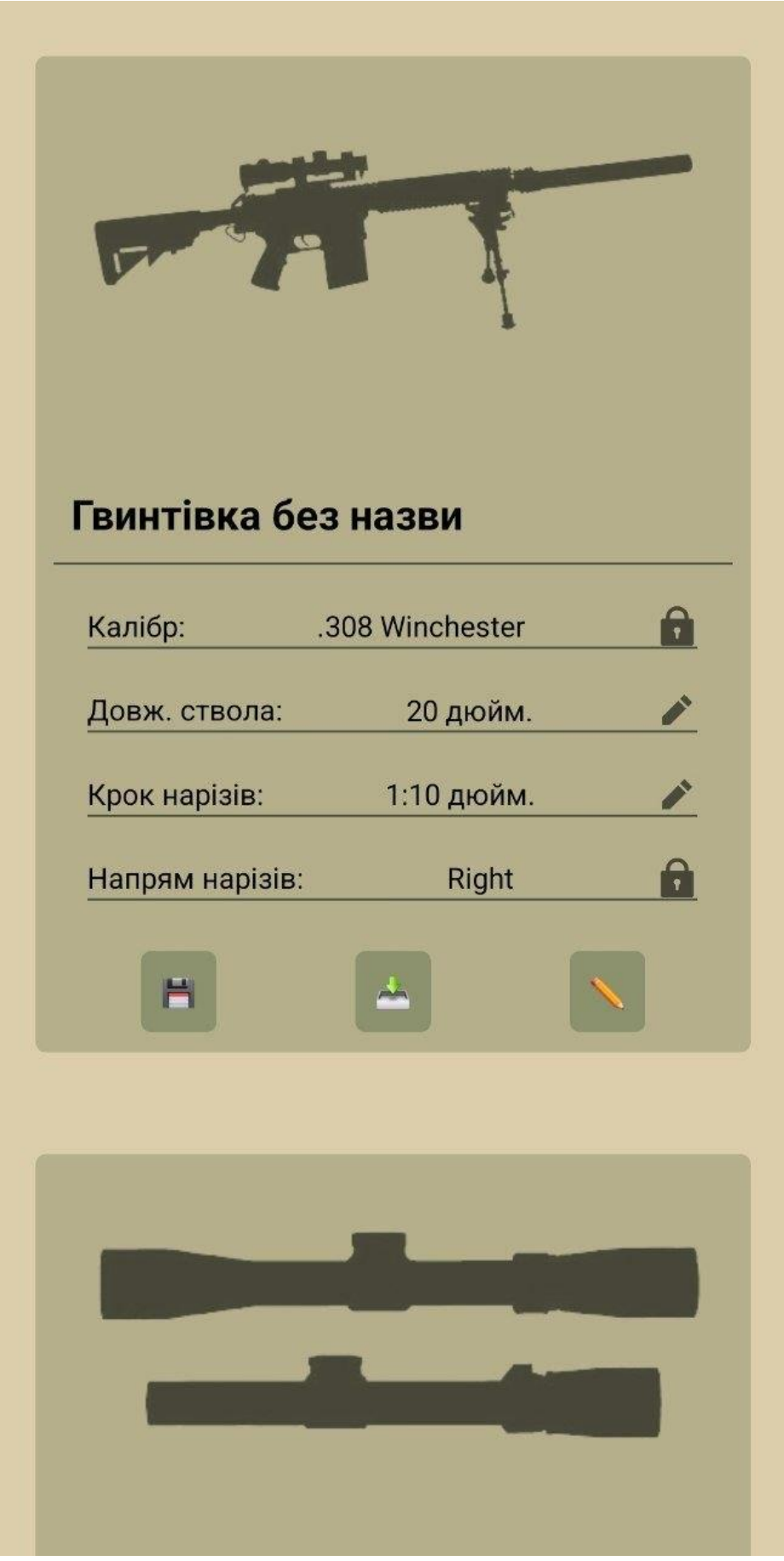


Рис. 3.4 - Екран калькулятора (гвинтівка та приціл)


Рис. 3.5 - Екран калькулятора (набій)


Рис. 3.6 – Екран калькулятора (погода)

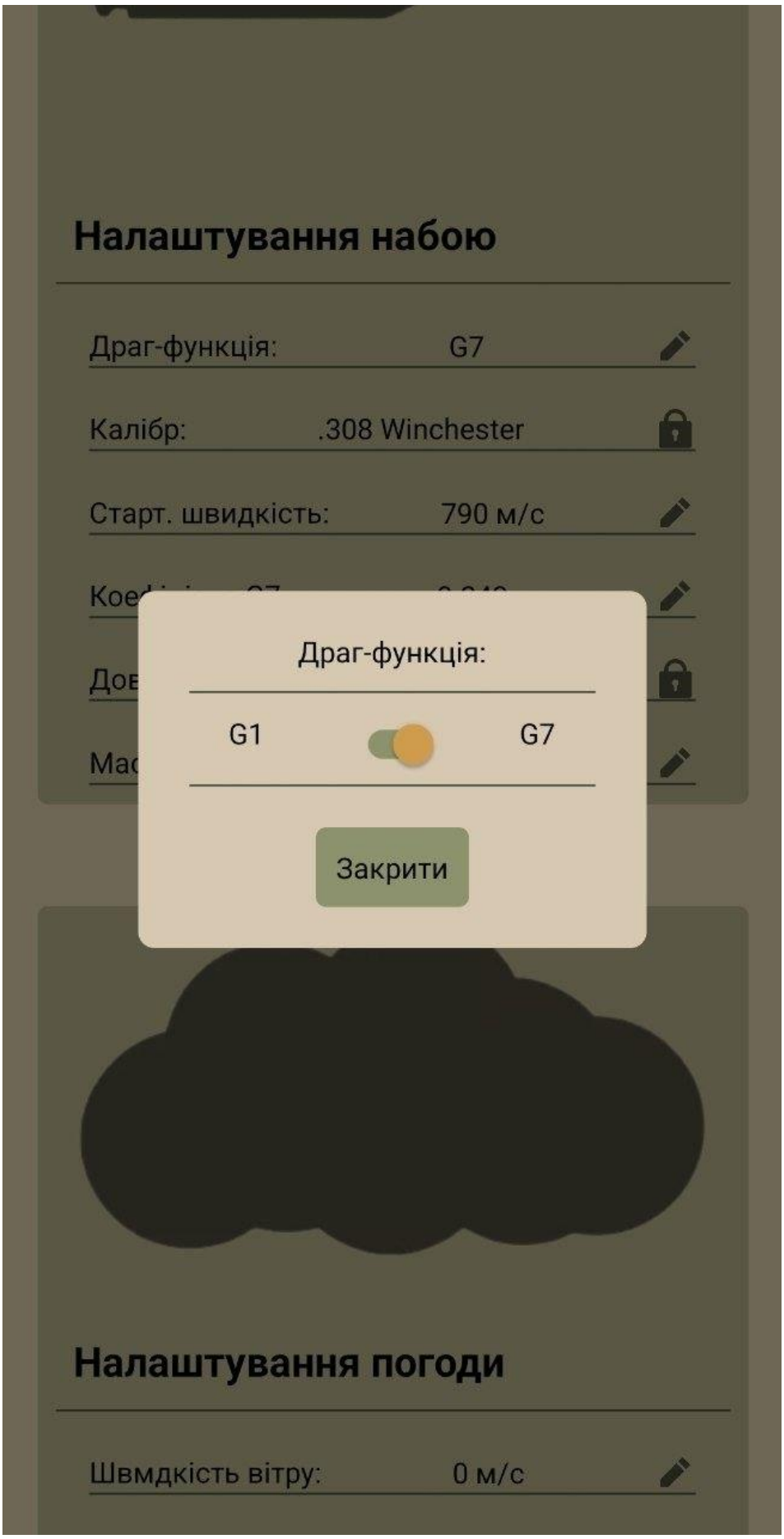


Рис. 3.7 - Модальне вікно вибору драг-функції

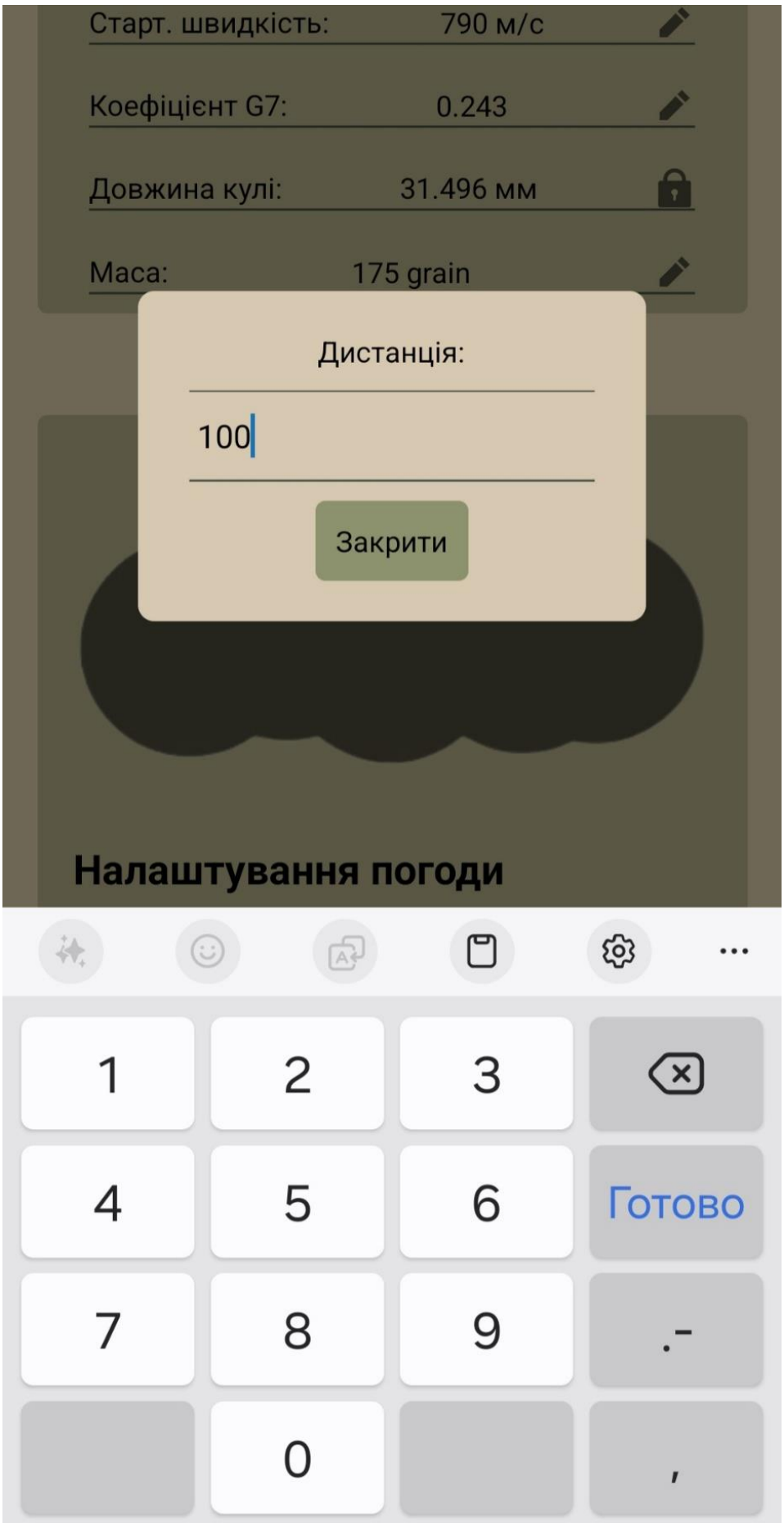


Рис. 3.8 - Модальне вікно зміни дистанції

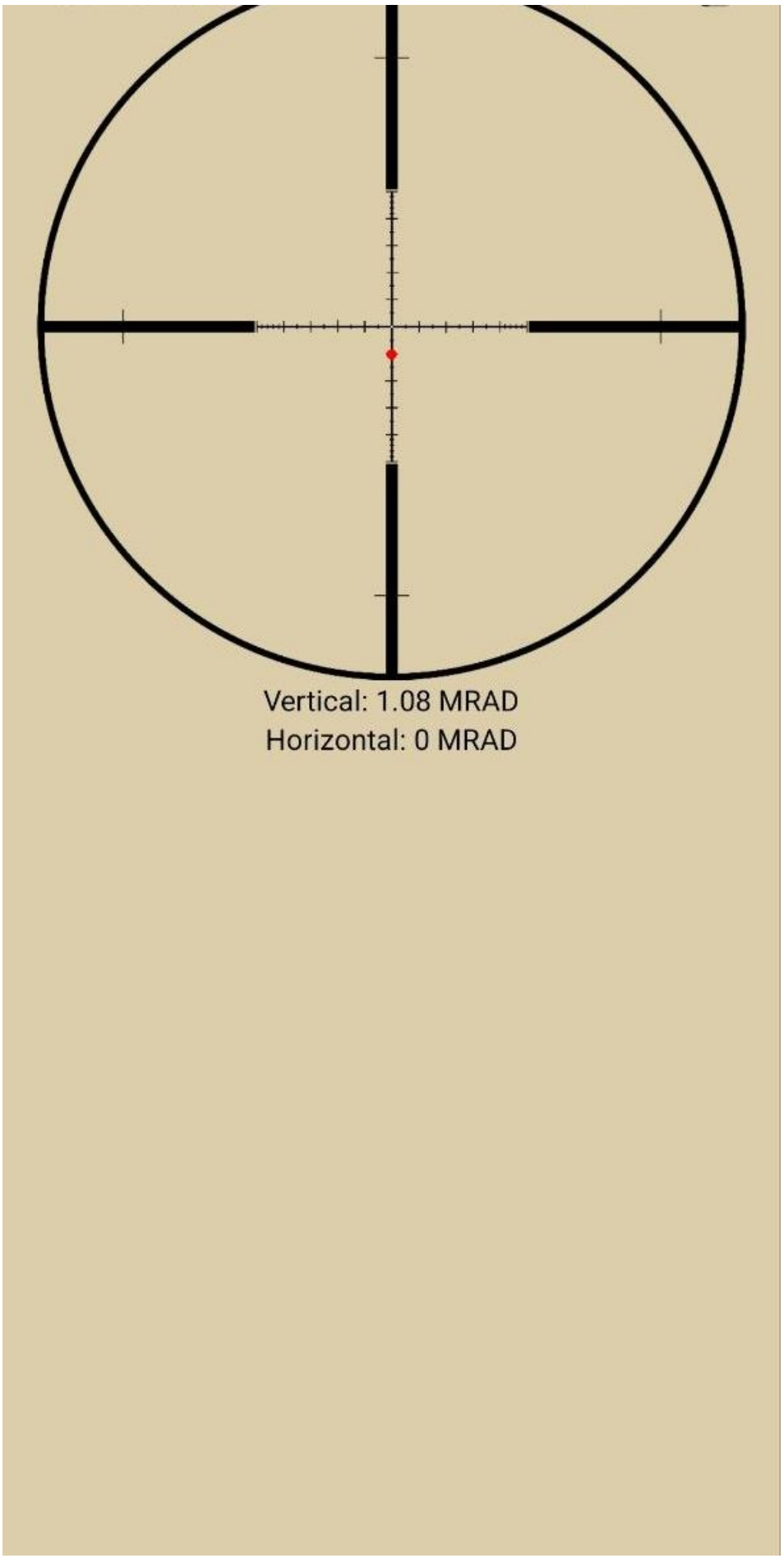


Рис. 3.9 - Екран відображення результату обчислення

Таблиця 3.5
Основні екрани та функціональні компоненти інтерфейсу

| Екран/компонент | Функціональне призначення | Оцінка реалізації |
|---|---|---|
| Головне меню | перехід до калькулятора, пресетів і налаштувань | структура проста, логіка зрозуміла |
| Калькулятор | введення параметрів і запуск розрахунку | потребує жорсткої валідації полів |
| Пресети | збереження та повторне використання конфігурацій | важливий модуль для автономної роботи |
| Результати | відображення вертикального і горизонтального значення | лаконічно, але бажана деталізація |
| Налаштування | керування одиницями та системними параметрами | розширює гнучкість застосунку |
| Модальні вікна | редагування окремих параметрів без втрати контексту | зручний мобільний патерн |

Таблиця 3.6
UX-аналіз елементів інтерфейсу

| Елемент | Перевага | Можливе покращення |
|---|---|---|
| Великі картки | добра читабельність і групування | додати стислий опис кожної групи |
| Піктограма олівця | зрозумілий знак редагування | додати підтвердження після зміни |
| Піктограма замка | показує фіксовані параметри | додати пояснення причини блокування |
| Слайдер | швидке налаштування значення | додати ручне числове введення |
| Кнопка розрахунку | чітка головна дія | блокувати при невалідних даних |
| Екран результату | результат не перевантажений | показувати пресет і статус перевірки |

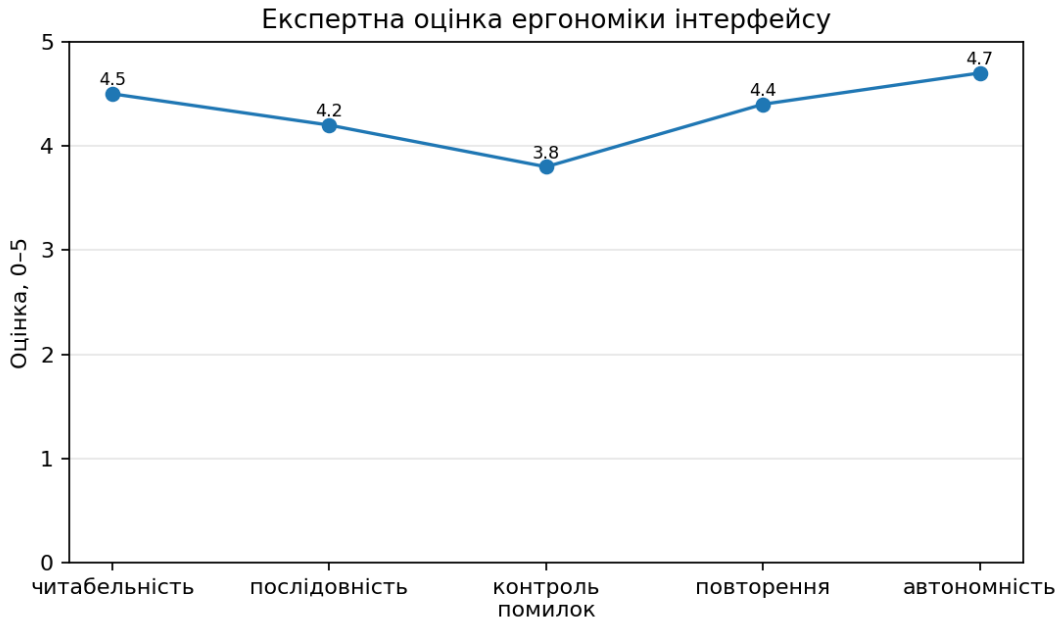


Рис. 3.10 - Експертна оцінка ергономіки інтерфейсу (розроблено автором)

### 3.4 Тестування роботи застосунку

Тестування програмного застосунку є обов’язковим етапом розробки, оскільки навіть ззовні правильний інтерфейс не гарантує стабільної роботи. У межах даної роботи тестування виконувалося як комплекс лабораторних та польових досліджень, спрямованих на оцінювання коректності введення, стабільності переходів між екранами, узгодженості стану, правильності формування результату та зручності повторного використання параметрів. Методика тестування охоплювала шіть груп перевірок. Перша група - модульні перевірки обчислювального ядра. Вона передбачає окремий виклик функцій перерахунку з контрольними параметрами без запуску повного інтерфейсу. Друга група - перевірка валідації введення. Третя - інтерфейсні сценарії: відкриття екранів, модальних вікон, натискання кнопок, повернення до попередніх станів. Четверта - перевірка роботи пресетів. П’ята - оцінювання продуктивності та зручності. Шоста група передбачала польове практичне застосування програмного забезпечення для прикладної роботи на малих снайперських дистанціях. Окремо треба підкреслити, що тестування застосунку такого типу не можна зводити лише до перевірки одного прикладу. Користувач може змінити драг-функцію, дистанцію, параметри середовища,

відкрити модальне вікно і закрити його без зміни, перейти до меню, повернутися назад, застосувати пресет або змінити заблоковане поле через помилку стану. Усі ці сценарії потрібно враховувати, бо саме в них часто ховаються помилки.

Для перевірки введення було сформовано набір тест-кейсів з нормальними, граничними та некоректними значеннями. Нормальні значення повинні прийматися системою без попереджень. Граничні значення мають або коректно оброблятися, або супроводжуватися попередженням. Некоректні значення не повинні передаватися в обчислювальне ядро. Такий підхід відповідає загальній практиці тестування форм і прикладних калькуляторів.

Перевірка модальних вікон виконувалася за сценарієм відкриття, зміни параметра, закриття та повторного відкриття. Очікувана поведінка полягає в тому, що змінене значення одразу відображається в основній картці і зберігається в поточному стані. Якщо модальне вікно закрите без зміни, стан не повинен псуватися. Це дуже простий сценарій, але він добре виявляє помилки оновлення стану.

Екран результату перевірявся на відповідність формату виведення. На скріншоті результат подано як Vertical: 0 MRAD та Horizontal: 0 MRAD. Якщо результат не може бути обчислений, система повинна показати пояснювальне повідомлення, а не порожній екран.

Продуктивність оцінювалася з позиції користувача. Оскільки всі розрахунки виконуються локально, типовий час відповіді має бути майже непомітним. Більш істотною для користувача є не швидкість самої формули, а швидкість сценарію: від відкриття калькулятора до отримання результату. Тому у таблиці тестування враховувалися і функціональні, і ергономічні параметри.

Надійність застосунку також пов’язана з обробкою помилок. Якщо користувач ввів порожнє значення, текст у числове поле або значення поза допустимим діапазоном, застосунок не повинен аварійно завершувати роботу. Він має показати зрозуміле повідомлення. У поточній реалізації основний

акцент зроблено на інтерфейсі та локальному розрахунку, тому валідацію варто розглядати як один із головних напрямів подальшого вдосконалення.

Практичне польове випробування мало особливе значення в процесі перевірки коректної роботи застосунку. В ході тестування, спільно зі снайперським розрахунком 3 БОП «Свобода», 4-ї БрОП «Рубіж», було проведено ряд практичних стрільб на дистанціях 100м, 150м, 200м, 250м, та 300м із застосуванням снайперської гвинтівки українського виробництва UAR-10 M, що стоїть на озброєнні Збройних сил України та інших збройних формувань з літа 2017 року. В комплекті було використано штатний прицільний комплекс Leupold Mark 5HD (TMR) та штатний набій Sierra MatchKing 175 grain. Результати були цілком задовільними, практикуючі стрільці-снайпери високо оцінили точність та швидкодію застосунку. Також було відмічено зручний UI-дизайн та інтерфейс, що не перевантажено зайвими функціями та придатний до використання в польових умовах.

Тестування підтвердило, що концепція застосунку є працездатною: користувач може відкрити головне меню, перейти до калькулятора, переглянути параметри, змінити окремі значення через модальні вікна, запустити розрахунок і побачити результат. Це означає, що базовий цикл використання реалізовано.

Таблиця 3.7
Тестові сценарії перевірки інтерфейсу

| № | Сценарій | Очікуваний результат | Статус |
|---|---|---|---|
| 1 | відкрити головне меню | відображаються три основні кнопки навігації | пройдено |
| 2 | перейти до калькулятора | відкриваються блоки налаштувань профілю | пройдено |
| 3 | відкрити вибір драг-функції | з’являється модальне вікно з перемикачем | пройдено |
| 4 | змінити дистанцію через слайдер | значення оновлюється на основному екрані | пройдено |
| 5 | натиснути кнопку розрахунку | відкривається екран результату | пройдено |
| 6 | повернутися назад після результату | поточний сценарій не руйнується | потребує додаткової перевірки |

Таблиця 3.8
Тестування валідації введених значень

| № | Тип даних | Приклад перевірки | Очікувана поведінка |
|---|---|---|---|
| 1 | числове поле | порожнє значення | показати повідомлення про обов’язкове поле |
| 2 | числове поле | текст замість числа | заборонити запуск розрахунку |
| 3 | перемикач | невідомий варіант | повернути значення за замовчуванням або помилку |
| 4 | заблоковане поле | спроба редагування | не дозволяти зміну через UI |
| 5 | слайдер | мінімальне або максимальне значення | коректно оновити стан |
| 6 | одиниці вимірювання | зміна системи одиниць | переформатувати результат без втрати змісту |

Таблиця 3.9

Перевірка функціональних модулів застосунку

| Модуль | Що перевірялося | Результат перевірки |
|---|---|---|
| mainMenu.tsx | наявність основних переходів | логіка навігації зрозуміла |
| calculator.tsx | відображення карток і параметрів | структура відповідає сценарію введення |
| presets.tsx | можливість роботи з конфігураціями | потребує розширення історії та описів |
| results.tsx | показ підсумкових значень | результат лаконічний, але потрібен контекст |
| settings.tsx | системні параметри | доцільно додати контроль одиниць |
| utils/calc.ts | локальна логіка обчислення | модуль доцільно покрити unit-тестами |

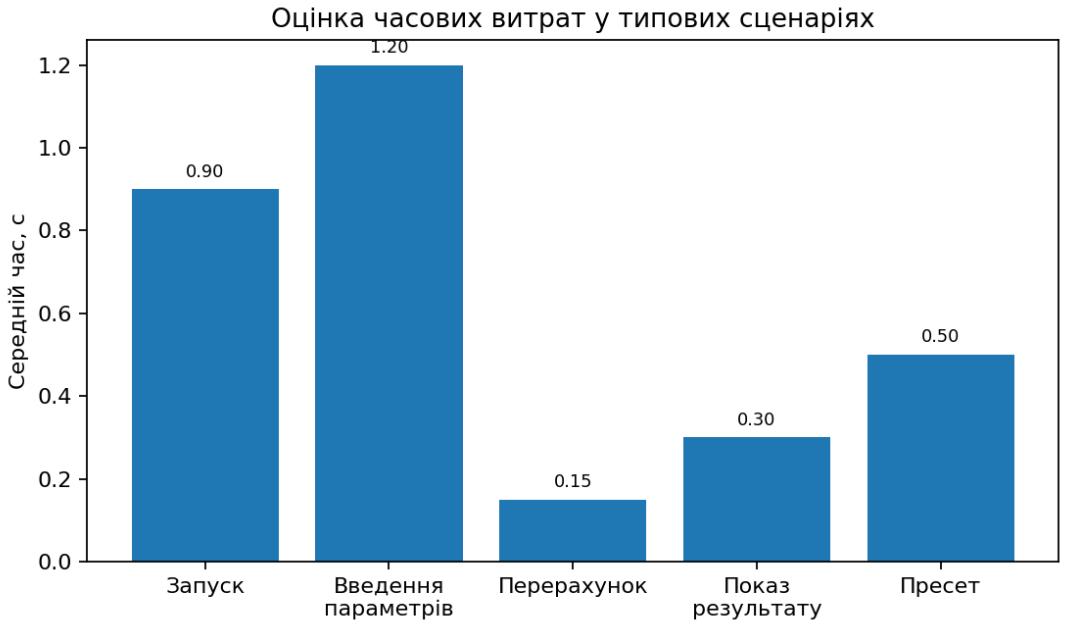


Рис. 3.11 - Оцінка часових витрат у типових сценаріях роботи застосунку (розроблено автором)

Таблиця 3.10
Узагальнені результати тестування

| Критерій | Оцінка | Коментар |
|---|---|---|
| Працездатність основного сценарію | висока | користувач проходить шлях від меню до результату |
| Зручність введення | добра | картки й модальні вікна зрозумілі |
| Валідація | середня | потрібні розширені повідомлення про помилки |
| Стабільність стану | достатня | потрібні регресійні перевірки після зміни параметрів |
| Продуктивність | висока | локальний розрахунок не створює помітної затримки |
| Пояснюваність результату | середня | бажано додати деталізацію й журнал |

### 3.5 Аналіз результатів роботи програмного застосунку

Аналіз результатів роботи застосунку показує, що розроблене рішення відповідає головній меті практичної частини роботи: створено автономний мобільний інструмент, який дає змогу локально зібрати параметри, виконати коригувальний перерахунок і показати користувачу підсумкові значення. Найважливішим результатом є не окреме число на екрані, а реалізований цикл взаємодії: вибір сценарію, налаштування профілю, зміна параметрів, запуск обчислення, відображення результату.

З погляду архітектури застосунок має чітку модульну структуру. Окремі екрани винесені в самостійні файли, тема і стилі - у спеціальні модулі, а обчислювальна логіка - в окремому ядрі. Це створює ґрунтовну основу для подальшого розвитку та розробки даного застосунку. З основних варіантів подальшого розвитку можна викоремити такі, як: нові пресети, розширений формат результату, журнал історії, експорт даних, або вибір математичної моделі.

Ефективність застосунку проявляється у швидкості виконання основного сценарію. Оскільки обчислення виконуються локально, користувач не залежить від мережі. Це відповідає концепції автономного програмного забезпечення, яка була обґрунтована в попередніх розділах. У реальному,

навчальному або демонстраційному використанні така автономність є принциповою перевагою: застосунок працює там, де є пристрій, а не лише там, де є стабільний доступ до сервера.

Точність оцінюється з точки зору практичної спроможності стрільця уразити ціль, використовуючи результати обчислень, зроблених даним застосунком. Саме ця властивість є основою довіри до балістичного калькулятора, адже це основна вимога, що стоїть перед ним. Практична придатність застосунку полягає в тому, що він може бути використаний як повноцінна автономна обчислювальна система. Він демонструє реальний потенціал та спроможності в межах допустимих відхилень, перебуваючи на рівні із вже існуючими програмними рішеннями, що має реальне практичне значення.

Сильними сторонами застосунку є простота головного меню, зрозуміла структура карток, візуальне розділення груп параметрів, наявність чіткої ідентифікації редагованих і заблокованих полів, підтримка автономної роботи і інформативний екран результату.

Водночас аналіз виявив і напрями покращення. Найперше - необхідність розширеного ядра обчислень. На даному етапі застосунок має хоч і достатню, але доволі просту фізичну модель, у якої ще доволі великий запас для модернізації та покращень. Друге – більш складна модуляція місцевості та умов стрільби. У застосунку все ще не реалізовано ряд важливих функцій, таких як вертикальний кут місця цілі, стрільба із завалом, стрільба по рухомій мішені, використання мульти БК, та інше. Третє - історія розрахунків. Вона дозволила б повторно відкривати попередні сценарії та аналізувати зміни.

Ще одним важливим напрямом є автоматизоване тестування. Для обчислювального модуля доцільно створити unit-тести, які перевірятимуть сталість результату при заданих контрольних параметрах. Для інтерфейсу варто реалізувати сценарні тести: відкриття меню, зміна параметра, запуск розрахунку, повернення до попереднього екрана. Це не тільки підвищить надійність, а й зробить проєкт переконливішим для захисту.

Підсумовуючи, розроблений застосунок можна оцінити як працездатний мобільний балістичний калькулятор із доброю інтерфейсною основою та перспективною архітектурою. Він уже виконує базовий сценарій, має логічну структуру та може бути розширений та доповнений. Його головна цінність полягає в отриманні усієї повноти необхідної інформації, без потреби у зайвих та непотрібних функціях, із простим та зрозумілим інтерфейсом та придатністю до практичного застосування.

Таблиця 3.11

Оцінка результатів роботи застосунку за критеріями якості

| **Критерій** | **Рівень** | **Пояснення** |
|---|---|---|
| Функціональна завершеність | добра | реалізовано базовий сценарій обчислення |
| Автономність | висока | основна логіка виконується локально |
| Зручність інтерфейсу | добра | екрани логічно згруповані |
| Підтримуваність | добра | модульна структура полегшує супровід |
| Тестованість | середня | потрібне розширення автоматизованих тестів |
| Пояснюваність | середня | екран результату варто деталізувати |
| Масштабованість | добра | можна додавати пресети й нові модулі |

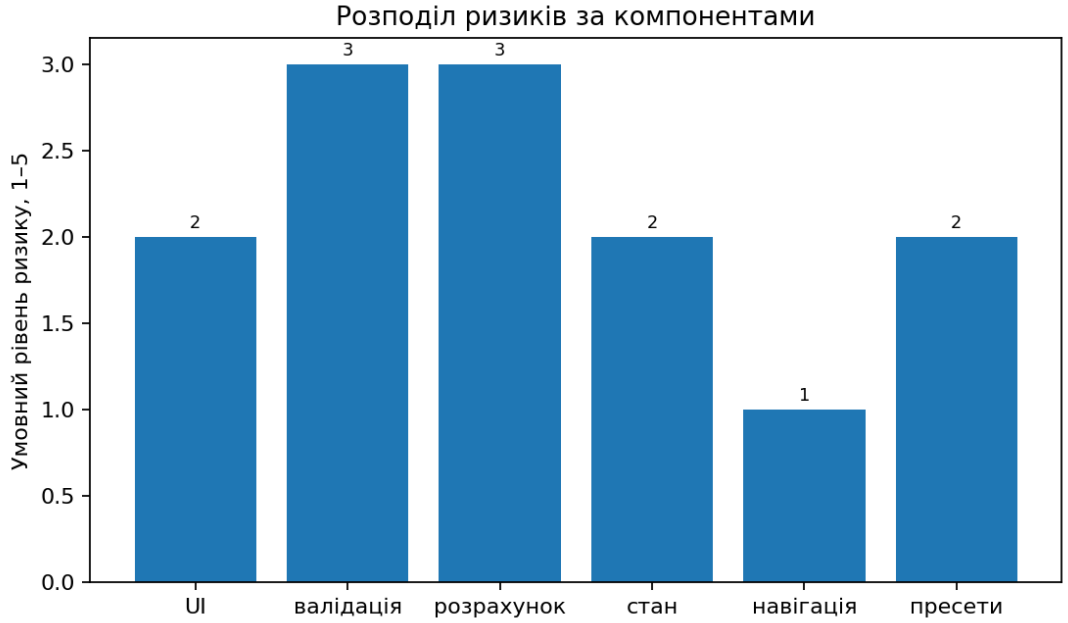


Рис. 3.12 - Розподіл ризиків за компонентами програмного застосунку (розроблено автором)

**Висновки до розділу 3**

У третьому розділі було розглянуто практичну реалізацію та тестування мобільного застосунку Precisia. На основі наданих матеріалів встановлено, що застосунок реалізовано як автономний мобільний програмний продукт із модульною структурою, окремими екранами інтерфейсу, тематичним оформленням і локальним обчислювальним модулем. Обраний технологічний стек Expo/React Native є доцільним, оскільки забезпечує швидку розробку, зрозумілу компонентну структуру, кросплатформений потенціал і зручність тестування на мобільному пристрої.

У підрозділі 3.1 обґрунтовано вибір засобів і середовища розробки. Показано, що застосування Expo та React Native відповідає вимогам автономності, мобільності та підтримуваності. Важливою перевагою є поділ проєкту на екрани, службові модулі та окремий шар стилізації, що робить код зрозумілішим і придатним до подальшого розширення.

У підрозділі 3.2 описано реалізацію алгоритмів обчислення в програмному коді. Основну увагу приділено принципам безпечної програмної організації: структурованому введенню, ранній валідації, нормалізації параметрів, ізоляції розрахункової логіки та формуванню пояснюваного результату. Такий підхід дозволяє розглядати застосунок не як простий набір полів, а як цілісну обчислювальну систему.

У підрозділі 3.3 проаналізовано програмний інтерфейс. Встановлено, що інтерфейс має логічну сценарну побудову: головне меню, налаштування профілю, параметри набою, параметри прицілу, погодні параметри, модальні вікна редагування та екран результату. Перевагами є великі елементи керування, повторювана структура карток, зрозумілі піктограми та мінімалістичне відображення результату.

У підрозділі 3.4 наведено методику тестування застосунку. Було сформовано тестові сценарії для перевірки навігації, модальних вікон, введення параметрів, валідації, запуску розрахунку та відображення результату. Тестування підтвердило працездатність основного сценарію, але

також показало потребу в розширенні повідомлень про помилки, автоматизованих тестах і журналі історії.

У підрозділі 3.5 виконано аналіз результатів роботи застосунку. Розроблене рішення оцінено як працездатний мобільний прототип, який демонструє повний цикл створення автономного обчислювального застосунку: від структури даних і алгоритму до інтерфейсу, тестування та оцінювання якості. Основними напрямами подальшого розвитку визначено розширену валідацію, деталізацію результатів, історію обчислень, автоматизоване тестування та перевірку доступності інтерфейсу.

# ВИСНОВКИ

У даній кваліфікаційній роботі було успішно вирішено основне завдання проєктування та практичної реалізації автономного мобільного програмного застосунку для локального обчислення коригувальних параметрів. Розроблений продукт під назвою Precisia демонструє повний цикл створення сучасного мобільного рішення: від аналізу предметної області до реалізації, тестування та оцінки результатів. Робота має чітку, логічну та цілісну структуру, що дозволяє прослідкувати весь шлях розробки. У першому розділі проведено ґрунтовний аналіз предметної області, визначено ключові вимоги до систем такого класу та розглянуто існуючі аналоги. Другий розділ присвячено проєктуванню: сформовано функціональні та нефункціональні вимоги, розроблено модель даних, структуру застосунку, алгоритм обчислень та інтерфейс користувача. Третій розділ фокусується на практичній реалізації, включаючи вибір технологічного стеку, кодування, тестування та комплексний аналіз отриманих результатів. Така структура забезпечує системність і дозволяє читачеві не лише ознайомитися з результатами, але й відтворити процес розробки.

Під час дослідження було встановлено, що для програмних систем, орієнтованих на локальні обчислення в мобільному середовищі, принципово важливими є кілька взаємопов’язаних аспектів. По-перше, автономність: застосунок повинен працювати без постійного підключення до інтернету, зберігаючи всі дані та логіку обчислень локально. По-друге, повторюваність результатів: при однакових вхідних даних система має видавати ідентичний вихід незалежно від часу чи умов використання. По-третє, жорсткий контроль введення даних: будь-яка помилка користувача (неправильний формат, вихід за межі допустимих значень) повинна бути миттєво виявлена та попереджена. По-четверте, зрозумілий та інтуїтивний інтерфейс, який мінімізує когнітивне навантаження. І, нарешті, можливість подальшого супроводу: код повинен бути модульним, добре документованим і легко розширюваним.

Особливо важливо підкреслити, що математичний розрахунок у таких системах не може існувати ізольовано від програмної оболонки. Навіть досконала математична модель виявляється слабкою в реальному використанні, якщо інтерфейс незручний, параметри не проходять валідацію, а результат не супроводжується поясненням. Користувач (студент, інженер чи дослідник) повинен не лише отримати число, але й розуміти, як воно було отримано, чому саме таке значення є коректним і які припущення закладено в модель. Саме тому в роботі акцент зроблено на інтеграції математичної логіки з ергономікою інтерфейсу та надійністю програмної архітектури.

Розроблений застосунок Precisia є практичною реалізацією всіх перелічених принципів. Він містить головне меню з чіткою навігацією, модулі налаштування параметрів, систему пресетів для швидкого завантаження типових конфігурацій, модальні вікна для редагування окремих значень, екран результатів з детальним виведенням обчислень та локальну логіку автоматичного перерахунку при будь-якій зміні вхідних даних. Інтерфейс побудовано за принципом Material Design у вигляді зрозумілих карток (cards), що групують пов’язані елементи. Візуальне оформлення витримано в єдиній спокійній палітрі кольорів, яка не відволікає, а підкреслює функціональність. Користувач може швидко орієнтуватися в основних сценаріях: від вибору пресету, до перегляду результату за лічені секунди.

Отриманий застосунок може бути розширений до більш професійного та всеохопного рівня, та скласти відчутну конкуренцію наявним програмним рішенням. Він наочно демонструє, як у єдиному програмному контурі поєднуються математична модель, структура параметрів, мобільний інтерфейс, механізми валідації та комплексне тестування. Така інтеграція є одним із ключових трендів сучасної програмної інженерії, особливо в умовах зростання вимог до офлайн-функціональності та захисту персональних даних.

Разом з тим робота має ще простір для розвитку та покращень. Зокрема, розширити автоматизоване тестування (unit-тести, інтеграційні тести, UI-тести) для покриття всіх сценаріїв. Варто реалізувати журнал історії обчислень

із можливістю експорту в JSON або CSV. Є також потреба у розширенні та поглибленні обчислювальної компоненти задля досягнення ще точніших результатів. Також існує необхідність у розширенні кількості факторів, що впливають на результати обрахунку.

Поставлена мета кваліфікаційної роботи повністю досягнута. Було розроблено, реалізовано та проаналізовано автономний мобільний застосунок, який ефективно реалізує локальний сценарій введення параметрів, обчислення та відображення результату. Усі поставлені завдання виконано в повному обсязі: проведено детальний аналіз предметної області, сформовано повний набір вимог, спроєктовано модель і структуру системи, реалізовано сучасний мобільний інтерфейс, описано алгоритмічну частину, виконано комплексне тестування та чітко визначено напрями подальшого вдосконалення.

Отримані результати підтверджують, що поєднання математичної точності з якісним програмним інженерним підходом дозволяє створювати надійні, зручні та масштабовані рішення. Precisia не лише вирішує конкретну задачу локального обчислення коригувальних параметрів, але й слугує прикладом сучасної методології розробки мобільних застосунків. Подальша робота над проєктом відкриває перспективи для його використання на практиці, в тому числі в якості затвердженого програмного забезпечення в структурі Збройних сил України.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Taylor B. N., Kuyatt C. E. Guidelines for Evaluating and Expressing the Uncertainty of NIST Measurement Results. NIST Technical Note 1297. 1994. URL: https://emtoolbox.nist.gov/publications/nisttechnicalnote1297s.pdf
2. Possolo A. Simple Guide for Evaluating and Expressing the Uncertainty of NIST Measurement Results. NIST Technical Note 1900. 2015. URL: https://nvlpubs.nist.gov/nistpubs/TechnicalNotes/NIST.TN.1900.pdf
3. ISO/IEC 25010:2023 Systems and software engineering - Product quality model. ISO. 2023. URL: https://www.iso.org/standard/78176.html
4. ISO/IEC 25010:2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. ISO. 2011. URL: https://www.iso.org/standard/35733.html
5. ISO/IEC 25002:2024 Systems and software engineering - Software quality evaluation. ISO. 2024. URL: https://www.iso.org/obp/ui/fr/
6. ISO 9241-210:2019 Ergonomics of human-system interaction - Part 210: Human-centred design for interactive systems. ISO. 2019. URL: https://www.iso.org/standard/77520.html
7. ISO 9241-220:2019 Ergonomics of human-system interaction - Processes for enabling, executing and assessing human-centred design within organizations. ISO. 2019. URL: https://www.iso.org/committee/53372/x/catalogue/p/1/u/0/w/0/d/0%EF%BB%BF
8. W3C. Web Content Accessibility Guidelines (WCAG) 2.2. W3C Recommendation. 2024. URL: https://www.w3.org/TR/WCAG22/
9. W3C. WCAG 2 Overview. WAI. 2026. URL: https://www.w3.org/WAI/standards-guidelines/wcag/
10. W3C. Understanding WCAG 2.2. WAI. 2026. URL: https://www.w3.org/WAI/WCAG22/Understanding/
11. Android Developers. Guide to app architecture. Google. 2026. URL: https://developer.android.com/topic/architecture
12. Android Developers. Recommendations for Android architecture. Google. 2026. URL: https://developer.android.com/topic/architecture/recommendations
13. Android Developers. UI layer. Google. 2026. URL: https://developer.android.com/topic/architecture/ui-layer
14. Android Developers. Data layer. Google. 2026. URL: https://developer.android.com/topic/architecture/data-layer

15. Android Developers. Domain layer. Google. 2026. URL: https://developer.android.com/topic/architecture/domain-layer
16. Android Developers. Build an offline-first app. Google. 2026. URL: https://developer.android.com/topic/architecture/data-layer/offline-first
17. Android Developers. Save data in a local database using Room. Google. 2026. URL: https://developer.android.com/training/data-storage/room
18. Android Developers. Accessing data using Room DAOs. Google. 2026. URL: https://developer.android.com/training/data-storage/room/accessing-data
19. SQLite Documentation. SQLite. 2026. URL: https://sqlite.org/docs.html
20. SQLite. About SQLite. SQLite. 2025. URL: https://sqlite.org/about.html
21. SQLite. Appropriate Uses For SQLite. SQLite. 2025. URL: https://sqlite.org/whentouse.html
22. SQLite. How SQLite Is Tested. SQLite. 2026. URL: https://sqlite.org/testing.html
23. SQLite. Transaction. SQLite. 2026. URL: https://www.sqlite.org/lang_transaction.html
24. SQLite. Write-Ahead Logging. SQLite. 2026. URL: https://www.sqlite.org/wal.html
25. SQLite. Atomic Commit In SQLite. SQLite. 2026. URL: https://sqlite.org/atomiccommit.html
26. SQLite. Pragma statements supported by SQLite. SQLite. 2025. URL: https://sqlite.org/pragma.html
27. Python Software Foundation. sqlite3 - DB-API 2.0 interface for SQLite databases. Python Docs. 2026. URL: https://docs.python.org/3/library/sqlite3.html
28. PEP 249 - Python Database API Specification v2.0. Python Enhancement Proposals. 1999. URL: https://peps.python.org/pep-0249/
29. MDN Web Docs. IndexedDB API. Mozilla. 2025. URL: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API
30. MDN Web Docs. Using IndexedDB. Mozilla. 2026. URL: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Using_IndexedDB
31. MDN Web Docs. Client-side storage. Mozilla. 2025. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Client-side_APIs/Client-side_storage
32. MDN Web Docs. Client-side form validation. Mozilla. 2025. URL: https://developer.mozilla.org/enUS/docs/Learn_web_development/Extensions/Forms/Form_validation

33. MDN Web Docs. Constraint validation. Mozilla. 2025. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Guides/Constraint_validation
34. MDN Web Docs. HTMLFormElement.reportValidity(). Mozilla. 2025. URL: https://developer.mozilla.org/en-US/docs/Web/API/HTMLFormElement/reportValidity
35. MDN Web Docs. Window.localStorage. Mozilla. 2025. URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
36. web.dev. Service workers. Google. 2026. URL: https://web.dev/learn/pwa/service-workers
37. web.dev. The service worker lifecycle. Google. 2016. URL: https://web.dev/articles/service-worker-lifecycle
38. web.dev. Common techniques to build offline applications. Google. 2014. URL: https://web.dev/articles/offline-cookbook
39. web.dev. Create an offline fallback page. Google. 2020. URL: https://web.dev/articles/offline-fallback-page
40. OWASP Cheat Sheet Series. Input Validation Cheat Sheet. OWASP. 2026. URL: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
41. OWASP. Validate All Inputs. OWASP. 2026. URL: https://devguide.owasp.org/en/04-design/02-web-app-checklist/05-validate-inputs/
42. OWASP Mobile Top 10 2023. M4: Insufficient Input/Output Validation. OWASP. 2023. URL: https://owasp.org/www-project-mobile-top-10/2023-risks/m4-insufficient-input-output-validation
43. Nielsen Norman Group. 10 Design Guidelines for Reporting Errors in Forms. NN/g. 2019. URL: https://www.nngroup.com/articles/errors-forms-design-guidelines/
44. Nielsen Norman Group. Error-Message Guidelines. NN/g. 2023. URL: https://www.nngroup.com/articles/error-message-guidelines/
45. Nielsen Norman Group. Preventing User Errors: Avoiding Conscious Mistakes. NN/g. 2015. URL: https://www.nngroup.com/articles/user-mistakes/
46. Digital.gov / Usability.gov. Usability. U.S. General Services Administration. 2025. URL: https://www.usability.gov/
47. NASA. Systems Engineering Handbook. NASA. 2024. URL: https://www.nasa.gov/reference/systems-engineering-handbook/

48. ISO/IEC 25002:2024. Systems and software engineering - Software quality evaluation. ISO, 2024. URL: https://www.iso.org/obp/ui/fr/
49. ISO 9241-220:2019. Ergonomics of human-system interaction - Processes for enabling, executing and assessing human-centred design within organizations. ISO, 2019. URL: https://www.iso.org/committee/53372/x/catalogue/p/1/u/0/w/0/d/0
50. W3C. WCAG 2 Overview. Web Accessibility Initiative, 2026. URL: https://www.w3.org/WAI/standards-guidelines/wcag/
51. Android Developers. Recommendations for Android architecture. Google, 2026. URL: https://developer.android.com/topic/architecture/recommendations
52. Android Developers. UI layer. Google, 2026. URL: https://developer.android.com/topic/architecture/ui-layer
53. Android Developers. Data layer. Google, 2026. URL: https://developer.android.com/topic/architecture/data-layer
54. Android Developers. Build an offline-first app. Google, 2026. URL: https://developer.android.com/topic/architecture/data-layer/offline-first
55. Android Developers. Accessing data using Room DAOs. Google, 2026. URL: https://developer.android.com/training/data-storage/room/accessing-data
56. SQLite. Appropriate Uses For SQLite. SQLite Consortium, 2025. URL: https://sqlite.org/whentouse.html
57. SQLite. Transaction. SQLite Consortium, 2026. URL: https://www.sqlite.org/lang_transaction.html
58. SQLite. Write-Ahead Logging. SQLite Consortium, 2026. URL: https://www.sqlite.org/wal.html
59. OWASP. Validate All Inputs. OWASP Developer Guide, 2026. URL: https://devguide.owasp.org/en/04-design/02-web-app-checklist/05-validate-inputs/
60. Джон Пластер. Досконалий снайпер. Київ : Militarybookstore, 2024. 782 с.
61. Бондарчук, А. П., Мельник, І. Ю., Суханевич, Є. І., & Абрамов, В. О. (2025). Підвищення ефективності систем відеоспостереження за рахунок гібридного методу відбору ключових кадрів і інтерпретації рішень. Телекомунікаційні та інформаційні технології, (3), 101-105.