#### **Опис завдання**

Метою проєкту було створення системи обліку електроспоживання, яка дозволяє:

* Додавати нові електролічильники.
* Вносити показники споживання (денне та нічне).
* Розраховувати загальну вартість спожитої електроенергії відповідно до тарифів.
* Отримувати історію споживання для конкретного лічильника.
* Оновлювати тарифи на електроенергію.
* Експортувати історію споживання у CSV-файл.

### **Обґрунтування вибору стеку технологій**

Для виконання завдання потрібно було розробити програму для обліку електроспоживання, яка мала отримувати показники двофазного лічильника ("день-ніч"), розраховувати вартість електроенергії за різними тарифами та записувати дані у збережену історію.

В методичці пропонувалося використовувати довільну мову програмування (Java, Python, JS, C, C#, C++...), тому перед вибором технологій було важливо визначити основні вимоги до системи:

* Облік та збереження даних – потрібна робота з БД.
* Обрахунки та коригування споживання – потрібен ефективний обчислювальний механізм.
* API для взаємодії – бажано мати REST API для роботи через запити.
* Зручний спосіб використання – консольний інтерфейс (CLI).
* Можливість тестування – обов'язкова перевірка функціоналу через тести.

**На основі цих вимог був обраний стек Python + FastAPI + MySQL + SQLAlchemy + Pytest.**

## **Використані технології та обґрунтування вибору**

### **1 - Python 3.11**

Вибір Python пояснюється тим, що він дозволяє швидко розробляти та легко підтримувати код. Даний проєкт не потребує високонавантажених обчислень, тому Python підходить ідеально.

Як використовувався у проєкті?

* Основна логіка додатку написана на Python.
* Використовувався для реалізації FastAPI-сервера, CLI та роботи з базою даних.
* Легко тестується через Pytest.

### **2 - FastAPI (Фреймворк для створення API)**

FastAPI був обраний, тому що він:

* Швидший за Flask через використання асинхронності.
* Автоматично генерує Swagger-документацію (http://127.0.0.1:8000/docs).
* Має інтеграцію з SQLAlchemy, що дозволяє працювати з базою без зайвого SQL-коду.

Як використовувався у проєкті?

* Реалізовані API-ендпоінти (/add\_meter, /add\_reading, /history, /update\_tariffs, /export\_history).
* Використаний для обробки запитів від CLI.
* Підтримує JSON-формат для обміну даними між сервером та клієнтом.

### **3 - MySQL (База даних)**

Вибір MySQL обумовлений тим, що:

* Це реляційна БД, яка добре підходить для збереження структурованих даних (лічильники, показники).
* Швидка і ефективна при роботі з великими обсягами інформації.
* Простий синтаксис SQL-запитів, що дозволяє швидко створювати, редагувати та отримувати дані.

Як використовувався у проєкті?

* Таблиці meters, readings, history для збереження даних.
* Автоматичне оновлення значень у БД.
* Підключення через ORM (SQLAlchemy).

### **4 - SQLAlchemy (ORM для взаємодії з БД)**

Було обрано SQLAlchemy, тому що воно дозволяє:

* Працювати з БД через об'єктно-реляційне відображення (ORM), а не писати "чистий" SQL.
* Використовувати автоматичні міграції.
* Забезпечує захист від SQL-ін'єкцій.

Як використовувався у проєкті?

* Створені моделі Meter, Reading, History.
* Робота з БД через ORM-запити (db.query(...).filter(...)).

### **5 - Pytest (Тестування)**

Для автоматизованого тестування використаний Pytest, бо він:

* Має зручний синтаксис (assert замість unittest-структури).
* Дозволяє перевіряти API-запити через TestClient.
* Легко інтегрується у CI/CD-системи (GitHub Actions).

Як використовувався у проєкті?

* Перевірка додавання нових лічильників (test\_add\_meter).
* Перевірка обрахунків вартості (test\_calculate\_cost).
* Перевірка оновлення тарифів (test\_update\_tariffs).
* Експорт CSV (test\_export\_history).

Всі 13 тестів пройдено успішно.

### **6 - argparse + requests (CLI)**

Було реалізовано консольний інтерфейс (CLI), щоб користувач міг взаємодіяти з програмою без Swagger.

* argparse використаний для обробки команд (python cli.py add\_meter ...).
* requests для надсилання HTTP-запитів до FastAPI-сервера.

Як використовувався у проєкті?

* Користувач вводить дані в терміналі.
* Надсилаються запити до FastAPI, отримується відповідь.
* Можна додавати лічильник, показання, оновлювати тарифи, експортувати історію.

Додано інтерактивне меню CLI для зручності.

### **7 - CSV (Експорт даних)**

Формат CSV обраний для експорту історії, бо він:

* Легко відкривається в Excel, Google Sheets.
* Простий у реалізації (csv.writer).
* Може бути використаний для звіту споживання електроенергії.

Як використовувався у проєкті?

* Історія показників зберігається у history\_<meter\_id>.csv.
* Файл генерується командою python cli.py export\_history <meter\_id>.

### 

### **Структура проєкту та призначення файлів**

Проєкт має структуровану файлову систему, де кожна папка і файл відповідає за конкретну функцію.

### **Корінь проєкту:**

📂 cli.py — Консольний інтерфейс для роботи з API (додавання лічильника, отримання історії, оновлення тарифів).  
 📂 config.py — Конфігураційний файл з параметрами підключення до БД та тарифами.  
 📂 main.py — Головний файл, що запускає FastAPI-сервер.  
 📂 requirements.txt — Список бібліотек, необхідних для роботи програми.  
 📂 history\_X.csv — Збережені файли історії (експортовані споживчі дані).

### **Папка app/ (Головна логіка застосунку)**

📂 api/

* routes.py — Основні API-ендпоінти (додавання лічильника, оновлення тарифів, отримання історії).

📂 database/

* db.py — Підключення до MySQL через SQLAlchemy.

📂 models/

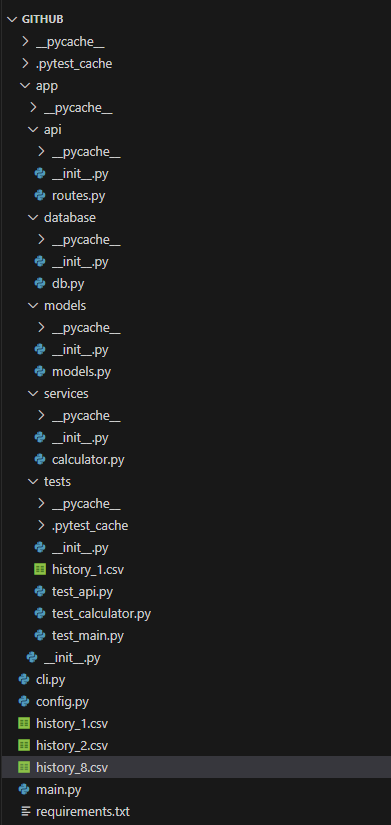
* models.py — Опис таблиць БД (meters, readings, history).

📂 services/

* calculator.py — Функція розрахунку вартості електроенергії.

### **Папка tests/ (Автоматичне тестування)**

📂 test\_api.py — Перевірка API-запитів (/add\_meter, /add\_reading, /history).  
 📂 test\_calculator.py — Перевірка розрахунків вартості електроенергії.  
 📂 test\_main.py — Загальні тести запуску сервера.

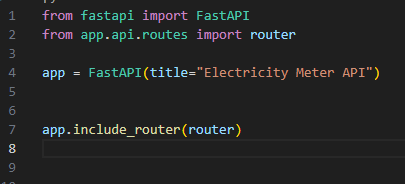
****

**Реалізація проєкту**

### **1. Запуск сервера (main.py)**

**Що робить:**

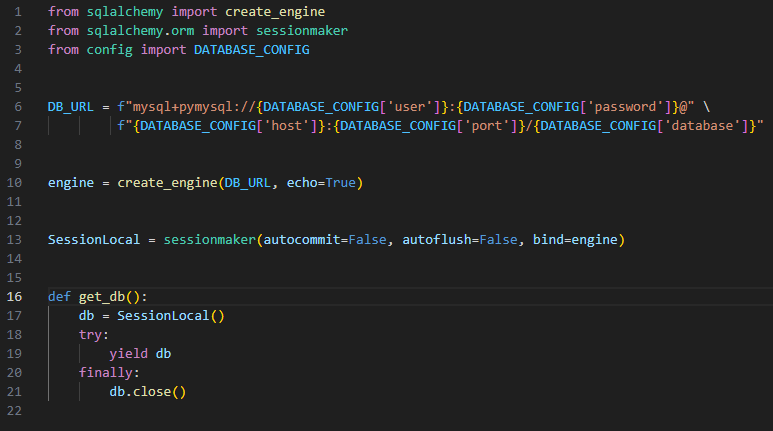
* Ініціалізує FastAPI
* Підключає маршрути (ендпоінти)
* Запускає сервер

****

### **2. Робота з БД (database/db.py)**

📌 **Що робить:**

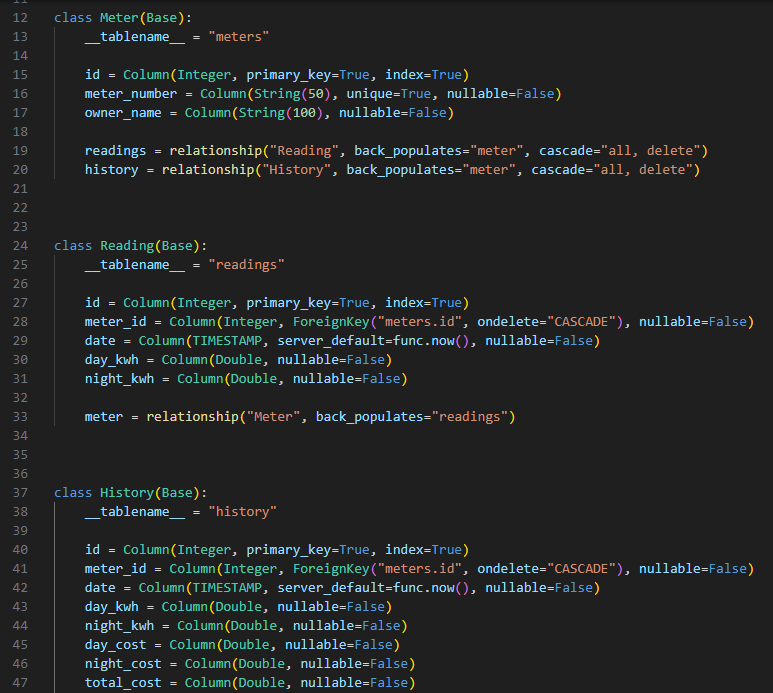
* Налаштовує підключення до MySQL
* Використовує **SQLAlchemy ORM**

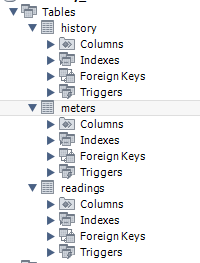


### **3. Опис таблиць у БД (models.py)**

**Що робить:**

* Створює таблиці meters, readings, history
* Зв’язує лічильники з історією показань

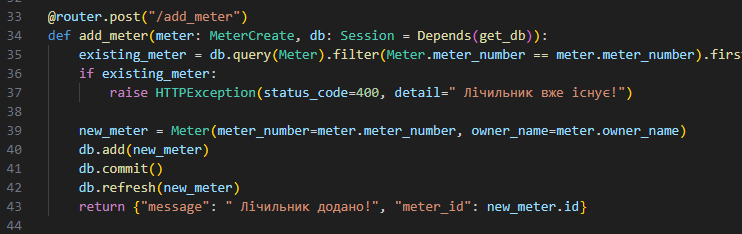


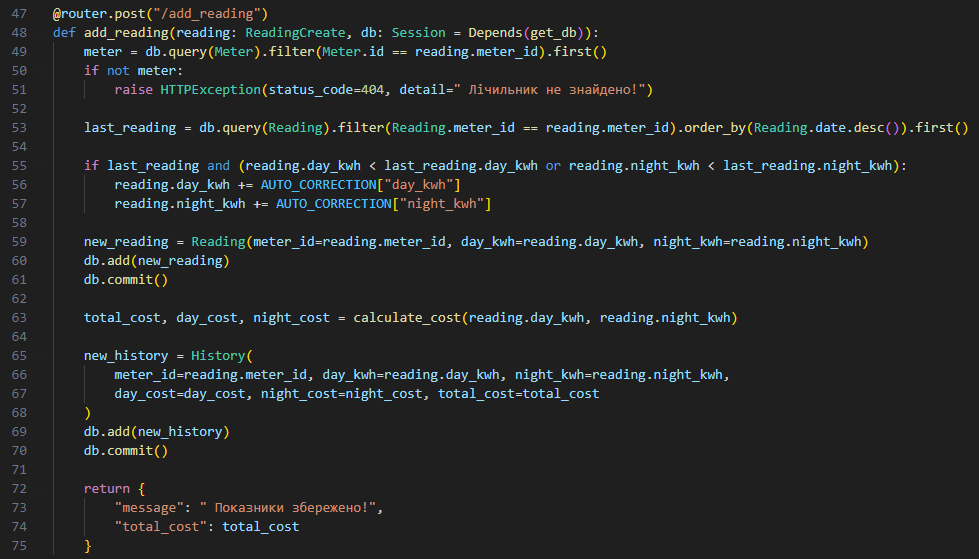


### **4. API-ендпоінти (routes.py)**

**Що робить:**

* Додає лічильник
* Додає показники
* Отримує історію
* Оновлює тарифи
* Експортує дані в CSV



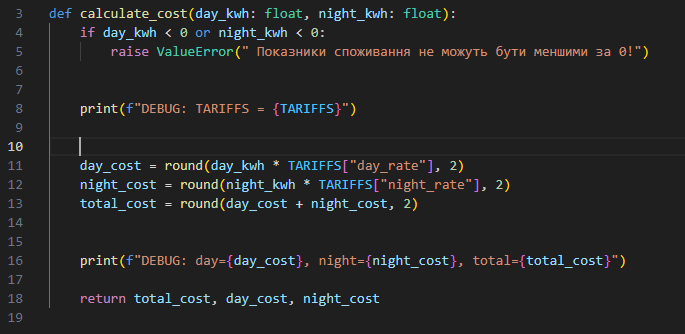




### **5. Розрахунок вартості електроенергії (calculator.py)**

**Що робить:**

* Обчислює **вартість денного та нічного споживання**
* Використовує **округлення** до 2-х знаків



### **6. Консольний інтерфейс (cli.py)**

**Що робить:**

* Користувач може вводити **номери команд**
* Використовує **argparse** для взаємодії



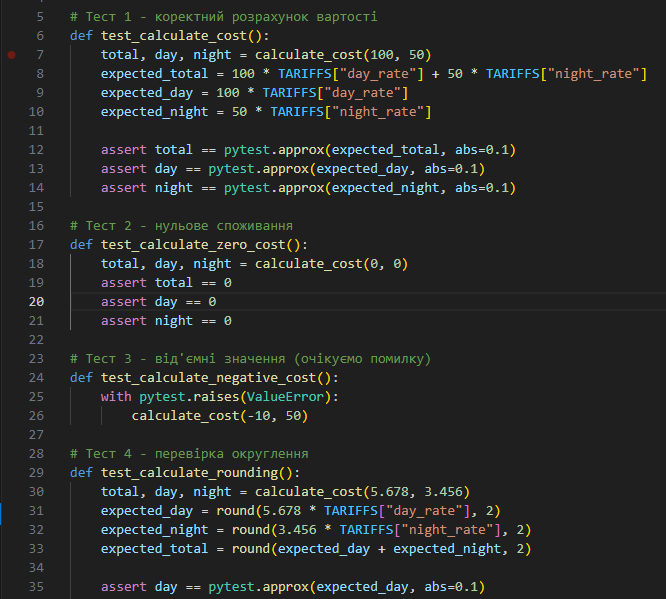
### 

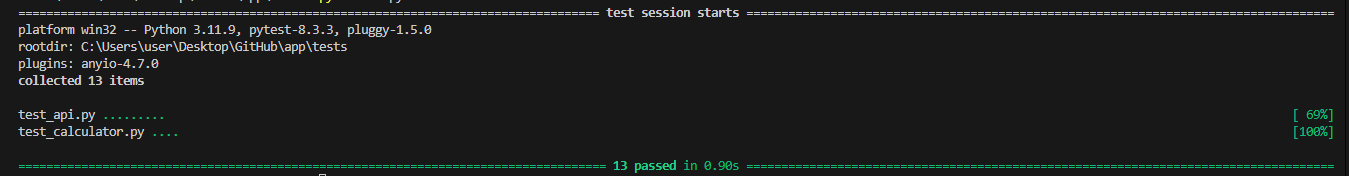
### **7. Автоматичне тестування (pytest, tests/)**

📌 **Що робить:**

* Перевіряє роботу API
* Тестує розрахунок тарифів



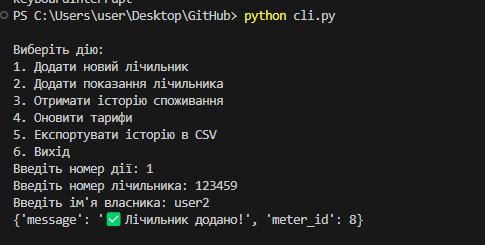


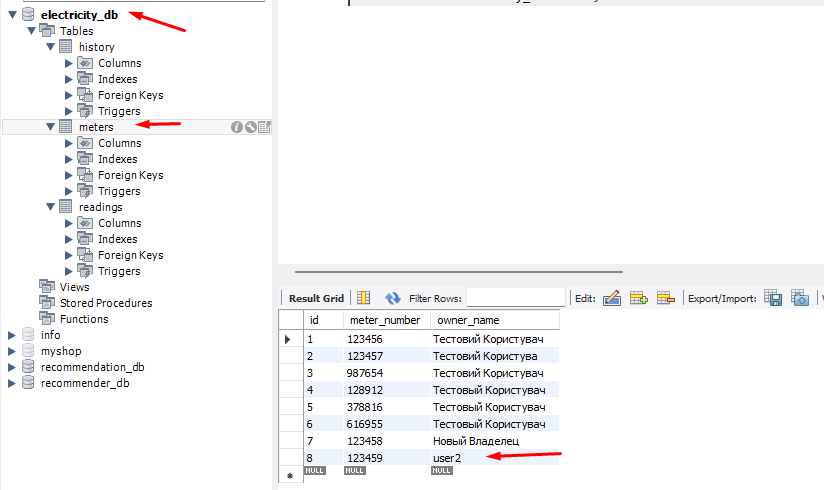


**13/13 тестів успішні**, перевірено точність розрахунків.

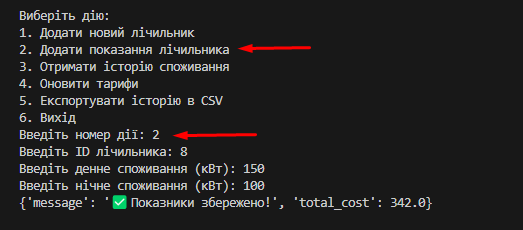
**Висновок**

**1 - Додаємо новий лічильник**

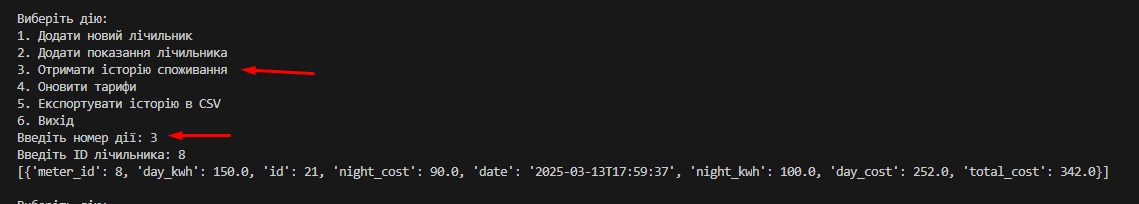




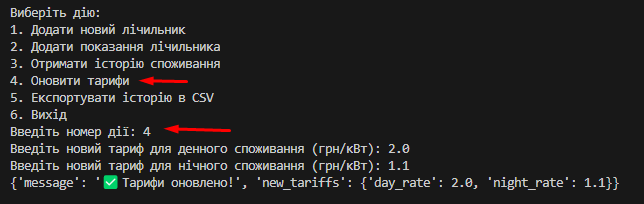
**2 - Додаємо показання**

****

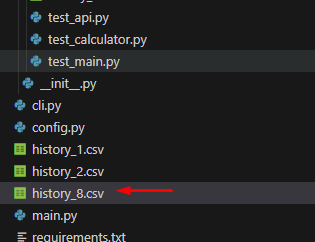
**3 - Отримуємо історію**

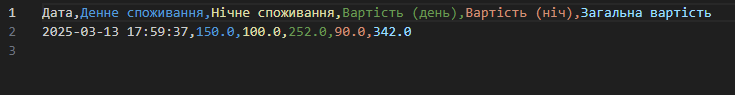
****

**4 - Оновлюємо тарифи**

****

**5 - Експортуємо історію**

****

****