ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного забезпечення термінів обслуговування домашньої техніки та інженерних систем»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Влада ЛЕНДА
(підпис)

Виконала: здобувачка вищої освіти групи ПД-42
Влада ЛЕНДА

Керівник: Тимур ДОВЖЕНКО
к.т.н.

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти Бакалавр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Ленді Владі Тарасівні

1. Тема кваліфікаційної роботи: «Розробка програмного забезпечення термінів обслуговування домашньої техніки та інженерних систем»
керівник кваліфікаційної роботи к.т.н, доцент кафедри ІПЗ Тимур ДОВЖЕНКО, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: нормативні документи та довідкові матеріали щодо термінів та регламентів обслуговування побутової техніки та інженерних систем; технічна документація щодо використання мови програмування Python; офіційна документація бібліотек CustomTkinter для створення графічного інтерфейсу та SQLite для організації локального збереження даних.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області та постановка задачі розробки програмного забезпечення для обліку термінів обслуговування домашньої техніки та інженерних систем.
2. Обґрунтування вибору інструментальних засобів та технологій розробки програмного забезпечення (Python, CustomTkinter, SQLite).
3. Проєктування структури застосунку: функціональні модулі, моделі даних, інтерфейс користувача.

4. Реалізація програмного забезпечення та опис його функціональни можливостей.
5. Тестування програмного забезпечення та аналіз результатів.
6. Висновки та перспективи подальшого розвитку проєкту.

5. Перелік графічного матеріалу: *презентація*
1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Алгоритм роботи застосунку.
6. Діаграма класів.
7. Екранні форми.
8. Апробація результатів дослідження
6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд існуючих алгоритмів обробки тексту та методів і технологій побудови тематичного словника	14.03-17.04.2025	
4	Проектування застосунку для автоматизованої побудови тематичного словника навчальної дисципліни	18.04.-19.04.2025	
5	Програмна реалізація застосунку	20.04-26.04.2025	
6	Тестування застосунку	27.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувачка вищої освіти _____ Влада ЛЕНДА
(підпис)

Керівник
кваліфікаційної роботи _____ Тимур ДОВЖЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 62 стор., 1 табл., 7 рис., 20 джерел.

Мета роботи — розробити зручне настільне програмне забезпечення для ведення обліку технічного обслуговування побутових приладів та інженерних систем з використанням мови Python та бібліотеки customtkinter.

Об’єкт дослідження — процес організації та контролю обслуговування домашньої техніки й інженерних систем у побуті.

Предмет дослідження — настільний застосунок для обліку, нагадування та ведення журналу технічного обслуговування побутових пристроїв.

Короткий зміст роботи: у роботі проаналізовано наявні способи фіксації та моніторингу термінів технічного обслуговування. Розроблено структуру та архітектуру настільного застосунку, що базується на мові Python та графічній бібліотеці customtkinter. Реалізовано функціонал додавання пристроїв, створення записів про планові роботи, автоматичних нагадувань і ведення журналу. Застосунок працює з базою даних SQLite та забезпечує зручну взаємодію з користувачем завдяки сучасному інтерфейсу. Сфера використання — домогосподарства, технічні фахівці, малий бізнес.

КЛЮЧОВІ СЛОВА: CUSTOMTKINTER, PYTHON, ТЕХНІЧНЕ ОБСЛУГОВУВАННЯ, НАГАДУВАННЯ, ЖУРНАЛ, SQLITE, НАСТІЛЬНИЙ ЗАСТОСУНОК

ЗМІСТ

ВСТУП

Сучасне життя неможливо уявити без широкого спектру побутової техніки та складних інженерних систем, які стали невід'ємною частиною наших домівок. Від кухонних приладів, таких як холодильники та посудомийні машини, до критично важливих систем життєзабезпечення – газових котлів, водонагрівачів, систем кондиціонування та вентиляції, фільтрів для очищення води – усі ці пристрої суттєво підвищують рівень комфорту, забезпечують безпеку та сприяють ефективному веденню домашнього господарства. Однак, паралельно зі зростанням кількості та складності цієї техніки, зростає і необхідність у її належному догляді та своєчасному технічному обслуговуванні.

Регулярне обслуговування є не просто рекомендованою практикою, а критично важливим фактором для забезпечення довготривалої та безперебійної роботи цих пристроїв. Воно дозволяє не тільки продовжити термін експлуатації техніки, запобігаючи її передчасному виходу з ладу, але й гарантує її безпечну роботу. Нехтування регламентними процедурами, такими як чистка фільтрів, перевірка з'єднань, діагностика компонентів, може призвести до цілої низки негативних наслідків. Серед них – зниження енергоефективності (що веде до збільшення комунальних платежів), погіршення якості роботи (наприклад, недостатнє охолодження кондиціонера або неефективний нагрів води), раптові поломки, що потребують дороговартісного екстреного ремонту, а в найгірших випадках – до аварійних ситуацій, таких як витік газу, затоплення, коротке замикання або навіть пожежа, що становить пряму загрозу життю, здоров'ю та майну мешканців. Крім того, відсутність документально підтвердженого обслуговування може призвести до втрати гарантії від виробника.

На жаль, у ритмі сучасного життя, перевантаженого інформацією та завданнями, власники житла часто забувають про необхідність проведення планового технічного обслуговування для кожного окремого пристрою. Різні прилади мають різні сервісні інтервали та специфічні вимоги до обслуговування,

що ще більше ускладнює процес контролю. Традиційні методи відстеження, такі як записи в блокнотах, позначки в календарях або сподівання на власну пам'ять, виявляються неефективними та ненадійними. Спроби виробників інтегрувати нагадування безпосередньо в пристрої або мобільні додатки часто є фрагментарними та не охоплюють усю сукупність домашньої техніки та інженерних систем.

У зв’язку з цим виникає гостра та актуальна потреба у створенні спеціалізованого програмного забезпечення. Така система має на меті централізовано та зручно для користувача вирішувати завдання моніторингу стану техніки, автоматичного нагадування про терміни наступного обслуговування, ведення детального обліку виконаних сервісних робіт (включаючи дату, опис робіт, вартість, виконавця), а також зберігання супутньої документації (гарантійні талони, інструкції, чеки, акти виконаних робіт).

Метою даної дипломної роботи є розробка універсального програмного забезпечення, призначеного для моніторингу стану та організації процесу обслуговування побутової техніки та інженерних систем у домашніх умовах. Досягнення цієї мети передбачає вирішення наступних **основних задач**:

1. Провести глибокий аналіз предметної області: дослідити типові види побутової техніки та інженерних систем, їх регламенти обслуговування, поширені проблеми та потреби користувачів.

2. Проаналізувати існуючі аналогічні програмні рішення (якщо такі є), виявити їх переваги та недоліки.

3. Сформулювати детальні функціональні та нефункціональні вимоги до розроблюваної системи, включаючи вимоги до інтерфейсу користувача, безпеки даних та продуктивності.

4. Обрати оптимальну архітектуру програмного забезпечення, що забезпечить гнучкість, масштабованість та можливість мультиплатформенного розгортання.

5. Розробити інтуїтивно зрозумілий та зручний інтерфейс користувача (UI/UX).

6. Реалізувати основний функціонал системи: додавання/редагування пристроїв, встановлення графіків обслуговування, систему нагадувань, журнал сервісних записів, сховище документів.

7. Провести тестування розробленого ПЗ для забезпечення його стабільності та відповідності вимогам.

Об'єктом дослідження виступає комплексний процес організації та виконання технічного обслуговування різноманітної побутової техніки та інженерних систем у житлових приміщеннях. **Предметом дослідження** є сучасні методи, технології та інструменти розробки програмного забезпечення, що застосовуються для автоматизації та оптимізації процесу моніторингу та управління обслуговуванням технічних пристроїв.

Ключовими характеристиками розроблюваної системи мають стати простота у використанні навіть для недосвідчених користувачів, надійність зберігання даних та роботи системи сповіщень, а також **мультиплатформенність**. Можливість доступу до системи як зі стаціонарного комп'ютера (для зручного введення даних чи перегляду історії), так і з мобільних пристроїв (для швидкого доступу до інформації та отримання сповіщень "на ходу") значно підвищить її практичну цінність. У перспективі, функціонал системи може бути розширений шляхом інтеграції з платформами «розумного» будинку для автоматичного отримання даних про стан пристроїв (якщо це підтримується технікою), підключення до хмарних сервісів для резервного копіювання та синхронізації даних між пристроями, або навіть інтеграції з базами даних сервісних центрів для полегшення пошуку кваліфікованих виконавців.

Таким чином, дана дипломна робота спрямована на створення практично корисного інструменту, який допоможе власникам техніки підвищити рівень безпеки та комфорту свого житла, оптимізувати витрати на експлуатацію та ремонт обладнання, і зберегти його працездатність на довгі роки.

1 АНАЛІТИЧНИЙ РОЗДІЛ

1.1 Аналіз предметної області

Сучасне домогосподарство є складною екосистемою, функціонування якої значною мірою залежить від справної та ефективної роботи численної побутової техніки та інтегрованих інженерних систем. Ці пристрої та системи не тільки забезпечують базовий комфорт та зручності, але й виконують критично важливі функції, пов'язані з безпекою житла, здоров'ям мешканців та раціональним використанням ресурсів. Однак, для підтримки їхньої надійної, безпечної та економічної роботи протягом усього терміну служби, необхідне регулярне та кваліфіковане технічне обслуговування. Предметна область даного дослідження охоплює саме цей аспект – процес управління технічним обслуговуванням домашнього обладнання.

1.1.1 Класифікація та Характеристики Обладнання, що Потребує Обслуговування

Спектр пристроїв, що вимагають періодичної уваги, надзвичайно широкий і може бути умовно класифікований за функціональним призначенням:

1. **Системи опалення та гарячого водопостачання:**

1.1. Газові котли (конвекційні, конденсаційні): це одні з найважливіших і потенційно небезпечних пристроїв. Регламентне обслуговування (зазвичай щорічне перед опалювальним сезоном) включає перевірку герметичності газової арматури, чистку теплообмінника та пальника, аналіз продуктів згоряння, перевірку автоматики безпеки (датчики тяги, перегріву, тиску), налаштування тиску газу.

1.2. Електричні котли: перевірка стану ТЕНів, контакторів, запобіжної арматури, чистка від накипу (залежно від жорсткості води).

1.3. Бойлери (накопичувальні водонагрівачі): щорічна (або раз на 2 роки, залежно від виробника та якості води) перевірка та заміна магнієвого аноду для

захисту бака від корозії, промивка бака від осаду та накипу, перевірка запобіжного клапана.

1.4. Проточні водонагрівачі (газові колонки, електричні): чистка теплообмінника, перевірка датчиків та автоматики, видалення накипу.

2. **Системи кондиціонування та вентиляції:**

2.1. Кондиціонери (спліт-системи, мобільні, віконні): чистка або заміна фільтрів внутрішнього блоку (часто щомісяця під час сезону використання), антибактеріальна обробка випарника, чистка теплообмінника зовнішнього блоку (раз на рік), перевірка тиску холодоагенту та герметичності системи, чистка дренажної системи.

2.2. Вентиляційні установки (рекуператори, припливно-витяжні системи): Заміна або чистка повітряних фільтрів (кожні 3-6 місяців), чистка теплообмінника (рекуператора), перевірка вентиляторів та автоматики.

3. **Системи очищення води:**

3.1. Магістральні фільтри механічної очистки: заміна картриджів (інтервал залежить від типу картриджа та забрудненості води, від місяця до півроку).

3.2. Системи пом'якшення води: регулярне поповнення солі для регенерації іонообмінної смоли, періодична перевірка стану смоли та налаштувань клапана управління.

3.3. Системи зворотного осмосу: заміна префільтрів (кожні 3-6 місяців), заміна мембрани (раз на 1-3 роки), заміна постфільтра та мінералізатора (кожні 6-12 місяців), дезінфекція системи.

4. **Велика побутова техніка:**

4.1. Пральні машини: регулярна чистка фільтра зливного насоса, чистка дозатора миючих засобів, періодичний запуск циклу очищення барабана зі спеціальними засобами, перевірка стану заливного та зливного шлангів.

4.2. Посудомийні машини: чистка фільтрів грубої та тонкої очистки, чистка розпилювачів, періодичне використання засобів для видалення жиру та накипу, перевірка рівня солі та ополіскувача.

4.3. Холодильники та морозильні камери: чистка конденсатора (радіатора на

задній стінці або знизу) від пилу (раз на 6-12 місяців) для забезпечення ефективного тепловідведення, перевірка ущільнювачів дверей, розморожування (для систем без No Frost), заміна фільтра для води (в моделях з льодогенератором/диспенсером).

4.4. Сушильні машини: очищення фільтра для ворсу (після кожного використання!), очищення теплообмінника/конденсатора (згідно інструкції). Нехтування цим є поширеною причиною пожеж.

5. **Системи безпеки:**

5.1. Датчики диму та чадного газу (CO): тестування працездатності (щомісяця), заміна батарейок (щорічно або за сигналом датчика), повна заміна пристрою (кожні 7-10 років, оскільки сенсори деградують).

5.2. Системи охоронної сигналізації: перевірка акумуляторів резервного живлення, тестування датчиків руху, відкриття, сирен.

5.3. Системи пожежогасіння (якщо є): перевірка тиску в балонах, терміну придатності вогнегасної речовини.

6. **Інше обладнання:**

6.1. Кухонні витяжки: чистка або заміна жирових фільтрів, заміна вугільних фільтрів (в рециркуляційних моделях).

6.2. Подрібнювачі харчових відходів (диспоузери): періодична чистка спеціальними засобами або за допомогою льоду та лимону.

1.1.2 Регламенти та Значення Технічного Обслуговування

Для кожного типу пристрою виробником встановлюється регламент технічного обслуговування, детально описаний в інструкції з експлуатації. Ці регламенти базуються на інженерних розрахунках, випробуваннях та досвіді експлуатації, і їх дотримання є ключовим для досягнення наступних цілей:

1. Безпека - своєчасне виявлення потенційно небезпечних несправностей (витік газу, ризик ураження струмом, негерметичність водяних систем, ризик пожежі) є першочерговим завданням обслуговування багатьох систем.

2. Ефективність - забруднені фільтри, теплообмінники, накип на нагрівальних елементах суттєво знижують ККД пристроїв, що веде до значного

перевитрати електроенергії, газу чи води. Регулярна чистка та налаштування відновлюють паспортні показники ефективності.

3. Надійність та довговічність - обслуговування дозволяє виявити та усунути дрібні несправності до того, як вони призведуть до серйозної поломки основних вузлів (наприклад, заміна аноду в бойлері запобігає проржавінню бака, чистка конденсатора холодильника зменшує навантаження на компресор). Це значно подовжує термін служби техніки.

4. Комфорт - підтримка належного стану техніки гарантує стабільну роботу (наприклад, постійна температура опалення, якісне охолодження кондиціонером, чиста вода з фільтра).

5. Збереження гарантії - багато виробників вимагають документального підтвердження проходження планового ТО для виконання гарантійних зобов'язань.

1.1.3 Наслідки Нехтування Технічним Обслуговуванням

Ігнорування або несвоєчасне виконання регламентних робіт може мати серйозні та різноманітні негативні наслідки:

1. **Прямі загрози життю та здоров'ю:** отруєння чадним газом від несправних котлів чи колонок, пожежі через несправну електропроводку або засмічені вентиляційні канали сушильних машин, ураження електричним струмом, вибухи газового обладнання.

2. **Значні фінансові втрати:**

2.1. Збільшення комунальних платежів через знижену енергоефективність обладнання.

2.2. Дорогий аварійний ремонт: вартість екстреного виклику майстра та ремонту серйозної поломки значно вища за вартість планового ТО.

2.3. Передчасна заміна обладнання: техніка, що не обслуговується належним чином, виходить з ладу значно раніше встановленого терміну служби.

2.4. Побічні збитки - затоплення приміщень через прорив шлангів або корозію бака бойлера, псування майна внаслідок пожежі.

3. **Зниження якості життя:** нестабільна робота опалення взимку, відсутність

гарячої води, погане кондиціонування влітку, неприємні запахи, зниження якості питної води, неякісне прання чи миття посуду.

4. **Екологічні наслідки:** надмірне споживання енергоресурсів, можливі витоки шкідливих речовин (наприклад, холодоагенту).

1.1.4 Аналіз Існуючих Практик Управління Обслуговуванням

Незважаючи на очевидну важливість проблеми, більшість користувачів не мають системного підходу до управління технічним обслуговуванням свого домашнього обладнання. Найпоширеніші практики включають:

1. **Паперові записи:** нотатки в блокнотах, на стікерах, позначки в паперових календарях. Цей метод вкрай ненадійний через ризик втрати записів, їхню неструктурованість та відсутність автоматичних нагадувань.

2. **Використання стандартних цифрових календарів:** створення подій-нагадувань в Google Calendar, Outlook Calendar тощо. Це краще за паперові нотатки, але все ще не є спеціалізованим рішенням. Відсутня можливість зручно зберігати детальну історію робіт, документацію, специфічні дані про пристрій. Керування великою кількістю різних нагадувань може бути незручним.

3. **Сподівання на пам'ять:** найменш надійний метод, який майже гарантовано призводить до пропуску термінів обслуговування.

4. **Стікери від сервісних компаній:** часто після обслуговування майстер залишає наліпку з датою наступного ТО. Це стосується лише одного пристрою та однієї компанії, не даючи загальної картини.

5. **Реактивний підхід:** звернення до сервісної служби лише тоді, коли пристрій вже вийшов з ладу або почав працювати незадовільно. Це найдорожчий та найнебезпечніший підхід.

Ці методи є неефективними в умовах зростаючої кількості та складності домашньої техніки. Вони не забезпечують централізованого контролю, систематичності, надійних нагадувань та зручного доступу до історії обслуговування та документації.

1.1.5 Потреба в Спеціалізованому Програмному Забезпеченні

Тенденція до цифровізації всіх аспектів життя, включаючи управління домом ("розумний дім"), а також виявлені недоліки існуючих практик управління ТО, чітко вказують на доцільність та актуальність розробки спеціалізованого програмного забезпечення. Така система повинна комплексно вирішувати завдання, пов'язані з обслуговуванням техніки, забезпечуючи користувачеві такі можливості:

1. **Централізований реєстр:** створення єдиної бази даних усієї техніки, що потребує обслуговування, з можливістю додавання детальних характеристик (виробник, модель, серійний номер, дата встановлення, термін гарантії тощо).

2. **Гнучке планування ТО:** можливість встановлення індивідуальних графіків обслуговування для кожного пристрою (на основі рекомендацій виробника або власних потреб) з різними інтервалами (дні, тижні, місяці, роки).

3. **Автоматизовані нагадування:** своєчасні та гнучко налаштовувані сповіщення про наближення терміну планового обслуговування через різні канали (сповіщення в додатку, email, push-повідомлення).

4. **Ведення історії обслуговування:** детальний журнал усіх виконаних робіт (дата, опис робіт, використані запчастини/матеріали, вартість, виконавець, коментарі). Це дозволяє аналізувати витрати та ефективність обслуговування.

5. **Цифрове сховище документів:** можливість прикріплювати до записів про пристрої та обслуговування скановані копії або фотографії інструкцій, гарантійних талонів, чеків, актів виконаних робіт.

6. **Резервне копіювання та синхронізація:** забезпечення збереження даних та можливості доступу до них з різних пристроїв (наприклад, ПК та смартфон) через хмарні сервіси.

1.2 Огляд аналогів та існуючих рішень

Для повного розуміння потреби у розробці нового програмного забезпечення для моніторингу та нагадування про обслуговування побутової техніки, необхідно

проаналізувати існуючі на ринку інструменти та підходи, які користувачі можуть застосовувати для вирішення подібних завдань. Аналіз проводитиметься за критеріями: функціональність для ведення обліку та історії обслуговування, зручність використання, гнучкість налаштувань, мультиплатформеність, вартість та специфічні обмеження.

1. Загальнодоступні цифрові календарі (Google Calendar, Microsoft Outlook Calendar, Apple Calendar та ін.)

1. **Опис** -це універсальні інструменти для планування часу, які широко використовуються для особистих та робочих завдань. Користувачі можуть створювати одноразові або повторювані події з нагадуваннями, що теоретично дозволяє відстежувати дати обслуговування техніки.

2. **Переваги:**

2.1. Переважно безкоштовні та вже встановлені або легко доступні на більшості пристроїв.

2.2. Більшість користувачів вже вміють працювати з календарями.

2.3. Добре реалізовані функції сповіщень та автоматична синхронізація між пристроями (ПК, смартфон, планшет) через хмару.

2.4. Зазвичай доступні через веб-інтерфейс та нативні додатки для різних ОС.

3. **Недоліки:**

3.1. Календарі не призначені для ведення бази даних обладнання. Неможливо зручно зберігати інформацію про пристрій (марка, модель, серійний номер, дата купівлі/встановлення, гарантія).

3.2. Неможливо вести структурований журнал виконаних робіт (що робилося, коли, ким, вартість, використані матеріали). Кожна подія – ізольований запис.

3.3. Не можна прикріпити до запису інструкцію, гарантійний талон, чек чи акт виконаних робіт.

3.4. При великій кількості техніки календар швидко заповнюється однотипними подіями, що ускладнює навігацію та візуальне сприйняття основних подій користувача.

3.5. Немає можливості аналізувати витрати на обслуговування, генерувати звіти, відстежувати статус гарантії.

2. Платформи для автоматизації "розумного будинку" (Home Assistant, OpenHAB, Hubitat та ін.)

1. **Опис -** це потужні, часто з відкритим кодом, системи, призначені для інтеграції та керування різними пристроями "розумного будинку". Вони дозволяють створювати складні сценарії автоматизації. Теоретично, в рамках таких систем можна налаштувати моніторинг стану деяких пристроїв та створити нагадування про обслуговування, особливо якщо техніка підтримує відповідні протоколи.

2. **Переваги:**

2.1. Майже необмежені можливості для кастомізації та створення складних автоматизацій.

2.2. Можливість отримувати реальні дані від деяких пристроїв (лічильники роботи, статус фільтрів) і використовувати їх для тригерів нагадувань.

2.3. Велика кількість готових інтеграцій, доповнень та форумів підтримки (наприклад, для Home Assistant).

3. **Недоліки:**

3.1. Налаштування та підтримка таких систем вимагає значних технічних знань та часу. Це не рішення для пересічного користувача.

3.2. Зазвичай вимагає наявності постійно працюючого серверного обладнання (одноплатний комп'ютер типу Raspberry Pi, NAS-сервер або віртуальна машина).

3.3. Основне призначення – керування та автоматизація, а не структуроване ведення історії обслуговування, витрат та документації. Реалізація цих функцій потребує додаткових налаштувань або сторонніх доповнень, інтерфейс яких часто не є інтуїтивним для таких завдань.

3.4. Не охоплює "нерозумну" техніку: більшість побутових приладів та інженерних систем не мають можливості інтеграції з такими платформами.

3. Універсальні бази даних та органайзери / Спеціалізовані мобільні

додатки (Notion, Airtable, Evernote, Memento Database, My Stuff Organizer, Centriq, Home Inventory, Binnie's Home Maintenance та ін.)

1. **Опи - ця** категорія включає як гнучкі інструменти для створення власних баз даних (Notion, Airtable), так і спеціалізовані мобільні додатки для інвентаризації майна або безпосередньо для відстеження обслуговування.

2. **Переваги:**

2.1. Можливість створити власну структуру даних під свої потреби.

2.2. Часто мають готовий інтерфейс з полями для внесення даних про техніку (марка, модель, фото, дата покупки), функції нагадувань, іноді можливість прикріплення файлів.

2.3. Зручний доступ та керування зі смартфона.

3. **Недоліки:**

3.1. Необхідність самостійної структуризації (для універсальних БД): вимагає від користувача проектування таблиць, зв'язків, налаштування нагадувань, що може бути складно.

3.2. Обмежена функціональність (для багатьох мобільних додатків): можуть бути занадто простими, не мати достатньо гнучких налаштувань нагадувань, можливостей для детального обліку витрат чи ведення історії.

3.3. Багато додатків мають обмежену безкоштовну версію та вимагають підписки для доступу до повного функціоналу, синхронізації або зняття обмежень на кількість записів.

3.4. Неможливість модифікації, дані зберігаються у пропрієтарному форматі або в хмарі розробника, що створює ризики для приватності та залежність від сервісу.

3.5. Не всі додатки мають якісну українську локалізацію.

3.6. Часто це рішення "mobile-first" або "mobile-only", з обмеженим або відсутнім доступом з настільного комп'ютера.

4. Програмне забезпечення від виробників техніки (Samsung SmartThings, LG ThinQ, Bosch Home Connect, Miele@mobile та ін.)

1. **Опис-** багато великих виробників побутової техніки розробляють власні

мобільні додатки для керування своїми "розумними" пристроями. Деякі з цих додатків включають функції моніторингу стану пристрою та можуть надсилати сповіщення про необхідність певних дій (наприклад, заміни фільтра, запуску циклу самоочищення).

2. **Переваги:**

2.1. Найкраща можлива інтеграція з конкретними моделями техніки даного бренду.

2.2. Можливість отримувати точну інформацію про стан пристрою, коди помилок, статистику використання, що може бути корисним для діагностики та планування ТО.

3. **Недоліки:**

3.1. Фрагментація та вендор-лок (ключовий недолік): кожен додаток працює лише з технікою свого бренду. Власник техніки різних виробників змушений використовувати кілька різних додатків, що повністю нівелює ідею централізованого обліку.

3.2. Основний фокус цих додатків – керування пристроєм. Функції, пов'язані з обслуговуванням, часто обмежені базовими нагадуваннями і не дозволяють вести повноцінний журнал робіт, витрат, зберігати документи для всіх видів обслуговування.

3.3. Далеко не вся техніка є "розумною", а старіші моделі чи пристрої інших брендів взагалі не можуть бути додані в такі додатки.

3.4. "App Fatigue": необхідність встановлювати та розбиратися з ще одним додатком для кожної марки техніки.

5. Електронні таблиці (Microsoft Excel, Google Sheets, LibreOffice Calc)

1. **Опис -** дуже поширений DIY-метод для ведення різноманітних обліків, включаючи домашню техніку та її обслуговування.

2. **Переваги:**

2.1. Широко розповсюджені, багато хто має базові навички роботи.

2.2. Користувач може створити будь-яку структуру таблиці для зберігання потрібної інформації.

2.3. Можна підраховувати витрати, терміни тощо за допомогою формул.

3. **Недоліки:**

3.1. Потребує ручної перевірки дат або складних скриптів для реалізації сповіщень.

3.2. Вимагає значної ручної роботи для внесення даних, підтримки актуальності, створення звітів.

3.3. Робота з таблицями, особливо на мобільних пристроях, менш зручна, ніж зі спеціалізованим додатком.

3.4. Легко випадково видалити дані, порушити формули чи структуру таблиці.

3.5. Відсутність інтегрованого зберігання документів.

1.3 Постановка задачі

На основі проведеного аналізу предметної області, який виявив критичну важливість регулярного обслуговування домашньої техніки та інженерних систем, та огляду існуючих рішень, що підтвердив відсутність універсального та зручного інструменту для комплексного управління цим процесом, формулюється постановка задачі для даної дипломної роботи.

Основна мета роботи: спроектувати та розробити програмне забезпечення (ПЗ) під умовною назвою "HomeTech Manager" (або аналогічною), призначене для централізованого обліку побутової техніки та інженерних систем домогосподарства, а також для планування, відстеження та нагадування про необхідність їх технічного обслуговування. Система має стати надійним помічником для користувача, що дозволить підвищити ефективність експлуатації обладнання, продовжити його термін служби, знизити ризики аварійних ситуацій та оптимізувати пов'язані витрати.

Функціональні вимоги до ПЗ:

Розроблюване програмне забезпечення повинно реалізовувати наступний набір функцій:

1. **Управління реєстром техніки:**

1.1. Додавання нових записів про пристрої/системи.

1.2. Редагування існуючих записів.

1.3. Видалення записів про пристрої.

1.4. Зберігання детальної інформації для кожного пристрою:

1.4.1. Назва (задається користувачем, напр., "Котел на кухні").

1.4.2. Категорія/Тип пристрою (вибір зі списку з можливістю додавання своїх: Котел, Кондиціонер, Фільтр води, Пральна машина тощо).

1.4.3. Виробник (Бренд).

1.4.4. Модель.

1.4.5. Серійний номер (необов'язково).

1.4.6. Дата покупки / введення в експлуатацію.

1.4.7. Термін закінчення гарантії.

1.4.8. Місце встановлення (напр., кухня, ванна, підвал).

1.4.9. Фотографія пристрою (необов'язково).

1.4.10. Додаткові нотатки.

1.5. Можливість перегляду списку всієї техніки з основними характеристиками, сортуванням та фільтрацією (напр., за типом, місцем встановлення).

2. **Планування та моніторинг обслуговування:**

2.1. Можливість визначення одного або декількох типів регламентних робіт для кожного пристрою (напр., "Чистка фільтрів", "Щорічне ТО", "Заміна картриджа").

2.2. Встановлення періодичності для кожного типу робіт (напр., кожні 3 місяці, раз на рік, кожні 10000 годин роботи - якщо є лічильник, або конкретна дата).

2.3. Фіксація дати останнього проведеного обслуговування для кожного типу робіт.

2.4. Автоматичний розрахунок дати наступного планового обслуговування на основі дати останнього ТО та заданої періодичності.

2.5. Візуальне відображення статусу обслуговування (напр., вчасно, скоро наближається, протерміновано).

3. **Система нагадувань:**

3.1. Автоматичне формування сповіщень користувачеві про наближення терміну планового обслуговування (напр., за 7, 14, 30 днів до дати - з можливістю налаштування).

3.2. Сповіщення про протерміновані задачі обслуговування.

3.3. Реалізація механізму сповіщень (на початковому етапі – внутрішні сповіщення в інтерфейсі програми).

4. **Ведення історії обслуговування:**

4.1. Можливість додавання записів про фактично виконані сервісні роботи, прив'язаних до конкретного пристрою та типу робіт.

4.2. Зберігання деталей кожного сервісного запису:

4.2.1. Дата виконання робіт.

4.2.2. Опис виконаних робіт.

4.2.3. Використані запчастини/матеріали (необов'язково).

4.2.4. Вартість робіт та запчастин (необов'язково).

4.2.5. Виконавець (назва сервісної компанії або ім'я майстра, контактні дані - необов'язково).

4.2.6. Нотатки до запису.

4.3. Перегляд хронологічної історії обслуговування для кожного пристрою.

5. **Управління документами:**

5.1. Можливість прикріплення електронних копій документів (скани або фото у форматах PDF, JPG, PNG тощо) до записів про пристрої (інструкції, гарантійні талони) та до записів про сервісні роботи (акти виконаних робіт, чеки).

5.2. Можливість перегляду та завантаження прикріплених документів.

6. **Інтерфейс користувача (UI/UX):**

6.1. Розробка інтуїтивно зрозумілого, логічного та візуально приємного графічного інтерфейсу користувача.

6.2. Забезпечення простоти навігації та зручності введення/перегляду даних.

6.3. Обов'язкова підтримка української мови інтерфейсу.

Нефункціональні вимоги:

1. Система повинна коректно зберігати дані, точно розраховувати дати та надійно генерувати нагадування. Має бути забезпечена цілісність даних.

2. Інтерфейс має бути відзивчивим, операції збереження та пошуку даних мають виконуватися за прийнятний час, навіть при збільшенні обсягу даних.

3. Архітектура має дозволяти потенційне розширення функціоналу в майбутньому (напр., додавання хмарної синхронізації, інтеграцій).

4. На першому етапі розробки планується створення настільного додатку (Desktop Application) для ОС Windows. У перспективі – розгляд можливості створення кросплатформенного рішення (напр., з використанням фреймворків типу .NET MAUI, Flutter, або як веб-додаток).

5. Дані користувача (реєстр техніки, історія обслуговування, налаштування) будуть зберігатися локально на комп'ютері користувача у файлі бази даних (напр., SQLite).

Основні задачі, що підлягають вирішенню в рамках дипломної роботи:

1. **Деталізація вимог:** провести поглиблений аналіз та формалізацію функціональних та нефункціональних вимог до системи.

2. **Проектування архітектури:** обрати архітектурний підхід (напр., MVC, MVVM), визначити основні модулі системи та взаємозв'язки між ними. Обрати технологічний стек (мова програмування, фреймворки, СУБД).

3. **Проектування бази даних:** розробити логічну та фізичну схему бази даних для зберігання всієї необхідної інформації.

4. **Розробка користувацького інтерфейсу:** створити макети інтерфейсу та реалізувати його програмну частину згідно з принципами UI/UX.

5. **Реалізація бізнес-логіки:** написати програмний код для реалізації основних функцій: управління пристроями, планування та розрахунок дат ТО, ведення історії, робота з документами.

6. **Реалізація системи нагадувань:** розробити механізм генерації та відображення нагадувань в інтерфейсі програми.

7. **Тестування:** провести модульне, інтеграційне та користувацьке тестування для виявлення та виправлення помилок, а також для перевірки зручності використання.

8. **Підготовка документації:** створити супровідну документацію (пояснювальну записку до дипломної роботи, можливо, короткий посібник користувача).

Очікуваний результат:

Результатом виконання дипломної роботи має стати функціональний прототип або готова до використання версія програмного продукту "HomeTech Manager" (для ОС Windows), що реалізує всі зазначені вище основні функціональні вимоги. Продукт повинен дозволяти користувачеві ефективно управляти процесом обслуговування домашньої техніки, систематизувати інформацію, отримувати своєчасні нагадування, тим самим сприяючи підтримці техніки у справному та безпечному стані.

2 ПРОЄКТНИЙ РОЗДІЛ

2.1 Вибір та Обґрунтування Архітектури Програмного Забезпечення

Вибір правильної архітектури є фундаментальним етапом у розробці будь-якого програмного забезпечення. Архітектура визначає структуру системи, взаємодію її компонентів, а також впливає на такі ключові аспекти, як надійність, продуктивність, можливість модифікації та подальшого розвитку ПЗ. Враховуючи цілі та вимоги, сформульовані в попередньому розділі (зокрема, потреба у надійному зберіганні даних, зручному інтерфейсі, системі нагадувань та офлайн-доступі), було прийнято рішення про розробку **настільного (десктопного) додатку**.

2.1.1 Обґрунтування Вибору Платформи та Базових Технологій

Вибір настільного додатку як цільової платформи на даному етапі розробки зумовлений кількома факторами:

1. **Автономність та Офлайн-доступ:** на відміну від веб-додатків, настільне ПЗ може повноцінно функціонувати без постійного підключення до Інтернету, що є важливим для користувачів, які можуть не завжди мати стабільний зв'язок або бажають зберігати свої дані виключно локально.

2. **Простота Розгортання (для кінцевого користувача):** сучасні інструменти дозволяють запакувати додаток разом з усіма залежностями в єдиний виконуваний файл або простий інсталятор, що спрощує процес встановлення для користувачів без глибоких технічних знань.

3. **Прямий Доступ до Локальних Ресурсів:** настільний додаток легко взаємодіє з локальною файловою системою для зберігання бази даних, прикріплення документів та створення резервних копій.

4. **Контроль над Середовищем:** розробка ведеться для більш прогнозованого середовища порівняно з різноманіттям веб-браузерів та їх версій.

Як основні технології для реалізації було обрано наступний стек:

1. **Мова програмування: Python 3.x**

1.1. Швидкість Розробки: Python є високорівневою мовою з лаконічним та читабельним синтаксисом, що дозволяє швидше реалізовувати функціонал порівняно з мовами нижчого рівня.

1.2. Багата Стандартна Бібліотека: включає готові модулі для роботи з файлами (os, shutil), датами та часом (datetime, calendar), базами даних (sqlite3), архівами (zip), форматами даних (csv) та багато іншого, що покриває значну частину потреб проекту без необхідності залучення сторонніх бібліотек.

1.3. Велика Екосистема: наявність величезної кількості сторонніх бібліотек та фреймворків на випадок потреби розширення функціоналу в майбутньому.

1.4. Кросплатформеність: сам Python є кросплатформенним, що спрощує потенційне портування додатку на інші ОС (Linux, macOS) у майбутньому, хоча вибір GUI-бібліотеки також впливає на це.

2. **Графічний Інтерфейс Користувача (GUI): Tkinter**

2.1. Стандартна Бібліотека Python: Tkinter входить до стандартної поставки Python, що усуває потребу в інсталяції додаткових залежностей для базового GUI та спрощує процес розповсюдження додатку.

2.2. Простота та Швидкість Розробки UI: дозволяє відносно швидко створювати стандартні елементи інтерфейсу (вікна, кнопки, поля вводу, списки, меню), що є достатнім для реалізації функціоналу програми обліку та нагадувань.

2.3. Стабільність та Достатня Продуктивність: для додатку, який не потребує складної графіки чи високої частоти оновлення інтерфейсу, Tkinter забезпечує прийнятну продуктивність та стабільність.

2.4. Альтернативи: розглядалися також PyQt/PySide (більш функціональні, але складніші та з ліцензійними особливостями) та Kivy (орієнтований на сучасні "touch" інтерфейси). Tkinter було обрано як прагматичний компроміс між функціональністю та простотою для цілей даної дипломної роботи.

3. **Система Управління Базами Даних (СУБД): SQLite 3**

3.1. Вбудована та Серверлес: SQLite є бібліотекою, що реалізує СУБД, яка

зберігає всю базу даних в одному файлі на диску. Вона не потребує окремого серверного процесу, інсталяції чи конфігурації, що ідеально підходить для однокористувацьких настільних додатків.

3.2. Стандартна Підтримка в Python: модуль sqlite3 є частиною стандартної бібліотеки Python, забезпечуючи легку та надійну інтеграцію.

3.3. Транзакційність (ACID): гарантує атомарність, узгодженість, ізольованість та довговічність транзакцій, що забезпечує цілісність даних при збоях.

3.4. Компактність та Продуктивність: дуже ефективна для локальних операцій, забезпечує хорошу швидкість читання та запису для типових завдань додатку.

3.5. Портативність: файл бази даних легко копіювати для резервного копіювання або перенесення на інший комп'ютер.

2.1.2 Вибір Архітектурного Патерну: MVC (Model-View-Controller)

Для структурування коду додатку та забезпечення його гнучкості й підтримуваності було обрано класичний архітектурний патерн **Model-View-Controller (Модель-Представлення-Контролер)**. Цей патерн пропонує розділення відповідальності між трьома основними компонентами:

1. **Model (Модель):** відповідає за дані та бізнес-логіку програми.

1.1. Обов'язки: управління даними (зберігання, вибірка, оновлення, видалення), інкапсуляція стану програми, реалізація правил бізнес-логіки (наприклад, розрахунок наступної дати ТО, валідація даних перед збереженням). У даному проекті Модель буде взаємодіяти з базою даних SQLite. Вона не залежить від Представлення чи Контролера.

1.2. Компоненти: класи даних (напр., Device, MaintenanceTask, ServiceRecord), клас(и) для доступу до даних (Data Access Layer - DAL), що інкапсулює SQL-запити до SQLite.

2. **View (Представлення):** відповідає за візуальне представлення даних користувачеві та відображення елементів керування.

2.1. Обов'язки: відображення даних, отриманих від Контролера, за допомогою віджетів Tkinter (форми, списки, таблиці, діалогові вікна). Передача дій користувача (кліки кнопок, введення тексту) до Контролера для обробки. Не містить бізнес-логіки.

2.2. Компоненти: вікна та віджети Tkinter, організовані в логічні екрани (головне вікно, вікно деталей пристрою, вікно історії, налаштування тощо).

3. **Controller (Контролер):** виступає посередником між Моделлю та Представленням.

3.1. Обов'язки: обробка вхідних даних та подій від користувача, отриманих від Представлення. Виклик відповідних методів Моделі для маніпуляції даними. Отримання даних від Моделі та передача їх у Представлення для відображення. Управління потоком роботи додатку.

3.2. Компоненти: класи або функції, що містять обробники подій (event handlers) від елементів Tkinter. Координує взаємодію, наприклад, при збереженні даних з форми: Контролер отримує дані з полів вводу Представлення, передає їх Моделі для збереження, отримує результат і оновлює Представлення.

Переваги використання MVC у даному проекті:

1. **Розділення Відповідальності:** чітке розмежування логіки даних, логіки представлення та логіки управління спрощує розробку, тестування та розуміння коду.

2. **Підвищена Підтримуваність:** зміни в одному компоненті (наприклад, редизайн інтерфейсу у View) мінімально впливають на інші компоненти (Model, Controller).

3. **Можливість Повторного Використання:** логіка Моделі потенційно може бути використана повторно, якщо в майбутньому буде вирішено створити інший інтерфейс (наприклад, веб).

4. **Покращена Тестованість:** компоненти Моделі та Контролера можна тестувати ізольовано від інтерфейсу користувача.

2.1.3 Основні Модулі Системи

Функціональність програмного забезпечення буде реалізована через наступні логічні модулі, кожен з яких буде містити компоненти Model, View та Controller:

1. **Модуль Управління Пристроями:**

1.1. Відповідальність: дозволяє користувачеві додавати, переглядати, редагувати та видаляти записи про побутову техніку та інженерні системи.

1.2. Компоненти: форми для введення/редагування даних пристрою (View), логіка обробки цих форм (Controller), функції для збереження/вибірки даних пристроїв з БД (Model).

2. **Модуль Планування та Відстеження Обслуговування:**

2.1. Відповідальність: дозволяє визначати типи та періодичність обслуговування для кожного пристрою, фіксувати дати останнього ТО та автоматично розраховувати наступні.

2.2. Компоненти: інтерфейси для налаштування графіків ТО (View), логіка розрахунку дат та оновлення статусів (Controller, Model), зберігання параметрів обслуговування в БД (Model).

3. **Модуль Нагадувань:**

3.1. Відповідальність: періодично (напр., при запуску програми) перевіряє наближення або протермінування дат обслуговування та інформує користувача.

3.2. Компоненти: логіка перевірки дат у БД (Controller, Model), вікно або секція інтерфейсу для відображення нагадувань (View).

4. **Модуль Історії Обслуговування:**

4.1. Відповідальність: дозволяє користувачеві додавати записи про фактично виконані роботи, переглядати історію по кожному пристрою, додавати коментарі та прикріплювати документи.

4.2. Компоненти: форми для введення даних про сервісні події (View), логіка обробки та збереження цих записів (Controller, Model), механізми для роботи з файлами документів (Controller, Model).

5. **Модуль Імпорту/Експорту та Резервного Копіювання:**

5.1. Відповідальність: надає можливість створити резервну копію бази даних

та, можливо, експортувати дані у зовнішні формати (напр., CSV).

5.2. Компоненти: функції інтерфейсу для запуску операцій (View), логіка для копіювання файлу БД або форматування даних для експорту (Controller, Model), використання бібліотек os, shutil, csv, zip.

6. **Модуль Налаштувань:**

6.1. Відповідальність: дозволяє користувачеві налаштовувати параметри програми (напр., інтервали нагадувань, шлях для збереження резервних копій, можливо, мова інтерфейсу - хоча повна локалізація виходить за рамки базової реалізації).

6.2. Компоненти: діалогове вікно налаштувань (View), логіка збереження та застосування налаштувань (Controller, Model - налаштування можуть зберігатися в тій же БД або окремому конфігураційному файлі).

Таблиця 2.1

Приклад модуля налаштувань

Компонент	Технологія / Бібліотека	Призначення та обґрунтування
Мова програмування	Python 3.x	Швидка розробка, читабельність, багата стандартна бібліотека.
Графічний інтерфейс	Tkinter	Стандартна бібліотека GUI, простота, достатня функціональність.
База даних	SQLite 3	Вбудована, серверлес, надійна (ACID), легка інтеграція (sqlite3).
Робота з датами / часом	datetime, calendar	Стандартні модулі для маніпуляцій з датами та розрахунку термінів.

Продовження таблиці 2.1

Приклад модуля налаштувань

Компоненти	Технологія / Бібліотека	Призначення / обгрунтування
Робота з файлами / ОС	os, shutil	Стандартні модулі для доступу до файлової системи (документи, БК).
Експорт даних (приклад)	csv	Стандартний модуль для можливого експорту у форматі CSV.
Архівація (приклад)	zip	Стандартний модуль для створення архівних резервних копій.

2.2 Опис функціоналу системи

Розроблюване програмне забезпечення призначене для спрощення обліку побутової техніки та планування її регулярного обслуговування. Воно має інтуїтивно зрозумілий графічний інтерфейс, розроблений з акцентом на легкість використання для широкого кола користувачів. Структура програми модульна, складається з декількох основних розділів, доступ до яких здійснюється через головне меню або систему вкладок, що дозволяє користувачеві швидко переходити між різними функціональними областями. Кожен розділ відповідає за реалізацію окремої частини загального функціоналу системи.

Основна мета системи: допомогти користувачам організувати інформацію про їхні пристрої, не пропустити важливі терміни обслуговування та мати повну історію проведених робіт, що може продовжити термін служби техніки та допомогти у гарантійних випадках.

Головні функції системи представлені наступними розділами:

1. **Облік пристроїв.** Цей розділ є центральним сховищем інформації про всю

техніку, яку користувач бажає внести в систему. Він надає повний набір інструментів для керування базою даних пристроїв.

1.1. **Додавання нового пристрою:** користувач має можливість легко додати нову одиницю техніки, заповнивши відповідну форму. Обов'язкові та опціональні поля включають:

1.1.1. **Назва:** коротке, зрозуміле ім'я для ідентифікації пристрою (наприклад, "Пральна машина Bosch Serie 6", "Телевізор Samsung QLED").

1.1.2. **Категорія:** дозволяє групувати техніку за типом (наприклад, "Кухонна техніка", "Кліматичні системи", "Електроніка", "Інструменти", "Опалення", "Фільтри для води"). Система може пропонувати стандартні категорії з можливістю додавання власних. Це поле корисне для фільтрації та організації даних.

1.1.3. **Виробник/Бренд (опціонально):** наприклад, "LG", "Samsung", "Bosch", "Philips".

1.1.4. **Модель (опціонально):** детальна назва моделі, важлива для пошуку інструкцій чи запчастин.

1.1.5. **Дата покупки:** дата придбання пристрою. Важлива для відстеження гарантійного терміну (якщо ця функція буде реалізована).

1.1.6. **Серійний номер:** унікальний ідентифікатор пристрою, часто необхідний для гарантійного обслуговування або технічної підтримки. Можлива валідація формату серійного номера для певних категорій.

1.1.7. **Місце розташування:** дозволяє вказати, де фізично знаходиться пристрій (наприклад, "Кухня", "Ванна кімната", "Вітальня", "Гараж", "Підвал"). Це поле також може бути використане для фільтрації.

1.1.8. **Фото пристрою (опціонально):** можливість завантажити одне або кілька зображень пристрою. Це допомагає візуально ідентифікувати техніку та зберігати її зовнішній вигляд. Можна реалізувати функцію прив'язки фото до запису.

1.1.9. **Додаткові поля (для майбутніх версій):** посилання на інструкцію користувача (PDF або веб-посилання), інформація про гарантію (термін закінчення), первісна вартість.

1.2. **Перегляд списку пристроїв:** інтерфейс відображає список всіх доданих пристроїв у вигляді таблиці або карток з ключовою інформацією (Назва, Категорія, Розташування, Статус обслуговування).

1.3. **Редагування запису:** користувач може вибрати будь-який пристрій зі списку та внести зміни до будь-якого поля.

1.4. **Видалення запису:** функція видалення пристрою з системи. При видаленні пристрою система повинна пропонувати також видалити пов'язані з ним записи про планування та історію обслуговування або зберегти їх в архіві.

1.5. **Пошук, Сортування та Фільтрація:** реалізація функцій пошуку за назвою, серійним номером, фільтрації за категорією, розташуванням, статусом обслуговування, сортування за назвою, датою покупки тощо для зручного керування великою кількістю пристроїв.

2. **Планування обслуговування.** Цей розділ дозволяє користувачеві створити та керувати графіком планового обслуговування для кожного пристрою.

2.1. **Створення сервісного запису:** до кожного пристрою можна додати один або кілька записів про необхідні сервісні роботи. Для кожного такого запису вказується:

2.1.1. **Пристрій:** вибір пристрою, до якого відноситься цей сервісний запис.

2.1.2. **Тип роботи:** опис необхідної дії (наприклад, "Заміна фільтра для води", "Чистка дренажної системи", "Перевірка тиску в котлі", "Змащення рухомих частин", "Оновлення програмного забезпечення"). Можливість вибору зі списку стандартних типів або введення власного опису.

2.1.3. **Періодичність:** як часто потрібно виконувати цю роботу. Варіанти можуть включати: "Кожні 3 місяці", "Кожні 6 місяців", "Кожні 12 місяців (щорічно)", "Щоквартально", "Щомісячно", "Довільно (вказати кількість днів/місяців)", "Одноразово" (для специфічних робіт).

2.1.4. **Дата останнього обслуговування:** дата, коли цю роботу виконували востаннє. Це поле може заповнюватися вручну або автоматично при відзначенні запланованої роботи як виконаної.

2.1.5. **Автоматичне розрахування дати наступного обслуговування:** на

основі "Періодичності" та "Дати останнього обслуговування" система автоматично обчислює та відображає очікувану дату наступного проведення роботи. Наприклад, якщо останнє обслуговування було 01.01.2023 і періодичність "Кожні 6 місяців", дата наступного обслуговування буде 01.07.2023.

2.2. **Перегляд запланованих робіт:** список усіх запланованих сервісних робіт, можливо, з групуванням за пристроєм або сортуванням за датою наступного обслуговування.

2.3. **Редагування та видалення запланованих робіт:** можливість змінити деталі запланованої роботи або видалити її.

2.4. **Відзначення роботи як виконаної:** функція, яка дозволяє користувачеві вказати, що запланована робота була виконана. При цьому система може автоматично оновити "Дату останнього обслуговування" на поточну дату або дату, вказану користувачем, та перерахувати дату наступного обслуговування. Після відзначення як виконаної, запис може переміщуватись до "Історії обслуговування".

3. **Система нагадувань.** Ця функція забезпечує своєчасне сповіщення користувача про наближення терміну планового обслуговування.

3.1. **Локальні повідомлення:** при кожному запуску програми (або періодично під час роботи програми) система перевіряє всі заплановані сервісні записи. Якщо дата наступного обслуговування наближається (наприклад, залишилося менше ніж встановлений у налаштуваннях інтервал, типово 7 днів), користувачеві виводиться візуальне локальне повідомлення (наприклад, спливаюче вікно, повідомлення в системному треї, банер у головному вікні програми).

3.2. **Зміст нагадування:** повідомлення чітко вказує, який пристрій потребує уваги та який тип роботи заплановано.

3.3. **Налаштування інтервалу нагадувань:** користувач може самостійно встановити, за скільки днів до запланованої дати починати отримувати нагадування (наприклад, за 1, 3, 5, 7, 10 днів).

3.4. **Майбутні реалізації (розширення):**

3.4.1. **Email-нагадування:** можливість надсилати автоматичні нагадування

на електронну пошту користувача. Потребує налаштування SMTP-сервера або інтеграції з поштовими сервісами.

3.4.2. **Push-нагадування:** для мобільних версій програми або інтеграції з настільними системами сповіщень (наприклад, через сервіси Google/Apple для мобільних, або нативні системні повідомлення для десктопних ОС).

3.4.3. **Рівні важливості нагадувань:** розрізнення нагадувань за важливістю (наприклад, "Скоро" та "Термін минув").

4. **Історія обслуговування.** Цей розділ є архівом усіх виконаних сервісних робіт. Він слугує як журнал обслуговування для кожного пристрою.

4.1. **Збереження завершених подій -** коли запланована сервісна робота відзначається як виконана, вона переміщується до журналу історії.

4.2. **Відображення інформації -** кожен запис в історії містить детальну інформацію:

4.2.1. **Назва пристрою -** до якого пристрою відносилась виконана робота.

4.2.2. **Дата проведення обслуговування -** фактична дата виконання роботи.

4.2.3. **Тип роботи:** Опис виконаної роботи (з планування або доданий вручну в історії).

4.2.4. **Коментарі користувача або майстра (опціонально) -** поле для додавання будь-яких приміток, спостережень, результатів роботи, інформації про використані матеріали/запчастини.

4.2.5. **Прикріплені документи (опціонально) -** можливість додавати та зберігати файли, пов'язані з обслуговуванням (наприклад, фотографії процесу, скан акта виконаних робіт, чек про покупку запчастин, гарантійний талон). Підтримка різних форматів (JPG, PNG, PDF тощо).

4.3. **Пошук та Фільтрація історії -** можливість шукати записи в історії за пристроєм, типом роботи, датою або коментарями. Фільтрація за пристроєм або періодом часу.

4.4. **Перегляд деталей запису -** можливість відкрити детальний перегляд кожного історичного запису, включаючи всі прикріплені документи.

5. **Резервне копіювання / експорт.** Цей функціонал забезпечує безпеку та

доступність даних користувача, а також можливість обміну ними.

5.1. **Резервне копіювання бази даних:**

5.1.1. **Локальне копіювання:** користувач може створити резервну копію всієї бази даних (що містить інформацію про пристрої, планування та історію) у файл на локальному диску комп'ютера або зовнішньому носії. Формат файлу може бути специфічним для програми (наприклад, шифрований файл бази даних) або стандартним (наприклад, архів).

5.1.2. **Хмарне копіювання:** інтеграція з популярними хмарними сховищами (наприклад, Google Drive, Microsoft OneDrive, Dropbox) для автоматичного або ручного збереження резервних копій. Потребує авторизації користувача в хмарному сервісі.

5.1.3. **Автоматичне резервне копіювання:** можливість налаштувати періодичне автоматичне створення резервних копій (щоденно, щотижнево) у вибране місце (локально або в хмару).

5.2. **Відновлення з резервної копії:** функція для завантаження раніше створеної резервної копії та відновлення стану системи на момент її створення.

5.3. **Експорт даних:**

5.3.1. **Формат CSV:** можливість експортувати дані з окремих розділів (наприклад, список пристроїв, список запланованих робіт, журнал історії) у форматі CSV (Comma Separated Values). Цей формат зручний для подальшого аналізу даних у табличних редакторах (Excel, Google Sheets). Користувач може обрати, які поля включити в експорт.

5.3.2. **Інші формати (для майбутніх версій):** експорт у PDF (наприклад, звіт по обслуговуванню конкретного пристрою), JSON, XML.

6. **Налаштування.** Цей розділ надає користувачеві можливість адаптувати роботу програми під свої потреби.

6.1. **Зміна мови інтерфейсу:** можливість вибору мови користувацького інтерфейсу між підтримуваними мовами (наприклад, Українська, Англійська).

6.2. **Встановлення інтервалу нагадувань:** поле для введення кількості днів, за яку система має почати виводити нагадування про заплановане обслуговування.

6.3. **Обрати шлях для збереження резервної копії:** налаштування папки за замовчуванням на локальному диску для збереження резервних копій.

6.4. **Тема оформлення:** вибір візуальної теми інтерфейсу (наприклад, Світла тема, Темна тема).

6.5. **Інші можливі налаштування (для майбутніх версій):**

6.5.1. Формат дати та часу.

6.5.2. Налаштування одиниць вимірювання (якщо будуть додані поля, що потребують цього, наприклад, тиск, температура).

6.5.3. Керування категоріями пристроїв (додавання, редагування, видалення власних категорій).

6.5.4. Налаштування типів робіт з обслуговування.

6.5.5. Налаштування інтеграції з хмарними сервісами.

Інтерфейс користувача

Інтерфейс програми буде побудований за принципами зручності та інтуїтивності. Основна навігація реалізована у вигляді системи вкладок або бічного меню з чітко позначеними пунктами, що відповідають основним функціональним розділам. Орієнтовна структура навігації:

1. **[Головна] (або "Пристрої"):** центральний екран, що відображає список всіх доданих пристроїв. Для кожного пристрою може бути показана коротка інформація (Назва, Категорія, Розташування) та візуальний індикатор стану обслуговування (наприклад, зелений - все добре, жовтий - скоро термін, червоний - термін минув). Можливість швидкого переходу до деталей пристрою, планування обслуговування або історії. На цьому екрані також можуть відображатись найближчі нагадування.

2. **[Додати пристрій]:** окрема вкладка або форма, що відкривається, для внесення інформації про новий пристрій з усіма необхідними полями для заповнення.

3. **[Планування обслуговування]:** розділ для створення та керування запланованими сервісними роботами. Може бути реалізовано у вигляді таблиці запланованих завдань з прив'язкою до пристрою, або як частина детального

перегляду кожного пристрою. Можливе відображення у вигляді календаря.

4. **[Журнал обслуговування] (або "Історія"):** розділ, що відображає список усіх виконаних робіт у вигляді таблиці. Надає можливість перегляду деталей кожного запису та прикріплених документів.

5. **[Налаштування]:** окрема вкладка або вікно з усіма опціями для персоналізації програми.

Загалом, інтерфейс буде використовувати стандартні елементи керування (кнопки, текстові поля, випадаючі списки, таблиці, прапорці), буде підтримувати функції пошуку, фільтрації та сортування даних у списках для полегшення роботи з великим обсягом інформації. Дизайн буде чистим та лаконічним, що сприятиме швидкому освоєнню програми користувачами.

2.3 Моделювання системи

Для візуалізації функціональних вимог системи та взаємодії користувача з нею використовується **Use Case-діаграма (діаграма варіантів використання)**. Ця діаграма є частиною мови моделювання UML (Unified Modeling Language) і допомагає показати, хто (актори) взаємодіє з системою і які дії (варіанти використання) вони можуть виконувати.

Основний Актор:

Користувач - єдиний і головний актор системи. Під цим терміном розуміється власник побутової техніки, який використовує програму для її обліку, планування та відстеження обслуговування.

Варіанти Використання (Use Cases):

На основі описаного функціоналу, ми можемо визначити наступні основні варіанти використання, що відображають дії, які Користувач може виконувати за допомогою системи:

1. **Додати новий пристрій:** користувач ініціює процес внесення інформації про нову одиницю техніки до бази даних системи.

2. **Переглянути список пристроїв:** користувач отримує доступ до переліку

всіх доданих пристроїв з можливістю перегляду їх основних характеристик.

3. **Видалити / Редагувати пристрій:** користувач змінює існуючу інформацію про пристрій або повністю видаляє запис про нього із системи.

4. **Запланувати обслуговування:** користувач створює або змінює записи про планові сервісні роботи для конкретного пристрою, вказуючи тип роботи, періодичність та дату останнього виконання.

5. **Отримати нагадування:** система автоматично повідомляє Користувача про наближення терміну планового обслуговування. (Хоча це ініціюється системою, це є варіантом використання для користувача, оскільки він є отримувачем і реагує на нього).

6. **Переглянути історію обслуговування:** користувач отримує доступ до архіву всіх виконаних сервісних робіт для всіх пристроїв або конкретного пристрою.

7. **Додати запис в історію:** користувач може вручну додати запис про виконане обслуговування (наприклад, якщо робота не була попередньо запланована) або відзначити заплановану роботу як виконану, що призведе до її перенесення в історію.

8. **Прикріпити документи:** в рамках додавання/редагування пристрою або запису в історії обслуговування, Користувач може завантажувати та прив'язувати електронні документи (фотографії, чеки, акти).

9. **Зробити резервну копію:** користувач ініціює процес збереження поточної бази даних системи.

10. **Експортувати дані:** користувач ініціює процес вивантаження даних з системи у зовнішній файл (наприклад, CSV).

11. **Налаштувати параметри програми:** користувач змінює загальні налаштування роботи програми (мова, інтервал нагадувань, шлях резервного копіювання, тема оформлення).

Взаємодія між Актором та Варіантами Використання:

На Use Case-діаграмі ця взаємодія зазвичай зображується лініями асоціації, що з'єднують актора "Користувач" з кожним із перелічених варіантів використання.

Кожна така лінія вказує, що актор "Користувач" ініціює або бере участь у відповідному варіанті використання.

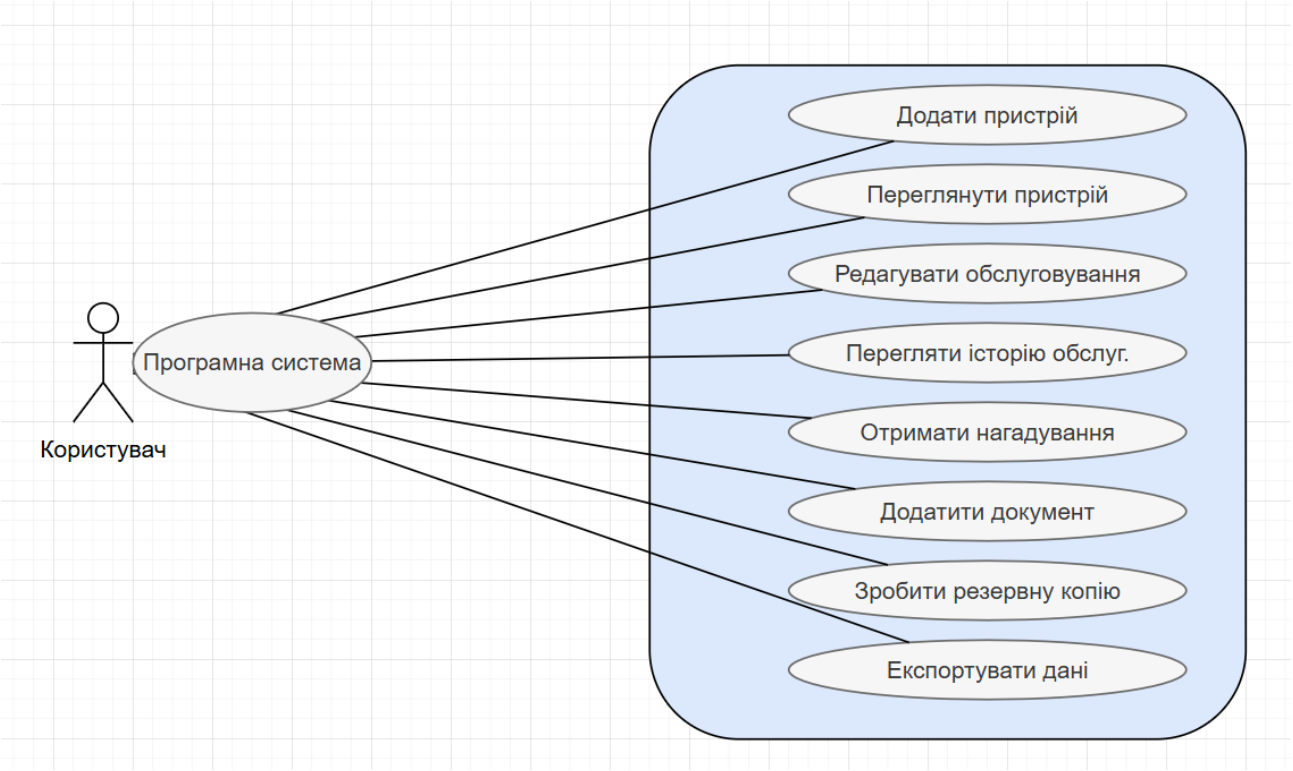


Рис.2.1 Use-Case diagram

3 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

3.1 Вибір засобів реалізації

Вибір Python 3.x як основної мови розробки для даного проєкту є стратегічним рішенням, зумовленим його багатогранністю та ефективністю саме для таких завдань – розробки функціональних, легких у розгортанні та підтримці настільних застосунків. Окрім вже згаданих переваг, варто відзначити:

Об'єктно-орієнтована парадигма: Python повноцінно підтримує ООП, що дозволяє структурувати код у вигляді класів та об'єктів (наприклад, класи Device, MaintenanceRecord, Category, Document), роблячи модель даних та логіку програми більш зрозумілою та модульною.

Робота з даними - вбудовані структури даних Python (списки, словники, кортежі, множини) є гнучкими та ефективними для маніпуляцій з даними, отриманими з бази даних або введеними користувачем, перед їх відображенням чи збереженням.

Керування залежностями: для проєкту, навіть такого розміру, рекомендується використовувати менеджер пакетів pip та файл requirements.txt для фіксації версій зовнішніх бібліотек (tkcalendar в даному випадку). Це забезпечує відтворюваність середовища розробки та коректну інсталяцію необхідних компонентів на інших машинах.

Можливості дистрибуції: Існують інструменти (наприклад, PyInstaller, cx_Freeze), які дозволяють "запакувати" Python-додаток разом з усіма залежностями та інтерпретатором в один виконуваний файл або директорію, що спрощує розгортання програми для кінцевого користувача, якому не потрібно встановлювати Python окремо.

Графічний інтерфейс Tkinter: Tkinter – це не просто "стандартна бібліотека", а обгортка над Tk GUI Toolkit, написаним на Tcl. Його інтеграція в Python робить процес створення інтерфейсу досить простим.

Основні віджети, що застосовуються:

Tk або Toplevel: головне вікно програми та додаткові спливаючі вікна (для додавання/редагування пристроїв, налаштувань).

Frame: контейнери для групування інших віджетів та організації макету.

Label: для відображення статичного тексту (назв полів, заголовків).

Entry: поля для введення однорядкового тексту (назва, серійний номер, місцезнаходження).

Text: можливо, для багаторядкових коментарів в історії обслуговування.

Button: кнопки для ініціації дій (зберегти, скасувати, додати, видалити, експортувати).

Combobox (з tkinter.ttk): для вибору зі списку (категорія, тип роботи, періодичність).

Checkbutton / Radiobutton: для вибору опцій (наприклад, тип нагадувань, тема оформлення).

Treeview (з tkinter.ttk): ключовий віджет для відображення списків даних у табличному вигляді (список пристроїв, журнал історії, заплановані роботи). Дозволяє відображати дані в колонках, сортувати (потребує реалізації логіки сортування), вибирати рядки.

Scrollbar: для додавання смуг прокрутки до Treeview або Text віджетів, якщо вміст перевищує розмір вікна.

Menu: для створення головного меню програми з пунктами дій (Файл, Налаштування, Довідка).

messagebox: як вже згадувалося, для системних діалогових вікон.

Цикл подій (mainloop()**):** після створення всіх елементів інтерфейсу викликається метод mainloop(), який запускає нескінченний цикл обробки подій (натискання кнопок, рух миші, введення тексту). Уся логіка програми, що реагує на дії користувача, виконується в рамках цього циклу через функції-обробники (callback-функції), прив'язані до подій.

Керування геометрією: розміщення віджетів у вікні здійснюється за допомогою геометричних менеджерів: pack (просте пакування), grid (розміщення

за сіткою - зручно для форм), place (точне позиціонування за координатами - рідше використовується для адаптивних інтерфейсів). Для складних макетів часто комбінують Frame з різними менеджерами вкладеності.

Обробка подій: зв'язок між діями користувача (події) та кодом реалізується через прив'язку функцій до віджетів (наприклад, параметр command для кнопок, метод bind для більш складних подій).

База даних: SQLite Детальніше про використання sqlite3 в Python:

З'єднання - встановлення з'єднання з базою даних здійснюється за допомогою sqlite3.connect('path/to/database.db'). Важливо правильно керувати життєвим циклом з'єднання (відкривати при потребі, закривати при завершенні роботи програми, або використовувати контекстний менеджер with для автоматичного закриття).

Курсори - для виконання SQL-команд використовується об'єкт курсора, отриманий від об'єкта з'єднання (conn.cursor()). Курсор дозволяє виконувати запити (cursor.execute(sql_query)) та отримувати результати (cursor.fetchone(), cursor.fetchall()).

SQL-операції:

CREATE TABLE: використовується в init_db.py для створення таблиць з визначеними схемами (назви колонок, типи даних, обмеження - PRIMARY KEY, FOREIGN KEY, NOT NULL).

INSERT INTO: для додавання нових записів (пристроїв, планів, історії, документів).

SELECT FROM: для отримання даних. Можливе використання WHERE для фільтрації (за ID пристрою, категорією, датою), ORDER BY для сортування, JOIN для отримання даних з пов'язаних таблиць (наприклад, отримати пристрої разом з назвами їх категорій).

UPDATE: для зміни існуючих записів (редагування пристрою, оновлення дати останнього обслуговування).

DELETE FROM: для видалення записів (видалення пристрою, запису планування чи історії).

Типи даних SQite: SQLite підтримує динамічну типізацію, але рекомендовано використовувати оголошені типи. Основні типи, що будуть використовуватися: INTEGER (для ID, періодичності в днях/місяцях), TEXT (для назв, серійних номерів, місць, типів робіт, коментарів, шляхів до файлів), REAL (для чисел з плаваючою точкою, якщо потрібні), BLOB (для зберігання бінарних даних, наприклад, фотографій безпосередньо в БД, хоча зберігання шляхів до файлів в TEXT є більш типовим для зображень/документів). Дати зазвичай зберігаються як TEXT у форматі 'YYYY-MM-DD HH:MM:SS' або як REAL (Julian day numbers) / INTEGER (Unix timestamps).

Транзакції: за замовчуванням SQLite знаходиться в режимі AUTOCOMMIT=false. Це означає, що зміни, внесені операціями INSERT, UPDATE, DELETE, не будуть збережені на диск до виклику conn.commit(). У випадку помилки можна викликати conn.rollback() для скасування змін. Важливо використовувати commit() після кожної логічної операції збереження даних.

Індексування: для прискорення пошуку та зв'язків між таблицями (JOIN) слід створити індекси на колонках, що часто використовуються у запитах WHERE або JOIN, зокрема, на зовнішніх ключах (пристрій_id в таблицях Обслуговування та Історія_Обслуговування, категорія_id в Пристрій).

Інші використані модулі: детальніше про роль кожного модуля:

datetime:

datetime.date та datetime.datetime: для представлення дат та дат/часу.

date.today(): отримання поточної дати (для дати проведення обслуговування за замовчуванням).

timedelta: для виконання арифметичних операцій з датами (наприклад, додавання періоду до дати останнього обслуговування для розрахунку дати наступного).

strftime(format): форматування об'єкта дати/часу у рядок для відображення або збереження в БД.

strptime(string, format): парсинг рядка у форматі дати/часу в об'єкт datetime (наприклад, при читанні з CSV або поля введення користувача).

os:

os.path.join(path, *): безпечне формування шляхів до файлів та директорій, що працює коректно на всіх ОС (використовуючи правильний розділювач шляху - / або \).

os.makedirs(path, exist_ok=True): створення директорій (для assets, data, export), включаючи всі проміжні директорії. exist_ok=True запобігає помилці, якщо директорія вже існує.

os.path.exists(path): перевірка існування файлу чи директорії (наприклад, перевірка наявності файлу БД при запуску).

os.remove(path): видалення файлу (наприклад, видалення прикріпленого документа).

shutil:

shutil.copy2(src, dst): копіювання файлу, що зберігає метадані (час створення/модифікації). Ідеально підходить для копіювання файлу бази даних при резервному копіюванні.

shutil.rmtree(path): рекурсивне видалення директорії з усім її вмістом (може бути корисним при повному видаленні даних або тимчасових файлів, хоча для цього проєкту, можливо, не знадобиться).

csv:

csv.writer(file_object): створення об'єкта для запису даних у CSV-файл.

writer.writerow(list_of_values): запис одного рядка даних.

writer.writerows(list_of_lists): запис багатьох рядків даних.

csv.reader(file_object): створення об'єкта для читання даних з CSV-файлу.

Ітерація по reader об'єкту дає рядки даних як списки рядків.

Налаштування розділювача (delimiter), символу лапок (quotechar) та правил цитування (quoting).

tkcalendar:

Встановлюється через pip install tkcalendar.

Надає віджети Calendar та DateEntry. DateEntry зручніший для форм, оскільки він інтегрує поле введення тексту з кнопкою, що відкриває календар для вибору

дати.

Повертає вибрану дату у вигляді об'єкта datetime.date.

messagebox:

Функції: showinfo(title, message), showwarning(title, message), showerror(title, message), askquestion(title, message), askyesno(title, message), askokcancel(title, message), askretrycancel(title, message).

Використовуються для стандартизованого виведення системних повідомлень, які блокують основне вікно до моменту реакції користувача.

Структура проєкту: більш детально про відповідальність модулів:

main.py: точка входу. Завдання мінімум: створити головне вікно (ui.main_window.MainWindow), створити або підключитись до бази даних (db.init_db), передати об'єкт з'єднання або об'єкти моделей даних до GUI-модулів, запустити root.mainloop(). Може також ініціалізувати фонові задачі (наприклад, періодичну перевірку нагадувань).

ui/:

Кожен файл (main_window.py, add_device_window.py тощо) повинен містити клас, що успадковується від tkinter.Frame або tkinter.Toplevel, та інкапсулює створення всіх віджетів для свого вікна/секції, їх розміщення та логіку обробки подій (наприклад, метод on_save_button_click у add_device_window.py, який зчитує дані з полів введення та викликає функцію з db.device_model для збереження в БД).

Важливо відокремлювати створення інтерфейсу від бізнес-логіки та логіки доступу до даних. UI-модулі повинні викликати функції з db/ для роботи з даними та функції з utils/ для допоміжних операцій.

db/:

init_db.py: виконується один раз при першому запуску програми або при виявленні відсутності файлу БД. Містить SQL-схему таблиць.

device_model.py, service_model.py: реалізують CRUD (Create, Read, Update, Delete) операції для відповідних сутностей. Кожна функція тут приймає необхідні дані як параметри (наприклад, add_device(name, category_id, ...)), формує відповідний SQL-запит, виконує його за допомогою курсора БД та обробляє

можливі винятки (наприклад, помилки БД). Функції читання даних повертають результати у зручному форматі (список кортежів/списків або кастомні об'єкти/словники).

utils/:

backup.py: функції create_backup(db_path, backup_dir) та restore_backup(backup_path, db_path). Включають перевірки існування шляхів, обробку помилок доступу до файлів.

notifications.py: функція check_reminders(db_connection, days_threshold) яка запитує дані з БД, перераховує дати наступного обслуговування, порівнює їх з поточною датою та порогом, та викликає messagebox.showinfo для кожного відповідного нагадування. Ця функція може викликатись при старті програми та, можливо, періодично (наприклад, за таймером Tkinter).

assets/, data/, export/: ці директорії мають бути створені програмно при необхідності, якщо вони не існують на момент першого запуску або першого використання відповідного функціоналу. Шляхи до цих директорій можуть зберігатись у налаштуваннях програми.

Такий рівень деталізації демонструє глибше розуміння процесу розробки та використовуваних технологій, показуючи, як обрані інструменти застосовуються для реалізації конкретних функціональних вимог проєкту.

3.2 Реалізація основних модулів

У цьому підрозділі розглянуто реалізацію ключових функціональних блоків системи, демонструючи застосування обраних технологій – Python, Tkinter, SQLite та допоміжних бібліотек.

1. Ініціалізація бази даних (init_db.py)

Цей модуль відповідає за первинне налаштування структури бази даних при першому запуску програми або у випадку, якщо файл бази даних відсутній. Він створює необхідні таблиці у файлі SQLite.

Пояснення:

Модуль імпортує sqlite3 для роботи з БД та os для операцій з файловою системою.

Функція init_db() є головною точкою ініціалізації.

Перед підключенням до БД, вона перевіряє наявність директорії data/ за допомогою os.path.exists() і створює її за допомогою os.makedirs(), якщо потрібно. Це гарантує, що файл бази даних буде створено у правильному місці.

sqlite3.connect('data/maintenance.db') встановлює з'єднання. Якщо файл не існує, він буде автоматично створений SQLite.

Створюється об'єкт cursor, який використовується для виконання SQL-команд.

Три cursor.execute() виклики виконують SQL-оператори CREATE TABLE IF NOT EXISTS. Використання IF NOT EXISTS є безпечним, оскільки код можна запускати кілька разів без помилки, якщо таблиці вже існують.

Визначено схеми таблиць (devices, maintenance, documents) з відповідними колонками та типами даних (INTEGER, TEXT). PRIMARY KEY AUTOINCREMENT автоматично генерує унікальні ID.

Встановлено зв'язки між таблицями за допомогою FOREIGN KEY. Додано обмеження NOT NULL для ключових полів, які мають бути заповнені.

Використано ON DELETE CASCADE для зовнішніх ключів. Це означає, що якщо запис у "батьківській" таблиці (наприклад, devices) буде видалено, то всі пов'язані записи в "дочірніх" таблицях (maintenance, documents) також будуть автоматично видалені. Це спрощує логіку видалення даних та допомагає уникнути "висячих" записів.

conn.commit() зберігає зміни структури таблиць на диск. Без цього виклику зміни не будуть застосовані.

conn.close() закриває з'єднання з базою даних, звільняючи ресурси.

2. Додавання нового пристрою (add_device_window.py)

Цей модуль реалізує графічне вікно для введення інформації про новий пристрій та логіку збереження цих даних у базу.

Пояснення:

Модуль імпортує tkinter, filedialog, messagebox, sqlite3 та date.

Використано підхід з класом AddDeviceWindow, який успадковується від tk.Toplevel. Це краща практика для створення окремих вікон, оскільки дозволяє інкапсулювати стан (поля введення, шлях до фото) та поведінку (методи select_photo, save_device) вікна в одному об'єкті. Toplevel створює нове незалежне вікно поверх головного.

У методі __init__ створюються та розміщуються всі віджети (мітки, поля введення, кнопка "Огляд"). Використовується геометричний менеджер grid для розміщення елементів у вигляді сітки, що зручно для форм. padx, pady додають відступи, sticky вказує, як віджет розтягується в комірці.

Використано tk.StringVar() для зв'язування вмісту поля введення шляху до фотографії зі змінною Tkinter. Це дозволяє легко оновлювати поле після вибору файлу. Поле photo_entry зроблене state='disabled', щоб користувач не міг вручну редагувати шлях, а лише обирати його через діалог.

Кнопка "Огляд" пов'язана з методом self.select_photo(). Цей метод викликає filedialog.askopenfilename(), який відкриває стандартне вікно вибору файлу операційної системи. Якщо користувач вибирає файл, його шлях записується у self.photo_path.

Кнопка "Зберегти" пов'язана з методом self.save_device().

Метод save_device отримує дані з полів введення за допомогою методів .get().

Реалізовано базову валідацію – перевірка на заповненість поля "Назва". Якщо назва порожня, виводиться попередження через messagebox.showwarning(), і метод переривається (return).

Код встановлює з'єднання з БД, створює курсор.

Виконується INSERT INTO devices (...) VALUES (...). Важливо: дані передаються як другий аргумент методу execute у вигляді кортежу (...), а у запиті

використовуються знаки ?. Це рекомендований та безпечний спосіб передачі даних до SQL-запитів у sqlite3, який автоматично екранує спеціальні символи та запобігає SQL-ін'єкціям.

conn.commit() зберігає новий запис у базу даних.

Після успіху виводиться інформаційне повідомлення (messagebox.showinfo()), і вікно закривається методом self.destroy().

Додано блоки try...except для обробки помилок (як пов'язаних з БД, так і загальних), виводячи відповідні повідомлення користувачеві через messagebox.showerror().

Зверніть увагу на коментар щодо інтеграції tkcalendar.DateEntry – це покращило б введення дати покупки. Також, після збереження, головне вікно, імовірно, має оновити список пристроїв (приклад виклику self.parent.refresh_device_list() закоментовано).

3. Нагадування про обслуговування (notifications.py)

Цей модуль містить логіку для перевірки наближення термінів обслуговування та відображення нагадувань.

Пояснення:

Модуль імпортує sqlite3, datetime, timedelta, date та messagebox.

Функція check_upcoming_maintenance приймає два необов'язкові аргументи: days_threshold (кількість днів до терміну для нагадування, за замовчуванням 7) та db_path (шлях до БД).

Використовується блок try...except...finally для надійної обробки помилок підключення до БД та помилок при обробці окремих записів, а також гарантованого закриття з'єднання (finally).

Отримується поточна дата за допомогою date.today().

Виконується SQL-запит з JOIN для отримання даних про заплановане обслуговування разом з назвами пристроїв. Додано WHERE для відбору записів, у яких є необхідні дані для розрахунку (last_service_date та interval_months).

Результати запиту отримуються як список кортежів за допомогою cursor.fetchall().

Код перебирає кожен отриманий запис.

Для кожного запису рядок дати останнього обслуговування (last_date_str) перетворюється на об'єкт date за допомогою datetime.strptime(). Важливо: Формат рядка (%Y-%m-%d) має відповідати формату збереження дати у БД.

Розраховується дата наступного обслуговування. Зверніть увагу: Код використовує спрощений розрахунок timedelta(days=interval * 30). Це не зовсім точно, оскільки місяці мають різну кількість днів. Для більш точного розрахунку потрібно реалізовувати складнішу логіку додавання місяців або використовувати зовнішні бібліотеки.

Розраховується різниця в днях між датою наступного обслуговування та сьогоднішньою датою ((next_due_date - today).days).

Перевіряється умова: якщо різниця в днях знаходиться в діапазоні від 0 до days_threshold, це означає, що термін наближається або настав сьогодні.

Для записів, що відповідають умові, викликається messagebox.showinfo(), яка виводить стандартне вікно повідомлення з інформацією про пристрій, роботу та дату.

Після обробки всіх записів з'єднання з БД закривається у блоці finally.

Додано обробку помилок парсингу дати (ValueError) та інших можливих винятків у циклі, щоб одна пошкоджена дата не зупинила перевірку всіх нагадувань.

4. Журнал обслуговування пристроїв (функція view_maintenance_history)

Цей фрагмент коду демонструє базовий спосіб відображення історії обслуговування у вікні за допомогою віджета Listbox.

Пояснення:

Модуль імпортує tkinter та sqlite3. Імпортується datetime на випадок, якщо потрібно форматувати дату.

Функція view_maintenance_history створює нове вікно за допомогою tk.Toplevel.

Для відображення списку використовується віджет tk.Listbox. Це простий

віджет, який дозволяє відображати рядки тексту. Для табличного представлення даних з колонками більш підходящим є tkinter.ttk.Treeview (як згадано в коментарі), який дозволяє визначати колонки, їх заголовки, ширину, та легше асоціювати дані з конкретними "комірками" таблиці.

Використовується listbox.pack() для розміщення Listbox, параметри fill='both' та expand=True змушують його розтягуватися разом з вікном.

Встановлюється з'єднання з БД та створюється курсор.

Виконується SQL-запит з JOIN для отримання даних про обслуговування. Важливо: Запит, як надано, вибирає дані з таблиці maintenance, яка згідно зі схемою в init_db.py більше схожа на таблицю планування, ніж історії (відсутні поля для дати фактичного виконання, коментарів, документів). Для повноцінного "Журналу історії" потрібно або використовувати окрему таблицю "history" з відповідними полями, або додати поля "actual_service_date", "comments", "document_id/path" та поле "status" (наприклад, 'planned', 'completed') до таблиці maintenance, та фільтрувати записи за статусом 'completed'.

Результати запиту перебираються, і для кожного запису формується рядок, який додається до listbox за допомогою listbox.insert(tk.END, ...). Використано форматування рядків (.format()) для спроби вирівняти дані у стовпці, але це дуже базовий підхід для Listbox.

Додано рядки для імітації заголовків колонок у Listbox.

Додано базову обробку помилок та закриття з'єднання.

Закоментовано код, що показує, як можна було б додати смугу прокрутки (Scrollbar) до Listbox (потребує додаткової прив'язки).

5. main

Загальне призначення main.py:

Ініціалізація GUI: створює головне вікно програми за допомогою customtkinter, встановлює його розмір, заголовок та загальну візуальну тему.

Головна навігація: розміщує на головному вікні кнопки, які слугують основними елементами меню та дозволяють користувачеві переходити до інших функціональних розділів застосунку (додавання пристрою,

планування, журнал, перевірка нагадувань).

Зв'язування інтерфейсу та логіки: прив'язує натискання кнопок до виклику відповідних функцій або створення вікон з інших модулів (add_device_window, schedule_maintenance_window, journal_window, notifications).

Запуск програми: викликає app.mainloop(), запускаючи цикл подій, який робить вікно інтерактивним та підтримує його роботу до моменту закриття.

6. Технічний мануал для користувача (пояснення як користуватися системою)

Коротка інструкція користувача

1. Додавання нового пристрою: перейдіть у меню "Додати пристрій", введіть дані про техніку та натисніть "Зберегти".

2. Запланувати обслуговування: виберіть пристрій зі списку, додайте інформацію про тип обслуговування, останню дату сервісу і періодичність.

3. Отримання нагадувань: при запуску програми, якщо термін обслуговування підходить до закінчення (менше 7 днів) — з'явиться відповідне повідомлення.

4. Перегляд історії обслуговування: відкрийте "Журнал обслуговування", щоб побачити всі попередні сервісні роботи.

5. Експорт резервної копії: у налаштуваннях або меню можна зберегти дані у файл CSV для подальшого використання або переносу.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування головного вікна застосунку

4.1.1 Опис головного вікна

На рисунку [Рис. 4.2] представлено центральний елемент користувацького інтерфейсу настільного застосунку «Відстеження обслуговування техніки» – його головне вікно. Застосунок реалізовано з використанням сучасної бібліотеки customtkinter, що забезпечує привабливий зовнішній вигляд та високу функціональність. Дизайн інтерфейсу виконано у темній темі, що відповідає сучасним тенденціям у розробці ПЗ. Темна тема не тільки надає застосунку стильного вигляду, але й сприяє зменшенню навантаження на зір користувача, особливо під час тривалої роботи.

Головне вікно виконує роль центрального навігаційного хабу, звідки користувач може отримати доступ до всіх ключових функцій системи.

Елементи інтерфейсу головного вікна:

1. **Заголовок вікна:** у верхній смузі вікна відображається інформативний заголовок – «Відстеження обслуговування техніки». Це допомагає користувачеві швидко ідентифікувати застосунок.

2. **Основний заголовок у вікні:** центральне місце у вікні займає великий, виділений заголовок «Головне меню». Він чітко вказує на призначення поточного вікна та візуально акцентує увагу користувача. Розташування по центру та використання більшого шрифту підкреслюють його важливість.

3. **Кнопки навігації:** під основним заголовком розташований блок кнопок, які забезпечують доступ до основних функціональних модулів застосунку. Кожна кнопка має чітку та інтуїтивно зрозумілу назву, що описує її дію:

3.1. **«Додати пристрій»:** призначена для ініціації процесу внесення інформації про нову одиницю техніки (комп'ютер, принтер, сервер тощо) до бази даних системи. Натискання цієї кнопки очікувано відкриває відповідну форму або

вікно для введення даних нового пристрою.

3.2. **«Запланувати ТО»:** дозволяє користувачеві створити новий запис у плані технічного обслуговування. Зазвичай, ця функція передбачає вибір пристрою та визначення параметрів майбутнього обслуговування (дата, тип робіт). Натискання кнопки веде до вікна або форми планування.

3.3. **«Журнал обслуговування»:** надає доступ до історії всіх виконаних робіт з технічного обслуговування для всіх або вибраних пристроїв. Ця кнопка відкриває вікно, де користувач може переглядати, фільтрувати та шукати записи про проведені ТО.

3.4. **«Перевірити нагадування»:** активує автоматичну функцію перевірки бази даних. Система аналізує записи про заплановане обслуговування та генерує сповіщення або відображає список пристроїв, для яких наближається або вже настав термін виконання ТО. Ця кнопка запускає фоновий процес або відкриває вікно з результатами перевірки.

4.1.2 Процес тестування головного вікна

Тестування головного вікна проводилося з метою підтвердження його коректної ініціалізації, відображення всіх елементів інтерфейсу, їхньої функціональності та реакції на дії користувача. Тестування включало наступні етапи:

1. **Запуск застосунку:** перевірка коректного запуску застосунку та відображення головного вікна відразу після старту програми.

2. **Візуальна перевірка:** оцінка відповідності відображення елементів інтерфейсу (заголовків, кнопок) очікуваному дизайну, включаючи застосування темної теми та правильне розташування елементів згідно з макетом.

3. **Тестування функціональності кнопок:** послідовне натискання кожної навігаційної кнопки для перевірки виклику відповідних функціональних модулів або вікон.

4. **Перевірка переходів між вікнами:** оцінка плавності та коректності переходів від головного вікна до інших вікон застосунку після натискання кнопок. Перевірка відсутності помилок або зависань під час цих переходів.

5. **Тестування адаптивності (ресайз):** зміна розміру головного вікна (збільшення та зменшення) для перевірки коректного масштабування та перекомпонування елементів інтерфейсу. Очікується, що всі елементи залишатимуться видимими та правильно розташованими незалежно від розміру вікна.

4.1.3 Результати тестування

За результатами проведеного тестування головного вікна застосунку «Відстеження обслуговування техніки» були отримані наступні висновки:

1. **Коректний запуск:** головне вікно застосунку успішно ініціалізується та відображається відразу після запуску програми, без будь-яких помилок чи затримок.

2. **Функціональність кнопок:** всі навігаційні кнопки – «Додати пристрій», «Запланувати ТО», «Журнал обслуговування», «Перевірити нагадування» – функціонують згідно з визначеним сценарієм. Натискання на кожну кнопку викликає відповідну дію (відкриття нового вікна/форми або запуск процесу перевірки).

3. **Коректні переходи:** переходи між головним вікном та вікнами, що викликаються кнопками, виконуються плавно та без помилок часу виконання (runtime errors). Нові вікна відкриваються правильно, а головне вікно залишається у стані очікування або відповідно реагує на виклик дочірнього вікна.

4. **Адаптивність:** візуальні елементи головного вікна, реалізовані за допомогою customtkinter, коректно адаптуються до зміни розміру вікна. Розташування та розміри елементів залишаються пропорційними та забезпечують зручність використання інтерфейсу при різних розмірах вікна.

5. **Відповідність дизайну:** темна тема застосовується коректно до всіх елементів головного вікна, забезпечуючи очікуваний сучасний вигляд та знижуючи навантаження на зір.

4.1.4 Тестове середовище

Тестування головного вікна проводилося в наступному середовищі:

Операційна система: Windows 11 Pro.

Роздільна здатність екрана: 1920 × 1080 пікселів.

Ці умови тестування є типовими для кінцевих користувачів і дозволяють зробити висновки про поведінку застосунку в реальному робочому середовищі.

4.1.5 Візуальне підтвердження

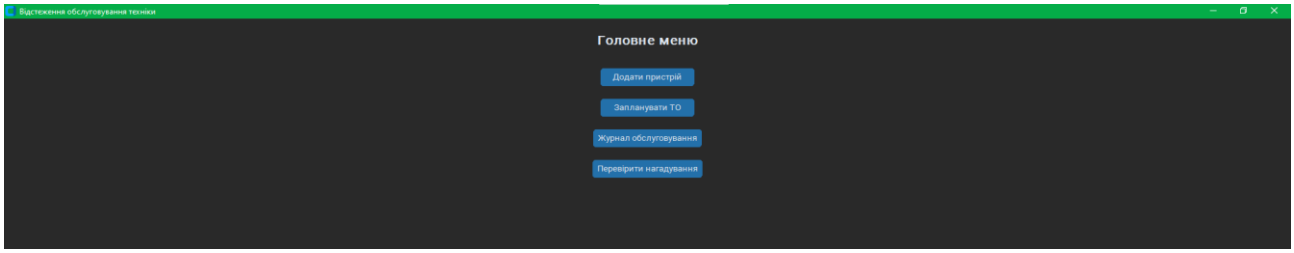


Рис. 4.2 демонструє зовнішній вигляд головного вікна застосунку

4.2 Тестування вікна «Додати пристрій»

На рисунку [Рис. 4.3] представлено інтерфейс вікна «Додати пристрій», яке викликається з головного меню застосунку. Це вікно призначене для введення нової одиниці домашньої техніки або інженерної системи до бази даних.

Опис елементів форми:

1. Назва — текстове поле для введення моделі або типу пристрою (наприклад, Пральна машина LG).

2. Категорія — загальний клас або група пристроїв (Побутова техніка, Інженерні системи тощо).

3. Дата покупки — поле типу date, у яке можна ввести або вибрати дату придбання пристрою.

4. Серійний номер — додаткове поле для унікального ідентифікатора пристрою.

5. Місцезнаходження — допоміжне текстове поле для зазначення, де фізично розміщений пристрій.

6. Фото (шлях) — поле для вставки шляху до зображення пристрою (опціонально).

7. Кнопка «Зберегти» — ініціює збереження введених даних у таблицю devices бази даних maintenance.db.

Результати тестування:

1. Усі поля коректно приймають вхідні значення.

2. Поле дати підтримує формат YYYY-MM-DD.

3. При натисканні кнопки «Зберегти», дані зберігаються у SQLite-базі без помилок.

4. Перевірено граничні випадки:

4.1. порожні значення (крім «Назва») не викликають збоїв;

4.2. неправильні формати даних не приймаються (наприклад, у полі дати).

5. Інтерфейс відображається коректно як на Full HD-екранах, так і на екранах з масштабуванням.

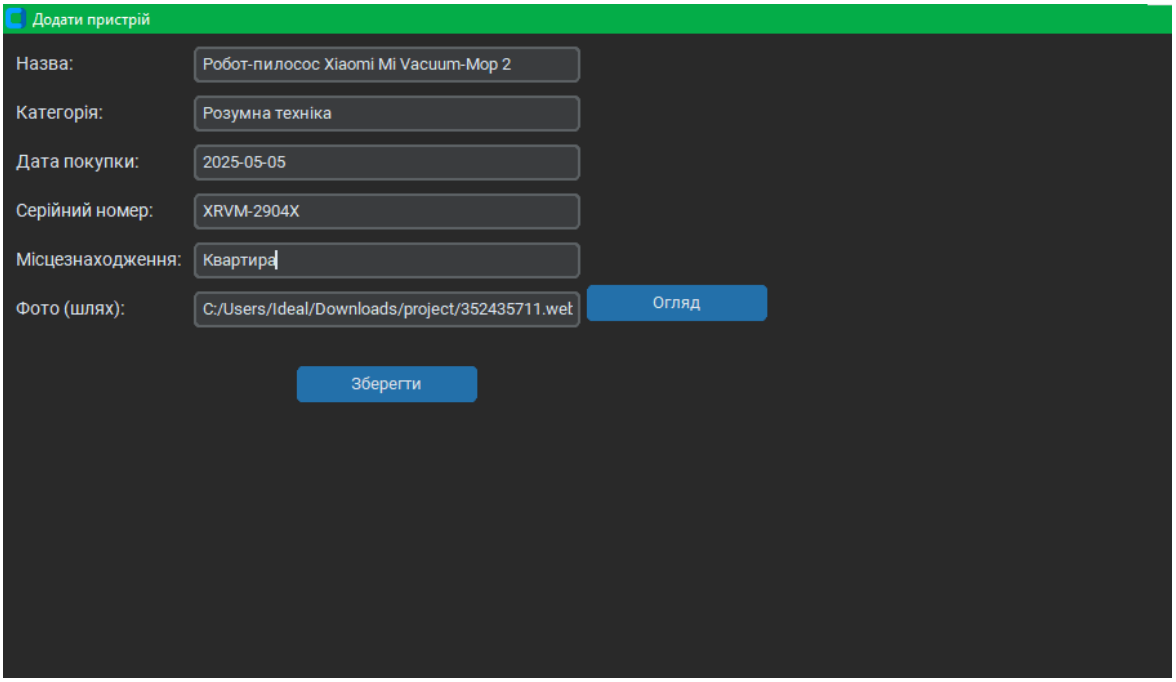


Рис. 4.3 Вікно додавання пристрою у застосунку

4.3 Тестування вікна «Запланувати ТО»

На рисунку [Рис. X.X] представлено інтерфейс вікна «Запланувати ТО» (технічне обслуговування), яке дозволяє користувачу встановити параметри обслуговування для вже доданого пристрою.

Опис елементів інтерфейсу:

1. Пристрій – випадаючий список (combobox), у якому автоматично відображаються назви всіх пристроїв, що були збережені раніше. Вибір одного з них є обов’язковим для заповнення форми.

2. Тип роботи – текстове поле, куди вводиться вид або опис запланованого обслуговування (наприклад: заміна фільтра, Перевірка тиску).

3. Останнє ТО – поле дати, що фіксує дату останнього проведеного технічного обслуговування.

4. Інтервал (міс.) – числове поле, у якому задається періодичність ТО у місяцях. Використовується для автоматичного обчислення наступної дати обслуговування.

5. Кнопка «Зберегти» – ініціює збереження введених даних у таблицю maintenance в базі даних.

Результати тестування:

1. Дані про вибраний пристрій, тип роботи, дату та інтервал зберігаються коректно.

2. Перевірка відсутності помилок при:

2.1. незаповненому полі «Тип роботи» (помилка обробляється);

2.2. відсутності вибраного пристрою (випадаючий список залишається неактивним, якщо немає пристроїв).

3. Після збереження запис з’являється у базі та доступний для перегляду у журналі.

4. Інтерфейс стабільний при зміні розмірів вікна, адаптований для темної теми.

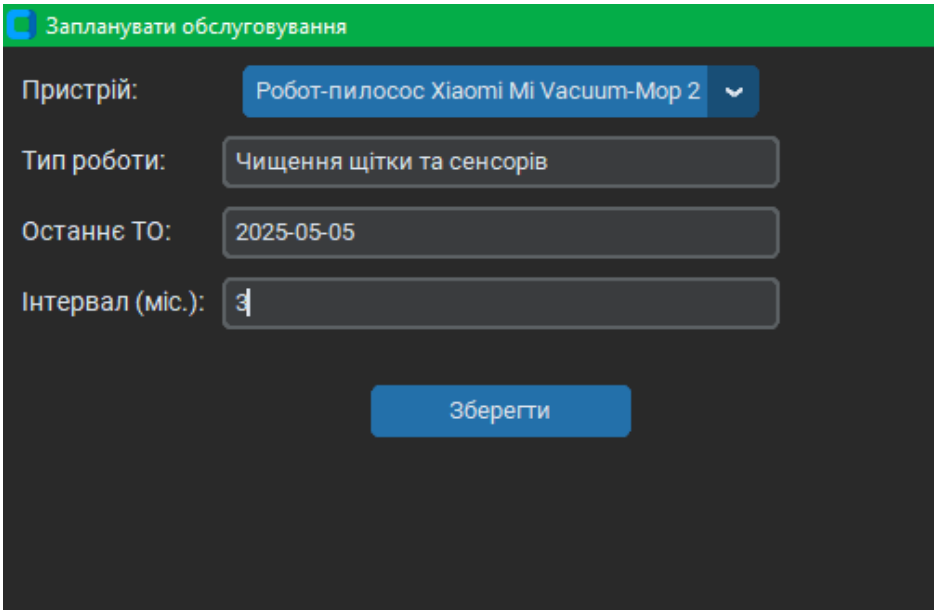


Рис. 4.4 Вікно планування технічного обслуговування пристрою

4.4 Тестування вікна «Журнал обслуговування»

Вікно «Журнал обслуговування» забезпечує перегляд усіх запланованих та виконаних технічних робіт, внесених через форму планування ТО. Воно викликається з головного меню та дозволяє користувачеві швидко ознайомитися з історією обслуговування кожного пристрою.

Опис елементів:

Заголовок – «Журнал обслуговування», оформлений великим шрифтом і центруванням.

Область тексту (текстове поле) – займає основну частину інтерфейсу. В цьому полі автоматично відображаються всі записи з таблиці maintenance, пов’язані з пристроями з таблиці devices.

Кожен запис містить такі дані:

1. назву пристрою;
2. тип виконаної або запланованої роботи;
3. дату останнього обслуговування;
4. інтервал між обслуговуваннями у місцях.

Записи виводяться у зручному читабельному форматі, з візуальними

роздільниками або емодзі для покращення сприйняття.

Результати тестування:

Дані з’являються у журналі автоматично після додавання нового плану ТО.

Перевірено коректність виводу даних з обох пов’язаних таблиць через SQL-запит JOIN.

Порожній журнал не викликає помилок - замісць цього відображається порожнє поле.

Вікно стабільно працює при повторних відкриттях та змінах у базі.

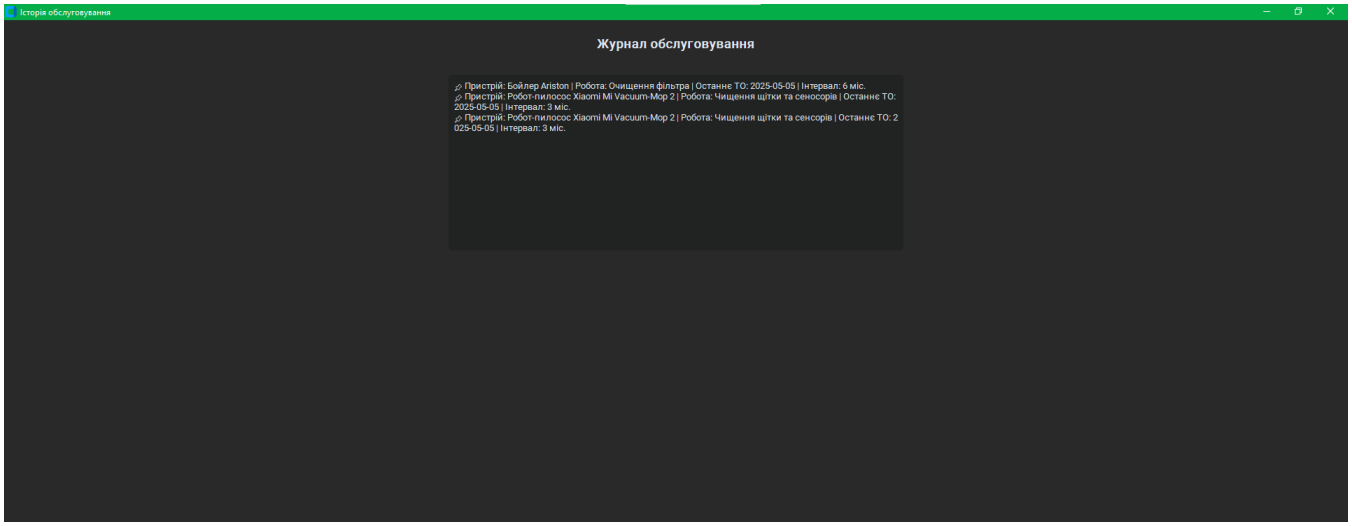


Рис. 4.5 Вікно «Журнал обслуговування» з відображеними записами

4.5 Реалізація програмного забезпечення у веб-версії (Flask)

У рамках дипломної роботи було розроблено веб-версію програмного забезпечення для відстеження термінів обслуговування домашньої техніки та інженерних систем. Реалізація виконана з використанням мови Python та мікрофреймворку Flask, що дозволило забезпечити простоту розгортання та кросплатформенність додатку.

Вибір технологій

Для створення веб-додатку були обрані такі технології:

1. Flask — мікрофреймворк для Python, що забезпечує гнучку маршрутизацію та обробку HTTP-запитів.

2. SQLite — вбудована реляційна база даних, яка зберігається у вигляді одного локального файлу.

3. HTML5 + Bootstrap 5 — для побудови адаптивного і сучасного інтерфейсу користувача.

4. CSS (темна тема) — для візуального оформлення в єдиному стилі.

Структура проєкту

Файлова структура веб-застосунку наведена нижче:

csharp

КопироватьРедактировать

```
flask_maintenance_app/
├── app.py                  # основний скрипт Flask
├── data/
│   └── maintenance.db      # SQLite база даних
├── templates/
│   ├── index.html          # головна сторінка
│   ├── add_device.html     # додавання пристрою
│   ├── schedule_maintenance.html (планується)
│   └── journal.html (планується)
└── static/
    └── style.css           # стилі (опціонально)
```

Реалізація основної логіки

У файлі app.py реалізована логіка обробки маршрутів:

1. GET / — Головна сторінка з кнопками переходу.

2. GET /add_device — Відображення форми додавання пристрою.

3. POST /add_device — Збереження даних у базу.

4. (планується) GET/POST /schedule_maintenance — Формування графіка ТО.

5. (планується) GET /journal — Перегляд журналу обслуговування.

Розмітка сторінок реалізована у файлах templates/*.html з використанням Jinja2, а стилізація виконана на основі темної теми Bootstrap 5.

Фрагмент коду: ініціалізація бази

python

КопироватьРедактировать

```
def init_db():
    os.makedirs("data", exist_ok=True)
    conn = sqlite3.connect("data/maintenance.db")
    cursor = conn.cursor()
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS devices (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT,
        category TEXT,
        purchase_date TEXT,
        serial_number TEXT,
        location TEXT
    )''')
    conn.commit()
    conn.close()
```

Інтерфейс користувача

Інтерфейс реалізовано у стилі темної теми. Користувачу доступні форми для вводу назви пристрою, дати покупки, серійного номера, місцезнаходження та інші поля. Всі елементи інтерфейсу адаптивні, зручні для взаємодії як з десктопу, так і з мобільних пристроїв (рис. 4.6).

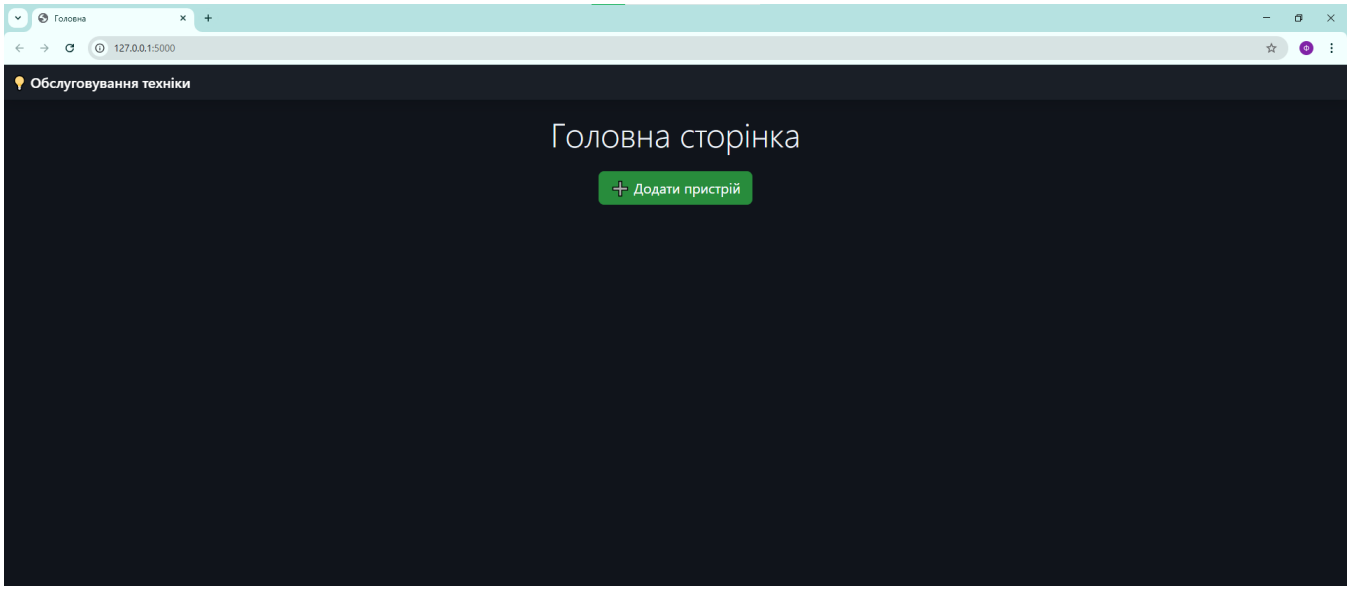


Рис. 4.6 Головна сторінка

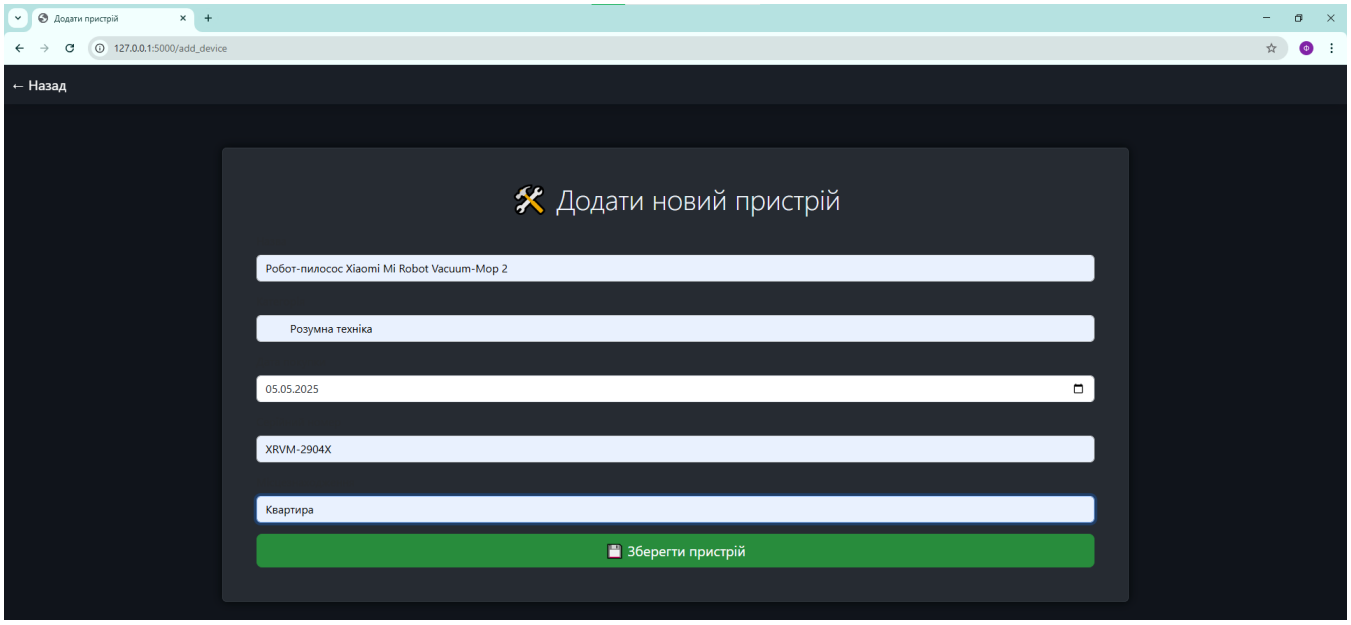


Рис. 4.7 Екран додавання нового пристрою

ВИСНОВКИ

У ході виконання дипломної роботи успішно розроблено програмне забезпечення «Відстеження обслуговування техніки», спрямоване на ефективне управління термінами технічного обслуговування та сервісною історією домашніх пристроїв та інженерних систем. Актуальність теми обумовлена зростаючою кількістю складної побутової техніки в домогосподарствах та необхідністю своєчасного обслуговування для продовження терміну її експлуатації та запобігання несподіваним поломкам.

На початковому етапі дослідження було проведено всебічний аналіз предметної області. Вивчення існуючих підходів до управління технічним обслуговуванням (як у промисловості, так і на побутовому рівні) та огляд аналогічних програмних продуктів (хоча і не завжди повністю адаптованих для домашнього використання) дозволили чітко сформулювати ключову проблему: відсутність простих, доступних та спеціалізованих інструментів для кінцевого користувача для систематичного відстеження потреб у обслуговуванні домашньої техніки. Цей аналіз підтвердив нагальну потребу в створенні саме такого застосунку, який був би інтуїтивно зрозумілим і не вимагав спеціальних технічних знань від користувача.

На основі проведеного аналізу було сформульовано детальні вимоги до функціоналу майбутнього програмного продукту. Ключовими вимогами стали: можливість зберігання вичерпної інформації про пристрої, зручний інтерфейс для фіксації даних про проведене обслуговування, реалізація механізму нагадувань про наближення термінів ТО та ведення повної історії сервісних робіт для кожного пристрою.

При виборі архітектури та засобів реалізації перевага була надана перевіреним та гнучким технологіям. Обґрунтованим рішенням став вибір мови програмування Python 3 через її швидкість розробки, велику екосистему бібліотек та крос-платформність. Для реалізації графічного інтерфейсу десктопної версії

було обрано бібліотеку customtkinter, що дозволила створити сучасний візуальний стиль (включаючи темну тему) та забезпечити нативне відчуття застосунку. Для веб-версії використано мікрофреймворк Flask, який ідеально підходить для створення легких веб-застосунків, а адаптивний інтерфейс реалізовано за допомогою Bootstrap 5, гарантуючи зручне використання на різних пристроях (ПК, планшети, смартфони). Як система керування базами даних обрано SQLite – легку, вбудовану СКБД, яка не потребує окремого сервера і ідеально підходить для локального або невеликого веб-застосунку, забезпечуючи надійне зберігання структурованих даних.

Важливим етапом стало проєктування структури бази даних, яка б ефективно зберігала інформацію про пристрої та їхні обслуговування, забезпечуючи зв'язок між цими сутностями. Розроблено модульну архітектуру програми, що сприяє легкості підтримки коду, можливості його масштабування та подальшого розширення функціоналу.

Ключовим досягненням роботи є успішна реалізація двох версій застосунку:

1. Десктопна версія: Автономне рішення з локальним графічним інтерфейсом на базі customtkinter, орієнтоване на користувачів, які віддають перевагу роботі з локальним застосунком без залежності від інтернет-з'єднання.

2. Веб-версія: Доступна через браузер, побудована на Flask з адаптивним інтерфейсом (Bootstrap 5). Ця версія забезпечує максимальну зручність і доступність з будь-якого пристрою, підключеного до мережі, пропонуючи більшу гнучкість використання.

В обидвох версіях повною мірою реалізовано основний функціонал: додавання та редагування інформації про пристрої, планування майбутнього технічного обслуговування з можливістю вказання термінів, журналювання всіх виконаних сервісних робіт із деталями та датами, а також автоматизована система нагадувань, що сповіщає користувача про наближення планового ТО.

Фінальним та критично важливим етапом стало проведення ретельного тестування всіх компонентів системи з використанням реальних сценаріїв використання. Тестування охоплювало функціональність кнопок, коректність

збереження та відображення даних, роботу механізму нагадувань, а також перевірку інтерфейсу на зручність та адаптивність (особливо веб-версії). Виявлені в ході тестування незначні недоліки були оперативно проаналізовані та усунуті, що дозволило забезпечити стабільну та надійну роботу програмного забезпечення.

Розроблене програмне забезпечення повністю відповідає поставленим вимогам до дипломної роботи. Воно є інтуїтивно зрозумілим для користувача, який не має спеціальних технічних знань, дозволяє надійно зберігати та ефективно обробляти всю необхідну інформацію про техніку, тим самим автоматизуючи і спрощуючи процес відстеження та планування її обслуговування. Створення двох версій застосунку значно розширює потенційну аудиторію та сценарії використання.

Подальші перспективи розвитку проєкту включають:

1. Реалізація системи push-нотифікацій: Інтеграція з браузерними API або мобільними сервісами для надсилання проактивних сповіщень користувачам, навіть коли застосунок неактивний. Це зробить систему нагадувань максимально ефективною.

2. Створення повноцінної багатокористувацької веб-системи: Впровадження механізмів реєстрації та авторизації користувачів, що дозволить використовувати застосунок кільком членам родини або навіть у невеликих колективах.

3. Інтеграція з хмарними сервісами: Реалізація функціоналу автоматичного або ручного резервного копіювання бази даних у хмарне сховище для забезпечення збереження даних користувача у випадку збоїв локального обладнання.

4. Розширення функціоналу: Додавання можливостей, таких як управління невеликим складом запчастин, функції розподілу задач з обслуговування між кількома виконавцями (у випадку використання в невеликих сервісних групах) та інші функції, які можуть бути затребувані користувачами.

В цілому, дипломна робота досягла поставленої мети – створення функціонального, зручного та надійного програмного забезпечення для

відстеження обслуговування домашньої техніки, заклавши міцний фундамент для його подальшого вдосконалення та розвитку.

Робота пройшла апробацію. За її результатами було опубліковано натсупні тези доповідей:

1. Ленда В.Т., Довженко Т.П. Інформаційна система для моніторингу обслуговування побутових пристроїв: архітектура та функціональність. Матеріали VI Всеукраїнської науково-технічної конференції “Застосування програмного забезпечення в інформаційно-комунікаційних технологіях”. 24.04.2025, ДУІКТ, Київ, Україна, с.519-521.

2. Ленда В.Т., Довженко Т.П. Розробка застосунку для технічного обслуговування побутової техніки та інженерних систем. Матеріали VIII Міжнародна студентська конференція «Пріоритетні напрямки та вектори розвитку світової науки» 06.06.2025, УкрІНТЕІ, Суми, Україна (подано до друку).

ПЕРЕЛІК ПОСИЛАНЬ

1. Python Software Foundation. Python 3 Documentation. [Електронний ресурс] — Режим доступу: https://docs.python.org/3/

2. Tkinter — Python interface to Tcl/Tk. [Електронний ресурс] — Режим доступу: https://tkdocs.com/

3. Hipp, D. R. SQLite — Lightweight Database Engine. [Електронний ресурс] — Режим доступу: https://sqlite.org/

4. Tkcalendar Library Documentation. [Електронний ресурс] — Режим доступу: https://tkcalendar.readthedocs.io/en/stable/

5. Гнатенко, В.І., Сидоренко, О.М. Проектування інформаційних систем: Навчальний посібник. — К.: Видавництво Ліра-К, 2020. — 320 с.

6. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.

7. ДСТУ 2861-94. Безпека праці. Робочі місця при роботі з відеотерміналами і ПЕОМ. Загальні ергономічні вимоги.

8. Home Assistant Documentation. [Електронний ресурс] — Режим доступу: https://www.home-assistant.io/docs/

9. Memento Database — мобільний додаток для обліку даних. [Електронний ресурс] — Режим доступу: https://mementodatabase.com/

10. Flask Documentation — Micro Web Framework for Python. [Електронний ресурс] — Режим доступу: https://flask.palletsprojects.com/

11. Bootstrap 5 Documentation — Frontend Toolkit. [Електронний ресурс] — Режим доступу: https://getbootstrap.com/

12. Jinja2 Templating Language. [Електронний ресурс] — Режим доступу: https://jinja.palletsprojects.com/

13. W3Schools HTML & CSS Reference. [Електронний ресурс] — Режим доступу: https://www.w3schools.com/

14. Tom Schimansky. customtkinter Documentation. [Електронний ресурс] —

Режим доступу: https://customtkinter.tomschimansky.com/

15. Бондаренко О. В., Ковальчук В. В. Використання СУБД SQLite у програмних системах з обмеженими ресурсами // *Системи обробки інформації*. — 2021. — № 5 (170). — С. 58–62.

16. Іванченко І. П., Шевченко Ю. В. Розробка програмного забезпечення для ведення обліку технічного обслуговування об’єктів побуту // *Матеріали Міжнародної науково-практичної конференції «Інформаційні технології та комп’ютерна інженерія»*. — Кривий Ріг: ДВНЗ КНУ, 2022. — С. 113–116.

17. Шевченко Р. М., Сорока І. В. Побудова інтерфейсів користувача з використанням бібліотеки Tkinter // *Вісник Черкаського державного технологічного університету. Серія: Технічні науки*. — 2020. — № 2. — С. 45–50.

18. Петренко Д. М. Вибір СУБД для малих десктопних застосунків // *Збірник наукових праць студентів, аспірантів і молодих учених «Комп’ютерні науки та інженерія»*. — Вінниця: ВНТУ, 2021. — С. 89–92.

19. Козак О. В., Ткаченко Л. П. Програмна реалізація функціоналу нагадувань у сервісних застосунках // *Інформаційні технології і комп’ютерна інженерія*. — 2022. — № 1. — С. 67–71.

20. Литвин О. О., Кравчук М. Ю. Сучасні підходи до організації архівації та резервного копіювання даних у ПЗ малого масштабу // *Проблеми програмування*. — 2023. — № 2. — С. 33–37.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення термінів обслуговування домашньої техніки та інженерних систем

Виконала студентка 4 курсу
групи ПД-42
Ленда Влада Тарасівна
Керівник роботи
к.т.н. , доцент кафедри ІПЗ Довженко Тимур Павлович

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: покращити процес централізованого моніторингу, планування та обліку технічного обслуговування домашньої техніки та інженерних систем шляхом розробки настільного застосунку з системою нагадувань, журналом сервісних робіт і можливістю збереження супровідної документації.

Об'єкт дослідження: процес технічного обслуговування домашньої техніки та інженерних систем у побуті.

Предмет дослідження: застосунок для обліку, нагадування та ведення журналу технічного обслуговування побутової техніки та інженерних систем.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз предметної області та огляд аналогічних рішень.
2. Сформулювати функціональні та нефункціональні вимоги до ПЗ.
3. Спроєктувати архітектуру, структуру БД та інтерфейс користувача.
4. Реалізувати ключовий функціонал програми на Python.
5. Розробити систему нагадувань та журнал обслуговування.
6. Провести тестування та підготувати супровідну документацію.

3

АНАЛІЗ АНАЛОГІВ

Аналог	Google Calendar	Home Assistant	Centriq	Власна розробка
Нагадування	+	+	+	+
Облік техніки	-	-	+	+
Журнал ТО	-	-	-	+
Документи	-	-	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Реєстрація та редагування пристроїв
2. Планування обслуговування та нагадування
3. Ведення журналу обслуговування
4. Зберігання документів

Нефункціональні вимоги:

1. Інтерфейс адаптований для екранів від 1024x600
2. Програма повністю локалізована українською мовою
3. Дані зберігаються у локальній SQLite базі
4. Підтримується до 10 000 записів без втрати продуктивності

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

python™

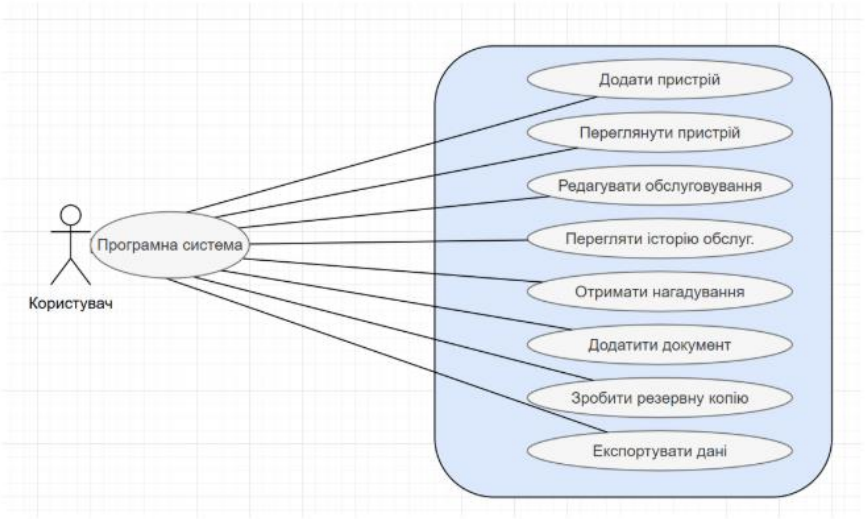
SQLite

CustomTkinter

Tkinter

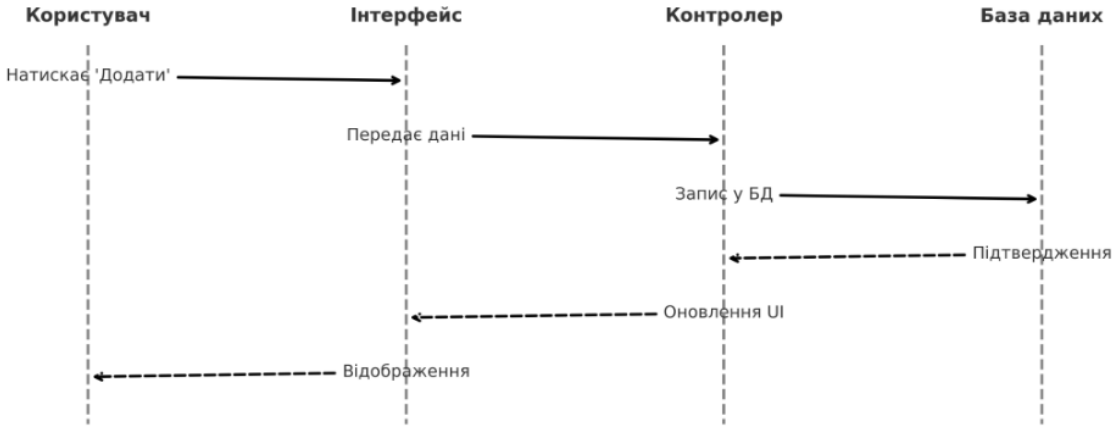
6

Діаграма варіантів використання



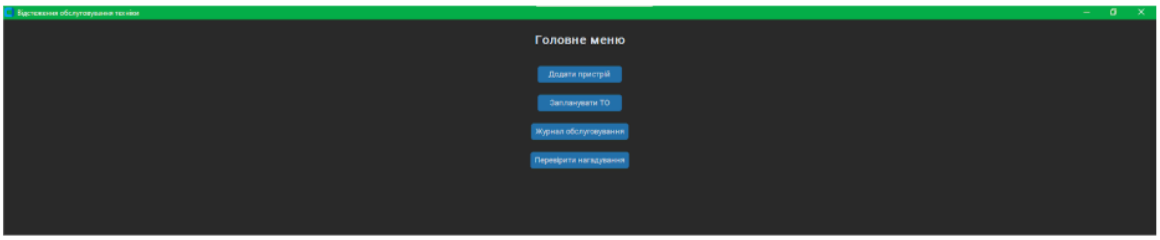
7

Діаграма послідовності

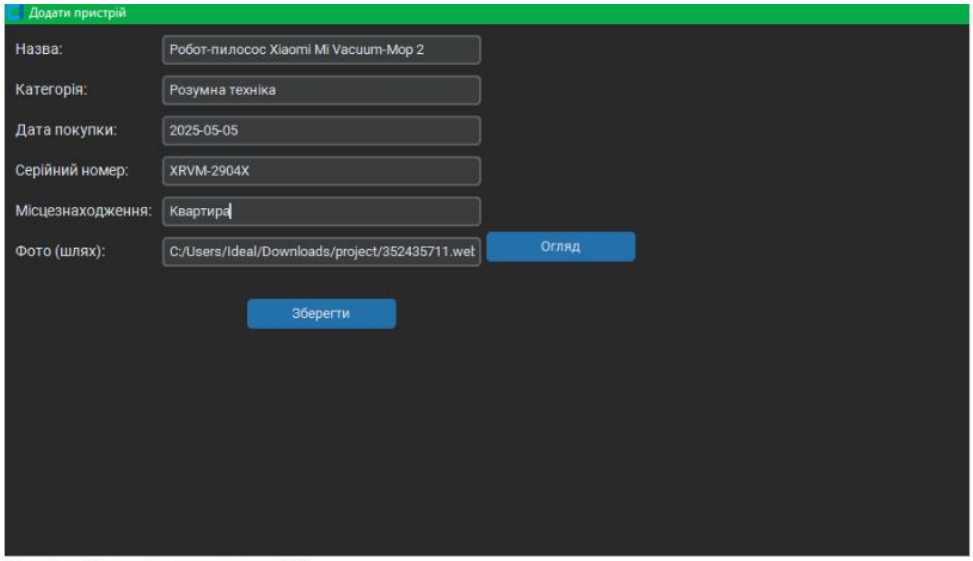


8

ЕКРАННІ ФОРМИ



Головний екран



Екран:Додати пристрій

10

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Ленда В.Т., Довженко Т.П. Інформаційна система для моніторингу обслуговування побутових пристроїв: архітектура та функціональність. Матеріали VI Всеукраїнської науково-технічної конференції “Застосування програмного забезпечення в інформаційно-комунікаційних технологіях”. 24.04.2025, ДУІКТ, Київ, Україна, с.519-521
2. Ленда В.Т., Довженко Т.П. Розробка застосунку для технічного обслуговування побутової техніки та інженерних систем. Матеріали VIII Міжнародна студентська конференція «Пріоритетні напрямки та вектори розвитку світової науки» 06.06.2025, УкрІНТЕІ, Суми, Україна (подано до друку)

ВИСНОВКИ

1. Проведено аналіз предметної області та огляд існуючих аналогів.
2. Сформульовано вимоги до програмного забезпечення та обґрунтовано вибір технологій для реалізації проєкту.
3. Реалізовано функціонал планування та контролю технічного обслуговування домашньої техніки й інженерних систем.
4. Проведено тестування, яке підтвердило стабільність роботи додатка, коректність обробки даних та відповідність заявленим функціональним вимогам.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

main.py

```
        import customtkinter as ctk
import add_device_window
import schedule_maintenance_window
import notifications
import journal_window

ctk.set_appearance_mode("Dark")
ctk.set_default_color_theme("blue")

app = ctk.CTk()
app.geometry("600x400")
app.title("Відстеження обслуговування
техніки")

ctk.CTkLabel(app, text="Головне меню",
font=ctk.CTkFont(size=20,
weight="bold")).pack(pady=20)
ctk.CTkButton(app, text="Додати пристрій",
command=add_device_window.add_device_wi
ndow).pack(pady=10)
ctk.CTkButton(app, text="Запланувати ТО",
command=schedule_maintenance_window.sch
edule_maintenance_window).pack(pady=10)
ctk.CTkButton(app, text="Журнал
обслуговування",
command=journal_window.view_maintenance
_history).pack(pady=10)
ctk.CTkButton(app, text="Перевірити
нагадування",
command=notifications.check_upcoming_main
tenance).pack(pady=10)

app.mainloop()
```

init_db.py

```
        import sqlite3

def init_db():
    conn =
sqlite3.connect('data/maintenance.db')
    cursor = conn.cursor()

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS devices
(
        id INTEGER PRIMARY KEY
AUTOINCREMENT,
        name TEXT,
        category TEXT,
        purchase_date TEXT,
        serial_number TEXT,
        location TEXT,
        photo_path TEXT
    )
    ''')

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS
maintenance (
        id INTEGER PRIMARY KEY
AUTOINCREMENT,
        device_id INTEGER,
        work_type TEXT,
        last_service_date TEXT,
        interval_months INTEGER,
        FOREIGN KEY (device_id)
REFERENCES devices (id)
    )
    ''')

    conn.commit()
    conn.close()

# ==== app.py ====
from flask import Flask, render_template, request,
redirect
import sqlite3
import os

app = Flask(__name__)

def init_db():
    os.makedirs("data", exist_ok=True)
    conn = sqlite3.connect("data/maintenance.db")
    cursor = conn.cursor()
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS devices (
        id INTEGER PRIMARY KEY
AUTOINCREMENT,
        name TEXT,
        category TEXT,
        purchase_date TEXT,
        serial_number TEXT,
        location TEXT
    )''')
    conn.commit()
    conn.close()

init_db()
```

```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/add_device', methods=['GET',
'POST'])
def add_device():
    if request.method == 'POST':
        name = request.form['name']
        category = request.form['category']
        date = request.form['purchase_date']
        serial = request.form['serial_number']
        location = request.form['location']

        conn =
sqlite3.connect("data/maintenance.db")
        cursor = conn.cursor()
        cursor.execute('''
        INSERT INTO devices (name, category,
purchase_date, serial_number, location)
        VALUES (?, ?, ?, ?, ?)''', (name, category,
date, serial, location))
        conn.commit()
        conn.close()
        return redirect('/')
    return render_template('add_device.html')

if __name__ == "__main__":
    app.run(debug=True)
-------------------------------------------------------
----------------------

# ==== templates/index.html ====
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Головна</title>
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5
.3.3/dist/css/bootstrap.min.css"
rel="stylesheet">
  <style>
  body {
      background-color: #0d1117;
      color: #c9d1d9;
      padding-top: 70px;
      font-family: 'Segoe UI', sans-serif;
  }
  .navbar {
      background-color: #161b22;
  }
  .btn-primary {
      background-color: #238636;
      border-color: #238636;
  }
  </style>
</head>
<body>
<nav class="navbar navbar-dark fixed-top
shadow">
  <div class="container-fluid">
    <span class="navbar-brand mb-0 h1">💡
Обслуговування техніки</span>
  </div>
</nav>
<div class="container text-center">
  <h1 class="mb-4 display-5">Головна
сторінка</h1>
  <a href="/add_device" class="btn btn-primary
btn-lg">➕ Додати пристрій</a>
</div>
</body>
</html>
-------------------------------------------------------
----------------------

# ==== templates/add_device.html ====
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Додати пристрій</title>
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5
.3.3/dist/css/bootstrap.min.css"
rel="stylesheet">
  <style>
  body {
      background-color: #0d1117;
      color: #c9d1d9;
      padding-top: 70px;
      font-family: 'Segoe UI', sans-serif;
  }
  .navbar {
      background-color: #161b22;
  }
  .card {
      background-color: #21262d;
      border: 1px solid #30363d;
  }
  input, select {
      background-color: #0d1117;
```

```
    color: #f0f6fc;
    border: 1px solid #30363d;
  }
  .btn-primary {
    background-color: #238636;
    border-color: #238636;
  }
  </style>
</head>
<body>
<nav class="navbar navbar-dark fixed-top
shadow">
  <div class="container-fluid">
   <a href="/" class="navbar-brand">←
Назад</a>
  </div>
</nav>
<div class="container mt-5">
  <div class="card p-5">
   <h2 class="mb-4 text-center display-6">🛠️
Додати новий пристрій</h2>
   <form method="POST">
    <div class="mb-3">
     <label class="form-label">Назва</label>
     <input name="name" class="form-
control" required>
    </div>
    <div class="mb-3">
     <label class="form-
label">Категорія</label>
     <input name="category" class="form-
control">
    </div>
    <div class="mb-3">
     <label class="form-label">Дата
покупки</label>
     <input name="purchase_date" type="date"
class="form-control">
    </div>
    <div class="mb-3">
     <label class="form-label">Серійний
номер</label>
     <input name="serial_number"
class="form-control">
    </div>
    <div class="mb-3">
     <label class="form-
label">Місцезнаходження</label>
     <input name="location" class="form-
control">
    </div>
    <button type="submit" class="btn btn-
primary btn-lg w-100">💾 Зберегти
пристрій</button>
   </form>
  </div>
</div>
</body>
</html>
```