Технічне завдання (без бота)

Проєкт: Десктопна програма для розрахунку ERR (Engagement Rate by Reach) за Instagram-постом
Форм-фактор: CLI/TUI (консоль) з опційним базовим GUI
Замовник: SMM-менеджер / маркетинг-відділ
Виконавець: Python-розробник (OOP)
Мова інтерфейсу: UA (базово), i18n-каркас

---

1) Мета

Надати простий інструмент, що швидко рахує **ERR** для одного або багатьох постів Instagram. Джерело даних: - **Ручний режим** (основний у MVP): користувач вводить reach/likes/comments/saves.
- **API-режим** (опційний): отримання метрик через Instagram Graph API при наявності токенів.

---

2) Глосарій

- **ERR (Engagement Rate by Reach):** `(engagements / reach) × 100%`
- **Engagements (Instagram пост):** `likes + comments + saves` (або агрегована метрика `engagement` з Insights)
- **Reach:** унікальні акаунти, що бачили пост (за одиничний пост)

---

3) Обсяг робіт (Scope)

3.1. Включено

- Консольний інтерфейс (CLI) + покращена **TUI** (на базі форматованого виводу; напр. підсвітка, таблиці)
- **Калькулятор ERR** для одиничного поста й пакетної обробки (CSV)
- **Історія розрахунків** у локальній БД (SQLite) з пошуком/фільтрами
- **Експорт** результатів у CSV/JSON
- **Гнучкі формули**: вибір, що входить до engagements; за замовчуванням `likes+comments+saves`
- **Валідація** введених значень, обробка помилок, дружні повідомлення
- **Конфігурація** через файл `.env` або `config.json` (десяткові розряди, мова, шлях БД, тощо)
- **Опціонально:** базовий GUI (мінімалістичне вікно з полями вводу), якщо дозволяє бюджет/терміни

3.2. Не включено (MVP)

- Інтеграції з іншими соцмережами
- Розширені дашборди/аналітика акаунта
- Хмарна синхронізація даних

---

4) Цільова платформа та середовище

- **OS:** Windows 10+, macOS 12+, Linux (x86_64)
- **Python:** 3.10+
- **Розповсюдження:** як скрипт (`pipx` / `venv`) або як самодостатній виконуваний файл (через pyinstaller)

---

5) Режими використання

5.1. Ручний режим (MVP)

- Користувач запускає програму → обирає «Розрахувати ERR»
- Вводить: `URL (необов'язково)` , `reach` , `likes` , `comments` , `saves`
- Програма рахує ERR, показує таблицю з деталями, пропонує «Зберегти в історію»

5.2. Пакетна обробка (CSV)

- Вхідний CSV: стовпці `url, reach, likes, comments, saves`
- Вивід: CSV із додатковими полями `engagements, err_%`
- Звіт про рядки з помилками (некоректні числа/порожні значення)

5.3. API-режим (опційно)

- Налаштування токенів IG Graph API у `.env` / `config.json`
- Користувач вказує `instagram_permalink` або `ig_media_id`
- Програма отримує reach та engagements (або like_count+comments_count+saves) → рахує ERR
- Кешує результати в БД (щоб менше звертатися до API)

---

6) Функціональні вимоги

6.1. Команди CLI

- `err single` — порахувати ERR для одного поста (ручний ввід)
- `err csv --in file.csv --out result.csv` — пакетна обробка
- `history list [--limit 50] [--filter date>=...]` — показати історію
- `history show <id>` — деталі запису
- `export csv/json [--out path] [--filter ...]` — експорт історії
- `formula set --use (lcs|insights)` — `lcs` = likes+comments+saves; `insights` = engagement з API
- `settings set --decimals 2 --lang ua` — системні параметри
- `api test` — перевірка доступності токенів та прав (якщо використовується)

6.2. Валідація вводу

- Цілі невід'ємні числа для `reach/likes/comments/saves`
- `reach = 0` → повідомити, що розрахунок неможливий
- Некоректний CSV (відсутні стовпці/типи) → читабельна помилка + приклад валідного файлу

6.3. Результати

- Вивід ERR у відсотках з **N** знаками (за замовчуванням 2)
- Відображення проміжних розрахунків: engagements, формула, первинні метрики

---

7) Архітектура (OOP)

- **Core**
- `Calculator` — обчислення ERR, форматування, округлення
- `Metrics` — модель даних (reach, likes, comments, saves, engagement)
- `Formula` — політика складу engagements (`lcs` або `insights`)
- **Storage**
- `Persistence` (SQLite): DAO для `users` (опц.), `settings`, `history`
- **Import/Export**
- `CsvService` — читання/запис CSV із валідацією
- **API (опц.)**
- `InstagramAdapter` — отримання метрик по `ig_media_id` або `permalink` (за наявності доступів)
- **UI**
- `CLI/TUI` — команди, форматований вивід таблиць, прогрес-бар (для CSV)
- `GUI (опц.)` — мінімалістичне вікно: поля, кнопка «Розрахувати», результат

---

8) Схема БД (SQLite)

- `settings(id PK=1, formula TEXT DEFAULT 'lcs', decimals INT DEFAULT 2, created_at TIMESTAMP)`
- `history(id PK, url TEXT, media_id TEXT, reach INT, likes INT, comments INT, saves INT, engagement INT, err REAL, created_at TIMESTAMP)`

---

9) Конфігурація

- `.env` / `config.json` параметри:
- `DECIMALS=2`
- `DB_PATH=./err.db`
- `LANG=ua`
- (опц.) `META_APP_ID`, `META_APP_SECRET`, `IG_LONG_LIVED_TOKEN`, `FB_PAGE_ID` тощо

---

10) UX-вимоги

10.1. CLI/TUI

- Команди з підказками `--help`
- Вивід у таблицях (заголовки, вирівнювання, підсумки)
- При пакетній обробці — підсумкова статистика: оброблено/помилки/середній ERR

10.2. GUI (опц.)

- Єдине вікно: поля (URL, reach, likes, comments, saves), перемикач формули, кнопки «Розрахувати», «Зберегти»
- Вивід ERR великим шрифтом, під ним — деталізація

---

11) Нефункціональні вимоги

- Продуктивність: 10k рядків CSV < 10 сек на сучасному ПК
- Надійність: транзакції при записі в БД, обробка винятків
- Портативність: пакування в один виконуваний файл за бажанням
- Локалізація: усі рядки через словники (UA/EN)

---

12) Тестування

- Юніт-тести: `Calculator` (нормальні/крайові випадки), `CsvService` (валід/невалід), `Persistence` (CRUD)
- Інтеграційні: пакетна обробка CSV (включно з битими рядками), (опц.) `InstagramAdapter` з моками відповіді
- UX-тести: читабельність виводу, сценарії помилок

---

13) Прийомка (Acceptance Criteria)

- **CLI**: доступні команди з §6.1, повертають коректні результати та коди виходу
- **ERR**: збігається з ручним розрахунком за специфікацією (2 знаки після коми, налаштовувані)
- **CSV**: коректно читається/пишеться, програма не падає на помилкових рядках
- **Історія**: зберігається в SQLite, доступна для перегляду/експорту
- **Документація**: README з інсталяцією/прикладами запуску, приклади CSV, приклад конфігу

---

14) План робіт (грубо)

1. Архітектура, моделі, калькулятор — 0.5–1 д.
2. CLI/TUI команди (single, csv, history, export, settings, formula) — 1.5–2 д.
3. Persistence + тести — 1 д.
4. CSV сервіс + тести — 0.5 д.
5. (Опц.) InstagramAdapter — 1–2 д.
6. (Опц.) Базовий GUI — 1 д.
7. Полірування, документація, пакування — 0.5–1 д.

---

15) Ризики та обмеження

- Без IG API неможливо автоматично отримувати reach для **чужих** постів — залишається ручний ввід
- Зміни в політиках Meta → нестабільність API-режиму
- Різні типи медіа (Reels/відео) — різні набори метрик; у MVP фокус на базових полях

---

16) Матеріали для розробника

- Приклади вхідного CSV

```
url,reach,likes,comments,saves
https://www.instagram.com/p/XXXX/,1500,120,14,30
,2000,180,25,40
```

- Приклад вихідного CSV

```
url,reach,likes,comments,saves,engagements,err_percent
https://www.instagram.com/p/XXXX/,1500,120,14,30,164,10.93
```

Код реалізації не входить до цього документа. За потреби підготую skeleton-репозиторій з README та тестами.