Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська дипломна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ ОБЛІКУ МАЙНА ВІЙСЬКОВОЇ ЧАСТИНИ

Виконав: студент 4 курсу, групи 1КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Прудніков Е. Р.
(прізвище та ініціали)

Керівник: к. т. н., доцент каф. КН
Арсенюк І. Р.
(прізвище та ініціали)
«04» червня 2025 р.

Рецензент: к. т. н., доцент каф. АІІТ
Кабачій В. В.
(прізвище та ініціали)
«05» червня 2025 р.

Допущено до захисту
Зав. кафедри Яровий А. А.
«06» червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра Комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д. т. н. Яровий А. А.
"05" травня 2025 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Прудніковy Едуарду Руслановичу

1. Тема роботи: Програмний модуль обліку майна військової частини.
   Керівник роботи: Арсенюк Ігор Ростиславович, к. т. н., доц.
затверджені наказом вищого навчального закладу від «20» 03 2025 року № 97
2. Термін подання студентом роботи 30 травня 2025 р.
3. Вихідні дані до роботи
   Кількість записів у базі даних – не менше 500.
   Кількість категорій облікового майна – не менше 35.
   Кількість видів документів, генерованих програмним модулем – 3.
   Мова програмування – об'єктно-орієнтована.
   Інтерфейс – графічний.
   Операційна система – Windows.
4. Зміст текстової частини (перелік питань, які потрібно розробити):
   Аналіз можливостей сучасних хмарних сховищ; проектування програмного забезпечення для покращення роботи з хмарними сховищами; реалізація програмного забезпечення для покращення роботи з хмарними сховищами; висновки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
   - таблиця переліку основних проблем з впровадженими підходами обліку майна;
   - структурна діаграма роботи програмного модулю для обліку майна;
   - діаграма взаємодії програмного модуля з базами даних;
   - діаграма компонентів програмного модулю;
   - діаграма користувачів та їх права;

- UML-діаграма алгоритму програмного модулю обліку майна;
- приклади роботи розробленого програмного забезпечення;

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Арсенюк І. Р., к. т. н., доцент		

7. Дата видачі завдання 5 травня 2025р

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітк
		початок	закінчення	
1	Аналіз існуючих програмних модулів обліку майна військової частини та постановка задачі	05.05 2025	13.05. 2025	
2	Проектування структури програмного модуля обліку майна військової частини	14.05. 2025	26.05 2025	
3	Програмна реалізація модуля обліку майна військової частини	26.05 2025	27.05 2025	
4	Розробка інструкції користувача	28.05.25	01.06.25	
5	Оформлення матеріалів до захисту БДР	02.06.25	04.0625	

Студент _____ Прудніков Е. Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Арсенюк І. Р.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 64 сторінок формату А4, на яких є 29 рисунків, 1 таблиці, список використаних джерел містить 27 найменувань.

Бакалаврська кваліфікаційна робота є частиною комплексної бакалаврської кваліфікаційної роботи, присвяченої створенню програмного модулю обліку майна військової частини. В даній бакалаврській роботі розроблено програмний модуль ведення обліку майна для військової частин. Він має можливість пошуку і фільтрації даних, а також розширений функціонал, що передбачає формування звітів та підтримку користувацьких ролей і прав доступу. Програмний модуль забезпечує підвищення ефективності, збереження часу та зменшення впливу людського фактору під час виконання облікових операцій у військовій частині. Додаток реалізовано з використанням мови програмування Python, а для зберігання даних застосовано систему керування базами даних SQLite.

Ключові слова: автоматизація, облік майна, військова частина, програмний модуль, функціональність.

ABSTRACT

Bachelor's Qualification Thesis consists of 64 A4-format pages, containing 29 figures and 1 table. The list of references includes 27 sources.

This bachelor's qualification thesis is part of a comprehensive bachelor's qualification project dedicated to the development of a software module for asset accounting in a military unit. The presented work implements A software module for property management has been developed for military units. This module offers robust data search and filtering capabilities, along with advanced functionalities such as report generation and comprehensive support for user roles and access permissions. Its implementation aims to improve efficiency, conserve time, and mitigate the influence of human factors during property accounting operations within military units. The application is built with the Python programming language, utilizing SQLite as its database management system.

Keywords: automation, asset accounting, military unit, software module, functionality.

ЗМІСТ

ВСТУП

Актуальність досліджень. У сучасних умовах, коли в Україні триває активна фаза повномасштабної війни, відбувається масштабне розширення складу Збройних Сил, а конкуренція за людські ресурси є надзвичайно високою, питання ефективного управління матеріальними ресурсами стає особливо критичним. Зокрема, система обліку майна військових частин повинна бути здатною забезпечити оперативну, точну та надійну інформацію в умовах інтенсивного бойового застосування техніки та частих змін у структурі підрозділів. Забезпечення ефективного тилового управління, зменшення впливу людського фактору, а також недопущення втрат через помилки чи неузгодженість – все це можливе лише за умови автоматизації процесів обліку. З цією метою доцільним є створення спеціалізованого програмного модуля, який би забезпечував централізоване зберігання та обробку даних, контроль над майном, а також прозорість усіх облікових процесів. Такий підхід дозволяє не лише покращити організацію ресурсного забезпечення, а й значно підвищити загальну боєздатність та стійкість оборонного сектора до зовнішніх загроз. Беручи до уваги сучасні виклики, пов’язані з масштабністю бойових дій, високою динамікою змін у бойових підрозділах і важливістю кожної одиниці ресурсу, розробка програмного забезпечення для обліку майна є актуальним напрямом дослідження. Це рішення відповідає вимогам часу та може стати основою для формування цифрової екосистеми обліку в оборонній галузі. Таким чином, дослідження у сфері автоматизації обліку матеріальних ресурсів військових частин є не лише своєчасним, а й має стратегічний вплив на забезпечення національної безпеки України.

Метою дослідження є розширення функціональних можливостей програмного модулю обліку майна. На відміну від аналогів, які використовують застарілий тип дизайну та спосіб обліку майна, у роботі планується збільшення частки автоматизації у процесах обліку.

Об'єктом дослідження є автоматизація процесів обліку майна.

Предметом дослідження є програмний модуль обліку майна військової частини.

Задачі дослідження

1. Розглянути аналоги програмного модуля обліку майна військової частини, проаналізувати функціональні можливості, переваги та недоліки.
2. Розробити загальний алгоритм роботи програмного модулю обліку майна військової частини.
3. Розробити UML-діаграми роботи програмного модулю.
4. Обґрунтувати вибір мови, середовища програмування та бібліотек для розробки програмного модулю обліку майна військової частини.
5. Розробити програмний модуль обліку майна у військовій частині
6. Виконати тестування розробленого програмного модулю.

Апробація роботи.

Робота апробувалась на наступній конференції:

МОЛОДЬ В НАУЦІ: ДОСЛІДЖЕННЯ, ПРОБЛЕМИ, ПЕРСПЕКТИВИ (МН-2025), 15.10.24 – 14.06.25 – Вінниця: ВНТУ, 2025. Доповідь: «Програмний модуль обліку майна для військової частини» [1].

Публікації: електронні тези конференції МОЛОДЬ В НАУЦІ: ДОСЛІДЖЕННЯ, ПРОБЛЕМИ, ПЕРСПЕКТИВИ (МН-2025), 15.10.24 – 14.06.25 – Вінниця: ВНТУ, 2025 [1].

1 АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ МОДУЛІВ ОБЛІКУ МАЙНА ВІЙСЬКОВОЇ ЧАСТИНИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Обґрунтування актуальності розробки програмного модулю обліку майна військової частини

В умовах сучасної геополітичної ситуації Україна переживає складний період активної військової агресії, що призводить до кардинальних змін в оборонній сфері. Зокрема, слід відзначити значне масштабування Збройних Сил України (ЗСУ) як кількісно, так і якісно [2]. Це зростання супроводжується підвищенням навантаження на усі елементи військового забезпечення, серед яких критично важливим є матеріально-технічне забезпечення й відповідно, належний облік майна військових частин. У цих умовах питання ефективного управління ресурсами, без значних затрат людино-годин набуває особливого значення для забезпечення боєздатності, мобільності та стабільності військових підрозділів.

У зв’язку з інтенсивністю бойових дій і масштабним розгортанням сил, у ЗСУ спостерігається швидка зміна структури підрозділів, чисельності особового складу та матеріального забезпечення. Така нестабільність ускладнює ведення традиційного обліку, оскільки збільшується обсяг документації і складність координації між різними рівнями управління. Це, у свою чергу, вимагає впровадження інструментів, які здатні адаптуватися до змін, автоматизувати рутинні операції і надавати актуальну інформацію в режимі реального часу.

З огляду на велику кількість структурних підрозділів, ротацій, бойових виїздів, постачання озброєння та амуніції, виникає нагальна потреба в єдиному, централізованому та водночас адаптивному рішенні, здатному забезпечити інтеграцію обліку на всіх рівнях – від окремої бойової одиниці до командування частини. Саме тому розробка нового, сучасного програмного модуля для обліку

майна є не просто технічною задачею, а елементом цифрової трансформації оборонного сектору.

Згідно з коментарями експертів [3], традиційні методи обліку, що засновані на ручному внесенні даних у паперові журнали або застарілі електронні таблиці, вже давно не відповідають вимогам часу. Вони не здатні забезпечити ані оперативності, ані точності в умовах, коли інформація про переміщення майна, зміну його технічного стану чи потреби в заміні має надходити й оброблятись у реальному часі. Крім того, такі системи схильні до людського фактору, що значно підвищує ризик помилок, дублювання або втрати даних. Ці недоліки особливо критичні в умовах ведення бойових дій, коли будь-яке зволікання або неточність в обліку може мати безпосередні наслідки для бойових завдань.

Також важливий чинник актуальності – це наявність зовнішніх і внутрішніх кіберзагроз, що обумовлює потребу у впровадженні сучасних засобів шифрування, автентифікації, контролю доступу та запис дій користувачів. Лише за умови реалізації багаторівневої моделі захисту можна гарантувати збереження чутливих даних, унеможливити несанкціоновану зміну записів та виключити можливість внутрішніх махінацій, пов’язаних із недостовірним відображенням наявності чи переміщення майна.

Додатково, жорстка конкуренція за людський ресурс у ЗСУ створює надлишкове навантаження на кадрові та матеріально-технічні служби, які змушені виконувати дедалі більше функцій з обмеженими ресурсами. Автоматизація обліку майна сприяє значному зменшенню бюрократичних витрат часу і сил, що дозволяє перекласти увагу фахівців на значно критичніші та важливіші задачі.

Застосування автоматизованого програмного забезпечення для обліку майна дозволить реалізувати низку ключових переваг. Зменшення людського фактору з рутинних операцій значно зменшить кількість помилок та пропусків. Можливість миттєвого внесення змін у базу даних дозволить швидко реагувати на зміни у складі майна та потребах підрозділів. Консолідація інформації на одному ресурсі забезпечить прозорість і зручність контролю зі сторони командування.

Автоматизація звітності і документообігу звільнить час персоналу для виконання більш складних завдань. Використання сучасних методів автентифікації і шифрування гарантуватиме захист інформації від несанкціонованого доступу [4].

Впровадження автоматизованої системи обліку майна має також вагоме соціально-економічне значення. Зниження витрат на ведення документації, запобігання втратам та оптимізація логістичних процесів ведуть до раціональнішого використання державних ресурсів. Це особливо важливо в умовах військового конфлікту, коли кожна одиниця техніки, кожен комплект спорядження, кожне людське життя має критичне значення. Крім того, підвищення прозорості процесів обліку зменшує ризики корупційних проявів і сприяє формуванню довіри до військового керівництва серед особового складу та громадськості.

Тож, розробка програмного модуля обліку майна військової частини є надзвичайно актуальною задачею. Автоматизація облікових процесів сприятиме підвищенню точності, оперативності та прозорості управління матеріальними ресурсами, мінімізує людський фактор, а також забезпечить надійний контроль і захист інформації. Упровадження такого програмного рішення є не лише технічним завданням, а й важливим кроком у підвищенні обороноздатності держави.

Відповідно, є критична необхідність у розробці та впровадженні ефективного програмного продукту, який відповідає викликам часу та специфіці військової діяльності, забезпечуючи тим самим оптимізацію тилового забезпечення в умовах сучасної війни.

1.2 Аналіз предметної області обліку майна у військовій частині

Забезпечення підрозділів Збройних Сил України матеріально-технічними ресурсами є фундаментальним елементом підтримання бойової готовності та ефективного функціонування військової інфраструктури. У цьому контексті особливе місце посідає облік майна – процес, що охоплює фіксацію, обробку, зберігання і контроль інформації про всі об'єкти матеріального забезпечення в межах

військової частини. Облік майна виступає не лише адміністративною функцією, а й критично важливим елементом тилового управління, безпосередньо пов'язаним з логістикою, плануванням, аудитом і забезпеченням прозорості у використанні ресурсів.

Облік майна у військовій частині - це цілісний комплекс заходів, процедур, правил і засобів, спрямованих на об'єктивне та систематизоване документування наявності, переміщення, зносу, ремонту та списання матеріальних засобів. Основними завданнями обліку майна, є підтримання актуального стану інформації щодо наявності майна, запобігання втратам, крадіжкам, дублюванню та помилковому використанню ресурсів, забезпечення прозорості руху матеріальних цінностей, формування статистичної звітності для вищих органів управління, а також ухвалення управлінських рішень на основі достовірних даних; таким чином, облік виступає не лише засобом контролю за ефективним використанням майна, але й важливим інструментом внутрішнього аудиту та складовою формування мобілізаційного потенціалу військової частини.

У Збройних Силах України майно поділяється на низку категорій залежно від його функціонального призначення, фізико-технічних характеристик та облікових особливостей. Умовно його можна поділити на такі групи [5]:

1) Озброєння та військова техніка (ОВТ) – це категорія майна, що включає стрілецьку зброю, бойові машини, артилерійські системи, ракетне озброєння, засоби ППО, транспортну техніку. ОВТ є найважливішим і найбільш ресурсозатратним елементом забезпечення військової частини, потребує точного обліку технічного стану, графіків обслуговування, ремонту, модернізації та бойової готовності.
2) Засоби індивідуального захисту та екіпірування – шоломи, бронежилети, форма, взуття, захисні окуляри, рюкзаки тощо. Їхнє своєчасне видавання, заміна та збереження мають критичне значення для збереження життя та ефективності особового складу.

3) Засоби зв’язку та навігації – радіостанції, передавачі, шифрувальне обладнання, супутникові термінали, GPS-навігатори. Ці засоби забезпечують оперативне управління, координацію дій та захист комунікацій від втручання противника.
4) Тилове та господарське майно – меблі, інструменти, генератори, кухонне обладнання, будівельні матеріали, медичне спорядження, техніка для складів та майстерень. Це майно забезпечує базову життєдіяльність військової частини.
5) Матеріали, що швидко споживаються – паливно-мастильні матеріали, боєприпаси, продукти харчування, медикаменти, запчастини. Особливість обліку таких матеріалів полягає у постійній змінності їх кількісного складу та критичності до своєчасного поповнення.

Організація обліку у ЗСУ є багаторівневою. На рівні роти/взводу ведеться первинний облік (журнали, відомості, квитанції). На рівні батальйону/бригади функціонують тилові підрозділи, які акумулюють інформацію, аналізують звіти, відповідають за централізовану звітність. Вся система підпорядкована тиловому управлінню оперативного командування або Генерального штабу.

Облік регламентується низкою нормативних актів, зокрема:

- Наказами Міністерства оборони України (наприклад, Наказ № 440 "Про затвердження Інструкції з обліку військового майна") [6];
- Статутами та відомчими положеннями;
- Вимогами Державної аудиторської служби;
- Військовим законодавством щодо мобілізації та матеріального резерву.

Також існує багато системних проблем, таких як:

- облік різних типів майна за різними методиками, що ускладнює централізований контроль;
- помилки, недбалість, свідомі зловживання військовослужбовцями при ручному веденні документації;
- низький рівень автоматизації;

- складність інвентаризації;
- проблеми безпеки даних;

Облік майна у військовій частині є надзвичайно важливою та складною сферою, що охоплює логістику, нормативне регулювання, цифровізацію та менеджмент. Ретельне і точне ведення обліку є не просто формальністю, а фактором, що безпосередньо впливає на бойову ефективність, оперативність дій та захист особового складу. Відповідно виникає необхідність створення сучасних інтегрованих рішень – автоматизованих модулів, що здатні централізовано вести облік, надавати актуальні звіти, забезпечувати багаторівневий захист даних і адаптуватися до умов воєнного часу.

1.3 Огляд та аналіз існуючих підходів та програмних додатків обліку майна військової частини

Розбір різноманітних сучасних підходів до обліку майна в ЗСУ, а також порівняння з передовими світовими практиками дасть змогу виявити слабкі місця існуючих рішень та обґрунтувати необхідність розробки нового програмного модуля для автоматизації цього процесу.

У сучасних умовах активної фази повномасштабної війни в Україні система обліку майна військових частин набуває стратегічного значення для обороноздатності країни. Питання захищеного, оперативного і надійного обліку матеріальних ресурсів є не лише питанням ефективного тилового забезпечення, але й безпосередньо впливає на спроможність військових підрозділів виконувати бойові завдання. Водночас, враховуючи масштабне зростання особового складу, структурні зміни у військах та необхідність стрімкого переоснащення – традиційні підходи до обліку майна демонструють низку критичних проблем, які призводять до втрат, неузгодженості і надмірного впливу людського фактору.

Облік військового майна є комплексним процесом, що включає реєстрацію, контролювання переміщення, зберігання і списання матеріальних цінностей. У

мирний час ці процеси можна відносно легко налагодити за допомогою централізованих баз даних та традиційних методів документообігу. Проте у воєнний час, особливо в умовах активних бойових дій та частих перестановок особового складу, система обліку стикається з низкою унікальних викликів. Наприклад часті переміщення підрозділів, стали вже класикою сьогодення. Що вже говорити про зміни командування, й те що отримання/втрати майна на полі бою вимагають миттєвого оновлення інформації. Серед значних проблематик теж є відсутність стабільного інтернет-з'єднання на передовій, брак комп'ютерної техніки та кваліфікованих операторів. Це підкріплюється емоційним та фізичним навантаження на військовослужбовців що й призводить до помилок, втрат документів, зловживань. Не слід забувати, що системи обліку мають бути захищені від кібератак, але в той же час бути доступними для користувачів у складних умовах. Й на додачу різні підрозділи можуть користуватись різним рівнем технічного забезпечення і різними системами, що ускладнить централізований контроль.

Станом на 2024-2025 роки в ЗСУ використовуються декілька ключових інформаційних систем для ведення обліку майна:

Автоматизована Система Управління Тилом – комплексне рішення для логістики і тилового забезпечення, що включає модулі обліку майна [7].

Електронний облік майна – система, що впроваджується для ведення електронного документообігу у військах [8].

Та різноманітні розробки Міноборони та приватних IT-компаній – що були локальними програмними продуктами, які мали вузьку функціональність, орієнтовану на окремі види майна або підрозділи.

Незважаючи на існування цих систем, ряд досліджень та відгуків користувачів виявляють такі спільні проблеми в АСУТ та ЕОМ як:

- Недостатня інтеграція через що системи часто функціонують окремо, не обмінюються даними між собою, що ускладнює загальне управління.
- Обмежена мобільність та відсутність зручної клієнтської частини, що критично важливо для роботи на фронті.

- Відсутність автоматизації – багато процесів обліку покладені на ручну перевірку, що збільшує ймовірність помилок.
- Погана адаптація до змін у структурі та відсутність гнучкості у налаштуванні систем під нові потреби військових частин.
- Труднощі з навчанням персоналу через складність інтерфейсів та процесів призводить до необхідності тривалого навчання.

Ці проблеми призводять до неефективності, втрат матеріальних ресурсів та уповільнення тилового забезпечення, що є небезпечною реальністю в якій змушене існувати сучасне українське військо.

Logistics Functional Area Services (LOGFAS)– це модульна інформаційна система, розроблена НАТО для координації логістики між країнами-членами та партнерами. Система дозволяє стандартизовано планувати, відслідковувати, управляти й аналізувати переміщення та стан матеріальних ресурсів, техніки, особового складу, транспортних потоків тощо.

Аналіз міжнародного досвіду на прикладі LOGFAS, демонструє, що сучасні збройні сили провідних країн світу активно впроваджують комплексні автоматизовані системи управління майном, базовані на сучасних інформаційних технологіях таких як:

- Використання RFID та штрих-кодів для ідентифікації та відстеження руху майна у реальному часі [9].
- Забезпечення централізованого доступу через хмарні сервіси з різних пристроїв і підвищення стійкості систем.
- Мобільні додатки та офлайн-режими щоб забезпечити роботу навіть у зоні з обмеженим чи відсутнім зв’язком.
- Простий та інтуїтивно зрозумілий інтерфейс для швидкого навчання та використання.
- Здійснення аналітики на основі великих об’ємів даних для прогнозування потреб, оптимізації запасів і виявлення аномалій (стратегічне рішення доступне при виконанні попередніх) [10].

Такі підходи суттєво підвищують точність і оперативність обліку, зменшують людський фактор і забезпечують прозорість процесів. Враховуючи вище наведений аналіз отримаємо таблицю 1.1, що ілюструє основні проблеми з ефективністю та автоматизацією процесів обліку майна в ЗСУ.

Таблиця 1.1 – Перелік основних проблем з впровадженими підходами обліку майна

Категорія проблеми	**Конкретний прояв**	**Наслідки**	**Критичність ризику (1-5)**
Інтеграція систем	Розрізнені платформи без єдиної бази даних	Дублювання інформації, розбіжності даних	5
Мобільність	Відсутність або слабка підтримка мобільних пристроїв	Обмежений доступ до обліку на фронті	4
Автоматизація верифікації	Ручний облік та перевірка	Помилки, затримки в оновленні інформації	5
Адаптивність системи	Жорстка структура програмного забезпечення	Неможливість швидко реагувати на зміни	4
Навчання користувачів	Складність інтерфейсів і процесів	Затрати часу та зниження ефективності	3
Безпека даних	Недостатній захист інформації	Ризик несанкціонованого доступу	4
Залежність від інфраструктури	Необхідність стабільного інтернет-з'єднання	Неможливість роботи у польових умовах	5

Аналізуючи сучасні підходи до обліку майна в Збройних Силах України та порівнюючи їх із передовими світовими практиками, слід зробити декілька висновків.

Перш за все, найбільшою проблемою залишається відсутність належної інтеграції між окремими програмними комплексами, а також низька мобільність існуючих рішень. Через це відбуваються втрати інформації, які позначаються не лише на рівні технічної обробки даних, а й безпосередньо впливають на логістичні

ланцюги забезпечення військових підрозділів. Дублювання облікових записів і документів у різних підсистемах породжує плутанину та знижує оперативність реакції тилу. Такі прогалини у системі даних будуть спричиняти затримки у доставці боєприпасів, медикаментів чи технічного обладнання, що у бойових умовах є неприпустимим і загрожує життю військовослужбовців.

Додатково суттєвою проблемою є людський фактор, який залишається слабкою ланкою в системі обліку. Складні та неінтуїтивні інтерфейси програмного забезпечення збільшують ймовірність помилок під час введення даних, особливо у стресових умовах фронту. Тривалі і ресурсоємкі навчання персоналу створюють додаткові організаційні навантаження на тилові підрозділи, які і без того перебувають у стані наднапруги. Це стає причиною для низької мотивації та втрати продуктивності, що вкрай небажано в умовах масштабних військових операцій.

Вкрай важливий аспект, це застосування сучасних інформаційних технологій - таких як хмарні сервіси, мобільні додатки, штучний інтелект для виявлення аномалій у даних, а також інтернет речей для автоматичного контролю за переміщенням майна – є не просто бажаним, а критично необхідним. Вони здатні значно підвищити надійність і оперативність обліку, забезпечити прозорість процесів та суттєво знизити вплив людського фактору. Такий технологічний прорив у системах обліку військового майна створить умови для швидкої адаптації тилового забезпечення до динаміки бойових дій, що є визначальним для підтримки ефективності ЗСУ у довгостроковій перспективі.

Тож розробка нового програмного модуля, що враховуватиме специфіку військових операцій, гнучкість, безпеку та інтеграцію з існуючими системами, є не лише актуальною, а й необхідною для підтримки обороноздатності країни. Таким чином, подальші дослідження та розробки мають бути спрямовані на створення ефективного, зручного та масштабованого програмного забезпечення, яке автоматизуватиме процеси обліку майна і відповідатиме специфічним потребам Збройних Сил України.

1.4 Постановка задачі дослідження

Метою дослідження, як було зазначено у вступі, є розширення функціональних можливостей обліку майна. Тобто це передбачає створення нового програмного продукту - модуля, який забезпечить ефективну автоматизацію облікових процесів, мінімізує людський фактор та бюрократичне навантаження, й відповідно підвищить продуктивність роботи тилових підрозділів.

Як й було зазначено в попередніх розділах, система обліку майна в ЗСУ повинна виконувати не просто роль реєстру активів, а й стати інтелектуальним інструментом, що забезпечує оперативний і безпомилковий контроль над матеріальними ресурсами.

Тож основні виклики, які потребують вирішення, пов’язані із забезпеченням мають виглядати ось так:

- Безпека інформації та захищеність від несанкціонованого доступу;
- Надійність та доступність даних у будь-яких умовах, включно із бойовими операціями в зоні активних дій;
- Гнучкість в налаштуванні рівнів доступу відповідно до військових посад і функцій;
- Можливість оперативного оновлення даних із великим обсягом інформації;
- Простота та інтуїтивна зрозумілість інтерфейсу для користувачів із різним рівнем технічної підготовки;
- Автоматична генерація супровідної документації, що спрощує бюрократичні процедури.

Ці вимоги безпосередньо витікають із викликів, описаних раніше, де наголошено на необхідності мінімізації людського фактору, уникненні втрат через помилки і недоліки існуючих систем, а також на підвищенні прозорості і контролю. Тож давайте детальніше розберемо що це все значить на практиці.

У військовому секторі безпека даних – це питання першочергової важливості. Система обліку майна має забезпечувати багаторівневу, надійну систему

автентифікації користувачів із застосуванням сучасних криптографічних методів. Обов'язково має бути реалізовано захист від несанкціонованого доступу через використання ролей і прав, які жорстко контролюють, хто і які операції може виконувати. Запис всіх дій користувачів для забезпечення аудиту та можливості розслідування інцидентів, буде ключем для контролю роботи модуля. Захищена авторизація не лише підвищить довіру до системи, але й знизить ризики інформаційних витоків, що можуть бути використані ворожими агентами чи хакерами.

В умовах бойових дій централізовані системи часто виявляються надмірно вразливими до збоїв, втрати зв'язку або кібератак. Тому постановка задачі включає організацію зберігання даних у децентралізованих базах, які можуть функціонувати автономно. Також забезпечення синхронізації інформації між різними вузлами у моменти стабільного з'єднання. Використання резервного копіювання, дозволить відновлювати дані у разі часткових збоїв системи. Тож децентралізація забезпечить безперервність роботи системи у складних умовах і зменшить залежність від одного кластеру.

Військові структури суворо регламентовані за ієрархією, тому система обліку повинна враховувати це й включати жорстку систему рівнів доступу відповідно до посад та функцій військовослужбовців, що одночасно обмежить доступу до певної інформації лише тим особам, яким вона необхідна для виконання службових обов'язків. Теж ключем буде забезпечення розмежування операцій: перегляд, редагування, внесення нових записів, затвердження змін, створення нових аккаунтів, чи редагування існуючих тощо. Саме такий підхід мінімізує ризики внутрішніх витоків і помилок, підтримуючи дисципліну і порядок у веденні обліку.

Облік майна в умовах інтенсивних бойових дій потребує швидкого оновлення великих обсягів інформації, що передбачає розробку інструментів для масового імпорту даних із зовнішніх джерел (наприклад, CSV, Excel, інтеграція з іншими інформаційними системами тилу). Важливим теж є автоматичне розпізнавання помилок чи валідація даних при імпорті, забезпечення можливості групового

редагування для прискорення процесу оновлення. Такі функції необхідні для підтримки актуальності обліку у режимі реального часу, коли неактуальна інформація може призвести до оперативних помилок.

Ключовим аспектом є те що складність інтерфейсів існуючих систем часто призводить до помилок і надмірної витрати часу. Для максимального спрощення роботи користувачів система має використовувати інтуїтивно зрозумілий графічний інтерфейс, що врахує всі типові сценарії роботи військовослужбовців та забезпечить мінімум кроків для виконання основних операцій, з урахуванням обмеженого часу і стресових умов. Теж важливим буде надавати можливість швидкого навчання нових користувачів за допомогою вбудованих підказок і документації. Інтерфейс має стати фактором підвищення продуктивності, а не додатковим джерелом помилок.

Однією з найбільш ресурсозатратних і одночасно критично важливих операцій є ведення документації, яка підтверджує факт володіння, передачі, ремонту або списання майна. Автоматизація цього процесу передбачає автоматичне формування стандартних форм і звітів відповідно до вимог військового управління разом із інтеграцію з електронним документообігом для швидкого розповсюдження і затвердження документів. Основними функціями буде забезпечення можливості генерації як індивідуальних, так і пакетних документів із збереженням повної історії змін з підтримкою друку і експорту в популярні формати (PDF, DOCX).

1.5 Висновок

У контексті активної фази військових дій та трансформації системи забезпечення Збройних Сил України, зростає потреба у впровадженні сучасних цифрових рішень, що здатні підтримати управління матеріально-технічними ресурсами. Серед найбільш критичних сфер є облік майна у військових частинах, який безпосередньо впливає на логістику, мобільність, прозорість та оперативність ухвалення рішень. Аналіз актуальності проблематики, а також глибинне вивчення

предметної області дозволило сформулювати чітке завдання – розробку та впровадження автоматизованого програмного модуля, який відповідатиме викликам сучасної військової організації.

Розширення масштабів Збройних Сил України, що зумовлене військовою агресією, створило додаткове навантаження на систему матеріально-технічного забезпечення. В умовах постійної ротації, зміни структури підрозділів, мобільності техніки та особового складу, традиційні методи обліку майна виявилися неефективними: ручне введення даних призводить до затримок, помилок, дублювання та втрати інформації. Відсутність уніфікованої системи обліку ускладнює координацію між структурними рівнями та знижує прозорість управлінських процесів. Сучасний програмний модуль, який дозволяє здійснювати облік у режимі реального часу, автоматизує документацію, забезпечує інформаційну безпеку та мінімізує людський фактор, стає критично необхідним компонентом цифрової трансформації оборонного сектору.

Облік майна у військових частинах є складним системним процесом, який охоплює повний життєвий цикл матеріальних ресурсів: від надходження до списання. Його функціональне навантаження включає фіксацію, контроль, звітність, аудит та планування. З урахуванням специфіки військової діяльності, облік повинен охоплювати різноманітні категорії майна — озброєння, амуніцію, техніку, інструменти, медичні засоби тощо — з дотриманням регламентованих правил обліку та технічного обслуговування. Ключові завдання полягають у забезпеченні актуальності даних, запобіганні втратам, унеможливленні подвійного використання ресурсів, а також у створенні достовірної основи для внутрішнього та зовнішнього контролю.

Найчастіше застосовні підходи на практиці – традиційні, які базуються на паперовому документообігу або слабо інтегрованих електронних системах (як то Excel), виявляються неефективними в умовах мобільності військових підрозділів, постійної ротації особового складу, нестабільного зв’язку, а також браку кваліфікованих фахівців та технічного обладнання. В ЗСУ нині використовуються

такі системи, як Автоматизована система управління тилом (АСУТ), Електронний облік майна (ЕОМ) та низка локальних рішень, проте всі вони мають спільні обмеження – відсутність інтеграції між собою, недостатню мобільність, складність у користуванні, ручну верифікацію даних, брак адаптивності до змін у структурі військових частин, що призводить до втрат, дублювання інформації, затримок у постачанні та ризику критичних збоїв у логістичних ланцюгах. Аналіз міжнародного досвіду, зокрема використання системи LOGFAS у НАТО, демонструє потенціал сучасних підходів: застосування RFID-міток і штрих-кодів, централізованих хмарних сервісів, мобільних застосунків з офлайн-доступом, інтуїтивно зрозумілих інтерфейсів та автоматичної аналітики на основі великих обсягів даних. Усі ці елементи забезпечують надійність, оперативність і прозорість процесів обліку, що критично важливо для ефективного функціонування армії в бойових умовах.

На основі вище поданих обставин й умов, поставлено конкретне завдання: створення автоматизованого, захищеного та масштабованого програмного модуля для обліку майна у військових частинах. Такий модуль має забезпечити: єдину централізовану базу даних з можливістю інтеграції між підрозділами; підтримку категоризації майна з урахуванням функціональних особливостей; механізми обліку переміщення, ремонту, списання й технічного стану; автоматичне формування звітності для командування; системи автентифікації, шифрування і журналювання дій користувачів. Програмний модуль повинен відповідати стандартам інформаційної безпеки, підтримувати гнучке адміністрування доступів, а також мати інтерфейс, адаптований до військових умов використання.

Очікується, що майбутній програмний продукт реалізуватиме адаптивний інтерфейс користувача, можливість віддаленого доступу, багаторівневе структурування облікових одиниць, інтеграцію з мобільними пристроями, а також підтримку офлайн-режиму з наступною синхронізацією. З огляду на підвищений рівень ризику кібератак, програмний модуль має реалізовувати криптографічний захист даних, двофакторну автентифікацію, ведення логів усіх змін і доступів до системи. Особливу увагу слід приділити резервному копіюванню та забезпеченню

безперервності функціонування системи навіть у кризових умовах, що відповідає стандартам критичних інфраструктур.

Таким чином, у межах аналізу було доведено актуальність, стратегічну необхідність та предметну обґрунтованість розробки програмного модуля для обліку майна військової частини. Проблематика охоплює не лише технічні аспекти автоматизації обліку, а й ширший спектр завдань оборонного управління: інформаційну безпеку, мобільність, контроль, прозорість і ефективність використання ресурсів. Розробка такого програмного забезпечення має стати інтегральним компонентом реформи тилового забезпечення ЗСУ в умовах війни, підвищуючи не лише якість управління, а й обороноздатність держави в цілому.

2 ПРОЕКТУВАННЯ СТРУКТУРИ ПРОГРАМНОГО МОДУЛЯ ОБЛІКУ МАЙНА ВІЙСЬКОВОЇ ЧАСТИНИ

2.1 Розробка структурної схеми програмного модуля

В основу будь-якої програмної системи покладено алгоритм – чітку послідовність інструкцій, які визначають порядок та логіку обробки вхідних даних для досягнення певної мети. Алгоритми є фундаментальними елементами автоматизації, які забезпечують повторюваність, передбачуваність та ефективність операцій. У контексті військових систем обліку майна, де необхідні надійність, безпека та масштабованість, розробка алгоритму набуває особливої ваги [11].

Основними характеристиками алгоритму є його точність (кожен крок однозначно визначений), дискретність (складений із послідовних кроків), детермінованість (однозначність виконання при однакових вхідних даних), ефективність (оптимальність за часом і ресурсами) та кінцевість (закінчується після виконання всіх дій) [12]. Для забезпечення надійного функціонування модулю обліку майна військової частини алгоритм повинен включати багатоаспектний комплекс функцій, що охоплюють управління користувачами, захист доступу, обробку операцій з майном, звітність і синхронізацію даних. Ось й нашим першим розробленим алгоритмом стане структурна схема й формі діаграми роботи програмного модулю для обліку майна.

Алгоритм має починатися з базових налаштувань, які є критично важливими для подальшої стабільної роботи модуля – встановлення шляхів до баз даних і файлів – визначення локалізації сховищ користувацьких даних, майна, операцій та налаштувань. Це задає контекст, в межах якого відбуватиметься доступ до інформації.

Першою користувача має зустрічати функція перевірки вхідних даних, яка співставить логін і хеш пароля з записами у базі, визначаючи легітимність доступу. Якщо вже заговорили за хеш, то для нього теж необхідна криптографічна функція,

яка перетворить введений пароль у хеш за алгоритмом SHA256. Хешування є стандартною практикою захисту паролів, що унеможливлює їх викрадення у відкритому вигляді.

А там де вхід в аккаунт, там й функція реєстрації нового користувача із збереженням його даних у базі, що дає змогу підтримувати динамічний список користувачів із розмежуванням прав.

Налаштування таймерів блокування – механізм захисту від небажаних спроб входу, який автоматично блокує доступ після певної кількості невдалих спроб. Таймери слугують елементом безпеки, знижуючи ризик несанкціонованого доступу.

Також слід виконати первинну ініціалізацію бази даних користувачів: якщо база відсутня, то нехай створюватиметься необхідна структура таблиць, які містять критично важливі поля – ідентифікатор, ім’я, email, хеш пароля, посада, рівень доступу і фото користувача. Такий підхід дозволяє уникнути помилок доступу та підтримувати цілісність даних.

Перед переходом в сам застосунок слід створити й функції-близнюки які відповідають за запис та зчитування налаштувань. Це дозволяє адаптувати поведінку системи без потреби зміни коду, забезпечуючи гнучкість у її роботі.

Для швидкої й комфортної роботи у системі варто додати інтерфейс вводу з автодоповненням, що спростить пошук та вибір елементів, підвищить зручність користування та скорочуючи час операцій. Це зекономить безліч часу нашим хоробрим воїнам!

Центральною функцією буде автоматизоване створення документів у форматах DOCX або PDF на основі шаблонів, що відображають процеси видання, прийняття або вилучення майна. В алгоритмі врахується правильність відмінювання слів (plural-форми) для забезпечення лінгвістичної коректності. Саме воно за задумкою автора має економити найбільше часу.

Як й було наведено в минулих розділах, для комфорту слід додати функцію що імпортує типи майна з Excel-файлів у базу даних. Це критично необхідно для

оперативного оновлення і масштабування облікової інформації без ручного введення.

Теж потрібно додати функції котрі додають або видаляють типи та категорії майна, підтримуючи актуальність каталогу.

Поділ інтерфейсів на різні рівні (найкраще якщо їх буде три Спрощений, Розширений, Повний) має бути реалізований в окремих функціях [13]. Відповідно слід додати функцію яка б відкривала вікно налаштувань, та відповідні функції для коректної роботи самих налаштувань.

Вкрай необхідною буде функція яка відкриватиме інтерфейс застосунку на головній привітальній сторінці (солдати мають відчувати що застосунок турбується про них). Й окрему функцію для меню швидкого доступу збоку, через яке буде відбуватись навігація в один клік по застосунку.

Одною з вкладок що відкривається натисканням на кнопки бокової панелі – буде вікно де виводитиметься поточний перелік майна користувача у вигляді таблиці. Візуалізація структурованих даних допомагає швидко оцінити стан ресурсів.

Наступна ж кнопка відкриватиме вікно що демонструватиме останні операції, виконані користувачем чи над його майном, з можливістю перегляду супровідних файлів. Це важливо для прозорості та контролю.

Куди ж без вкладки котра відкриватиме форму для введення даних нового документа, ініціює генерацію, а також, за потреби, запускає автоматичну синхронізацію з базою даних. Таким чином, забезпечується інтеграція документообігу з обліком.

Узагальнюючи, структурна схема роботи модулю обліку майна побудована як комплекс взаємопов’язаних функціональних блоків, які послідовно ініціалізуються та виконуються згідно сценаріїв взаємодії користувача із системою. Враховуючи вище наведений аналіз отримаємо рисунок 2.1, що ілюструє структурну діаграму роботи програмного модулю для обліку майна.

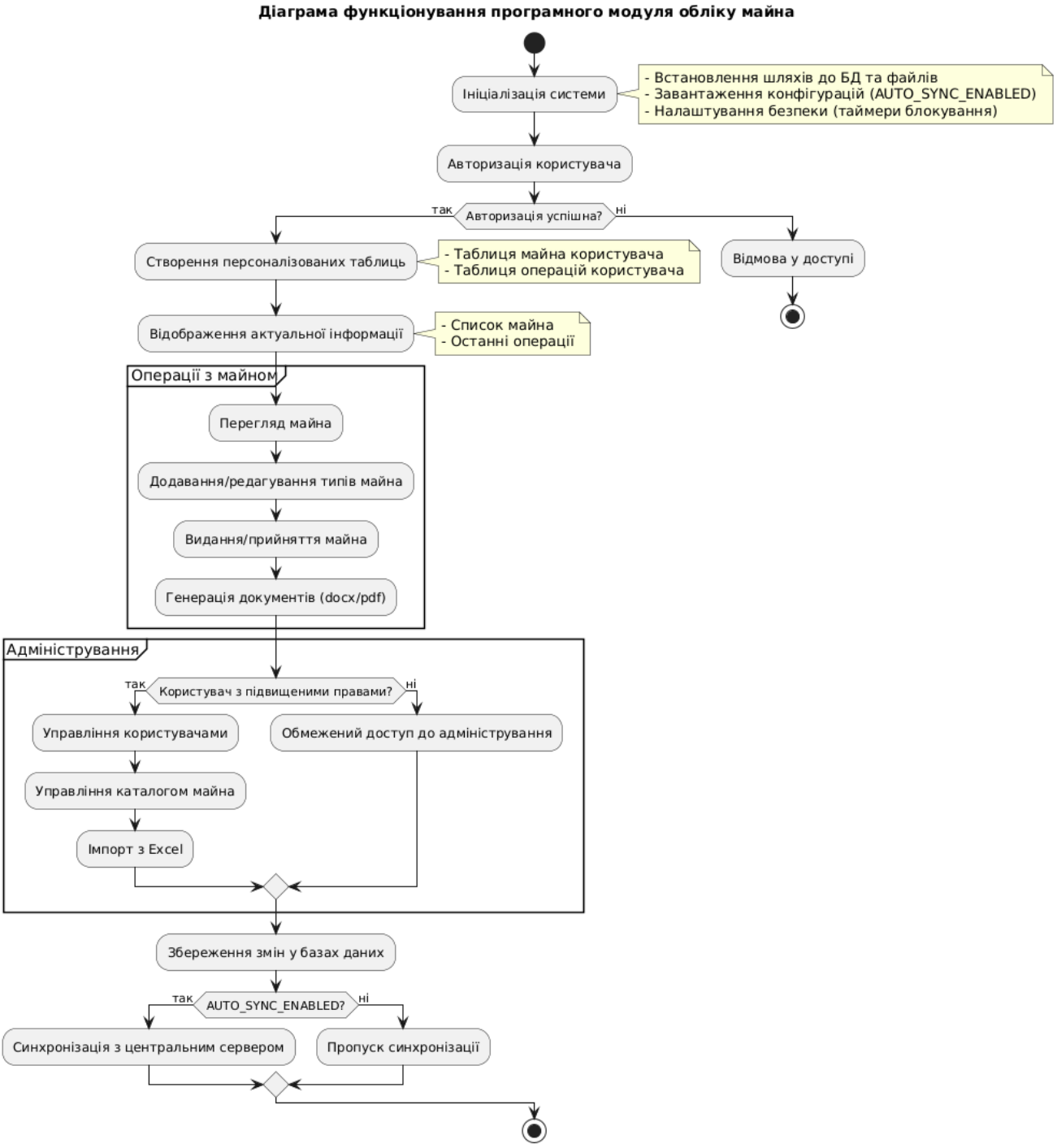

Рисунок 2.1 – Структурна діаграма роботи програмного модулю для обліку майна

Ініціалізація:

при старті системи виконується налаштування баз даних, завантаження конфігурацій і налаштування механізмів безпеки.

Авторизація користувача:

перевірка ідентифікаційних даних із використанням захищеного хешування.

Підготовка інтерфейсу:

створення персоналізованих таблиць майна та операцій, відображення актуальної інформації.

Операції з майном:

користувач може переглядати, додавати, редагувати типи майна, виконувати операції видання/прийняття, що супроводжуються автоматичною генерацією документів.

Адміністрування:

користувачі з підвищеними правами мають доступ до розширених функцій керування іншими користувачами та каталогом майна.

Синхронізація і збереження:

зміни регулярно фіксуються в базах даних, а при активованій опції – синхронізуються з центральним сервером.

Розробка алгоритму роботи програмного модулю обліку майна військової частини вимагає комплексного підходу, що забезпечує поєднання безпеки, ефективності та зручності користування. Покрокове формування функцій від початкової ініціалізації, управління користувачами, обробки операцій з майном до генерації документів і розширеного адміністрування створює стійкий фундамент для стабільного функціонування системи. Застосування криптографічних методів, гнучких налаштувань та багаторівневої ієрархії доступу відповідає сучасним вимогам захищеності та прозорості обліку військового майна.

Завдяки описаному алгоритму модуль може інтегруватись у більші інформаційні системи Збройних Сил України, підтримуючи надійний облік матеріальних ресурсів і сприяючи оперативному управлінню.

2.2 Розробка UML-діаграм баз даних, компонентів, користувачів та їх прав програмного модуля

У процесі створення програмного модуля для обліку майна військової частини надзвичайно важливим є систематичне та структуроване проєктування програмної архітектури. Це забезпечує логічність, передбачуваність і масштабованість усіх компонентів, що мають працювати у чітко визначеному контексті Збройних Сил України. Зважаючи на те, що даний модуль виконує функції обліку, документообігу та управління доступом, його архітектура повинна бути не лише гнучкою, але й відповідати вимогам безпеки та надійності.

У традиційних системах UML-діаграми класів використовуються для моделювання об’єктно-орієнтованої структури, однак в нашому випадку архітектура побудована переважно на процедурному та функціональному підході, де головну роль відіграють взаємодії між окремими базами даних, користувачами та функціональними модулями. Тому замість діаграм класів доцільніше використовувати діаграми взаємодії з БД, компонентні діаграми та діаграми доступу, що відображають реальні процеси у системі.

У процесі розробки програмного модуля обліку майна військової частини особливу увагу приділено структурному моделюванню системи, що забезпечує її надійність, масштабованість та відповідність специфічним вимогам військового середовища. З огляду на характер задачі, було прийнято рішення використовувати UML-діаграми для візуалізації архітектури системи, зокрема:

- Діаграми взаємодії між базами даних та кодом, які відображають структуру збереження та обробки даних;
- Діаграми компонентів, що ілюструють взаємодію між функціональними модулями системи;
- Діаграми рівнів доступу, які демонструють ієрархію прав користувачів та механізми контролю доступу.

Структура баз даних

Система використовує три основні бази даних, кожна з яких виконує специфічні функції:

users.db: містить інформацію про користувачів системи.

helpinfo.db: зберігає дані про майно, закріплене за користувачами.

operation.db: фіксує історію операцій з майном.

Опис таблиць та їх атрибутів

Таблиця users (users.db):

id: унікальний ідентифікатор користувача;

ім’я: повне ім’я користувача;

email: електронна адреса;

хеш пароля: захищений хеш пароля;

посада: посада користувача;

рівень доступу: визначає права користувача;

фото: шлях до зображення користувача.

Таблиця asset_info (helpinfo.db):

asset_id: унікальний ідентифікатор майна;

user_id: ідентифікатор користувача, за яким закріплено майно;

назва: назва майна;

категорія: категорія майна;

тип: тип майна;

стан: поточний стан майна;

дата надходження: дата отримання майна;

дата списання: дата списання майна (за наявності).

Таблиця operations (operation.db):

operation_id: унікальний ідентифікатор операції;

user_id: ідентифікатор користувача, який виконав операцію;

asset_id: ідентифікатор майна, до якого застосовано операцію;

тип операції: тип виконаної операції (наприклад, видача, повернення, списання);

дата: дата виконання операції;

документ: шлях до пов'язаного документа (за наявності).

Взаємозв'язки між таблицями

users.id ↔ asset_info.user_id: один користувач може мати кілька одиниць майна.

users.id ↔ operations.user_id: один користувач може виконати кілька операцій.

asset_info.asset_id ↔ operations.asset_id: одна одиниця майна може бути об'єктом кількох операцій.

Ці зв'язки забезпечують цілісність даних та дозволяють ефективно відстежувати історію взаємодії користувачів з майном. Враховуючи вище наведений аналіз отримаємо діаграму на рисунку 2.2, що ілюструє діаграму взаємодії програмного модуля з базами даних.

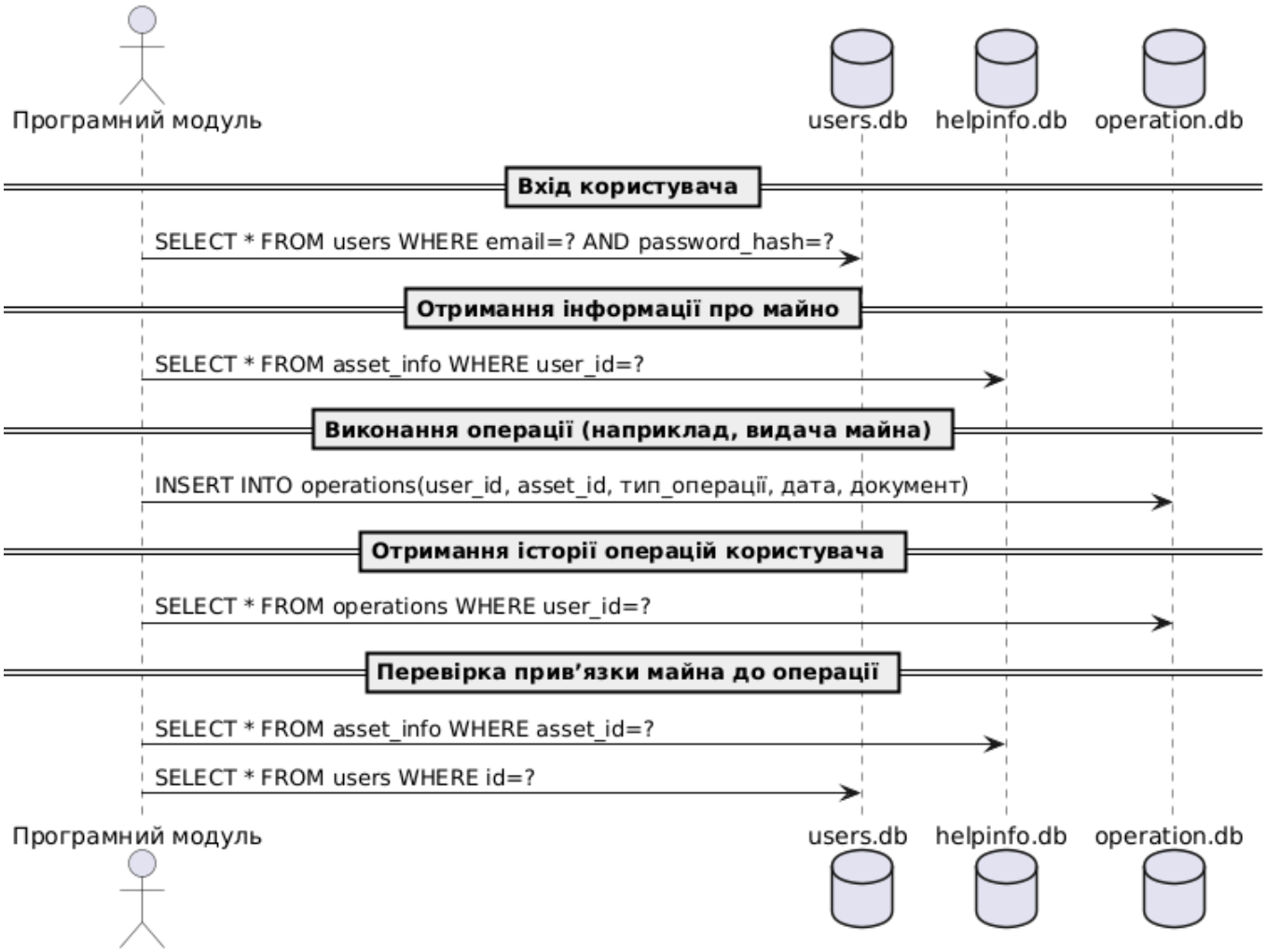

Рисунок 2.2 – Діаграма взаємодії програмного модуля з базами даних

Основні компоненти системи

Інтерфейс користувача (UI): забезпечує взаємодію користувача з системою, включаючи авторизацію, перегляд майна, створення документів тощо.

Модуль обробки даних: відповідає за логіку обробки запитів користувача, взаємодію з базами даних, генерацію документів.

Модуль безпеки: реалізує механізми захищеної авторизації, хешування паролів, управління рівнями доступу.

Модуль синхронізації: забезпечує автоматичну синхронізацію даних між локальними та центральними базами даних.

Модуль імпорту/експорту: дозволяє імпортувати дані з Excel та експортувати документи у форматах .docx та .pdf.

Взаємодія між компонентами

UI ↔ Модуль обробки даних: користувачі взаємодіють з інтерфейсом, який передає запити до модуля обробки даних.

Модуль обробки даних ↔ Бази даних: модуль обробки даних виконує запити до баз даних для отримання або збереження інформації.

Модуль обробки даних ↔ Модуль безпеки: при авторизації користувача модуль обробки даних звертається до модуля безпеки для перевірки облікових даних.

Модуль обробки даних ↔ Модуль синхронізації: при внесенні змін у дані модуль обробки даних ініціює синхронізацію з центральною базою даних.

Модуль обробки даних ↔ Модуль імпорту/експорту: при імпорті або експорті даних модуль обробки даних взаємодіє з відповідним модулем.

Ця структура забезпечує модульність системи, полегшує її обслуговування та масштабування. Враховуючи вище наведений аналіз отримаємо діаграму на рисунку 2.3, що ілюструє компоненти й роботу інфтерфейсу програмного модулю обліку майна.

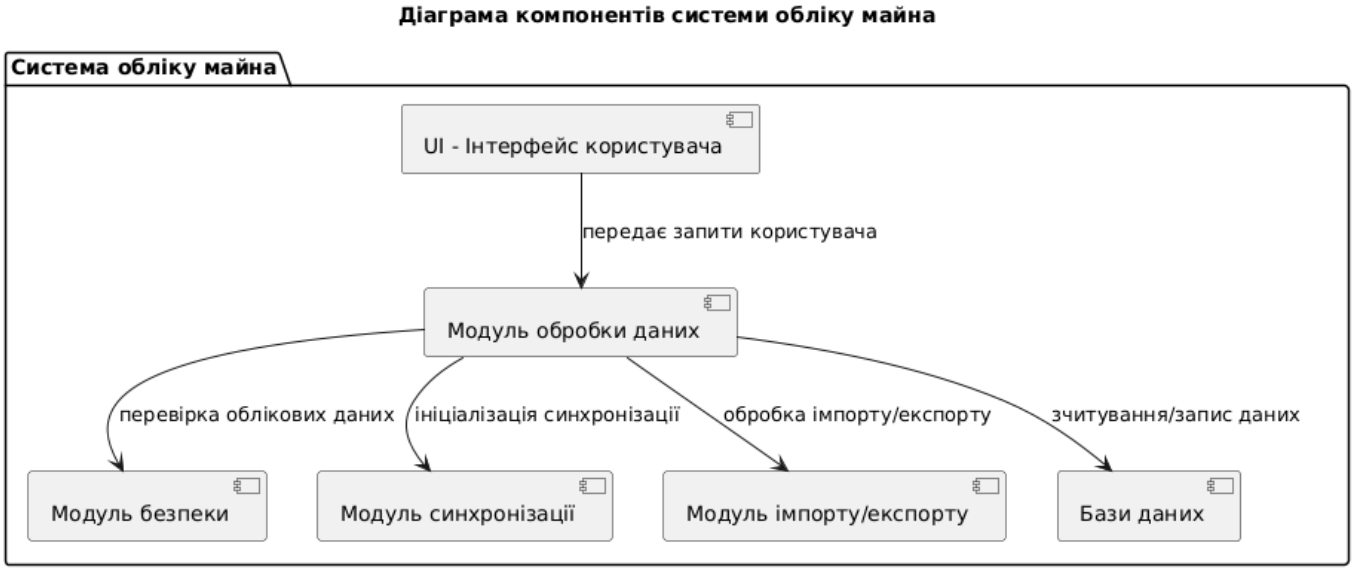

Рисунок 2.3 – Діаграма компонентів програмного модулю

Ролі користувачів та їх права

Адміністратор:

Повний доступ до всіх функцій системи;

Управління користувачами та їх правами;

Налаштування системи.

Керівник підрозділу:

Перегляд та редагування майна, закріпленого за підрозділом;

Генерація звітів та документів;

Імпорт/експорт даних.

Користувач:

Перегляд власного майна;

Ініціювання операцій з майном (наприклад, повернення);

Перегляд історії операцій.

Аутентифікація: перевірка облікових даних користувача при вході в систему.

Авторизація: визначення прав користувача на основі його ролі.

Журналювання: фіксація всіх дій користувача для забезпечення аудиту та безпеки.

Ці механізми забезпечують захист системи від несанкціонованого доступу та дозволяють контролювати дії користувачів. Враховуючи вище наведений аналіз отримаємо діаграму на рисунку 2.4, що ілюструє діаграму користувачів та їх права.

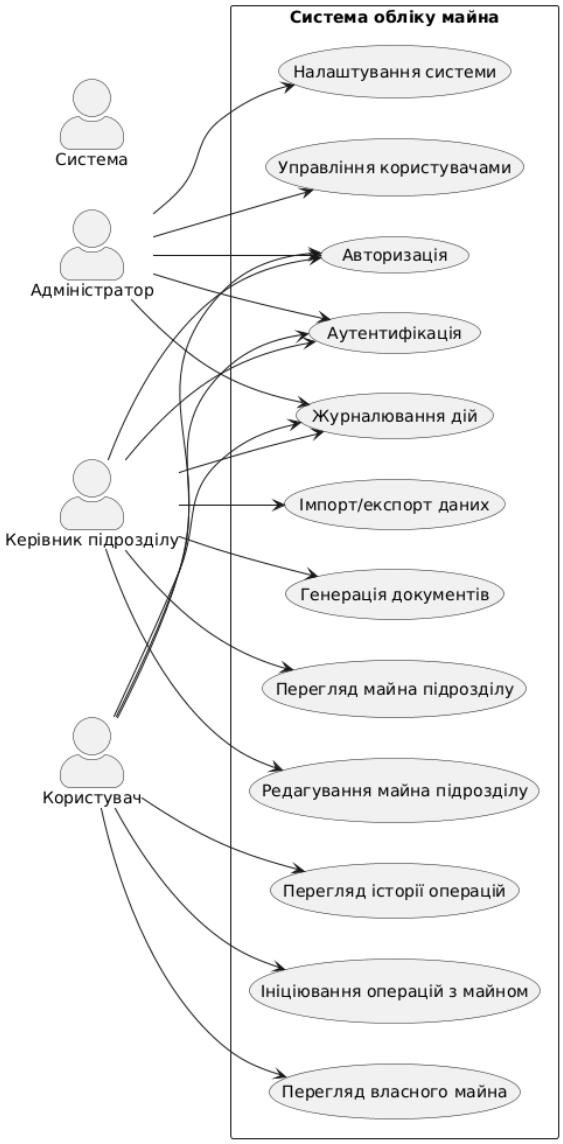

Рисунок 2.4 – Діаграма рівнів користувачів та їх прав

2.3 Розробка алгоритму роботи програмного модуля

У сучасних умовах ефективного функціонування військових частин надзвичайно важливо забезпечити високий рівень організованості, обліку та управління матеріальними ресурсами. Саме тому розробка програмного модуля для обліку майна стає ключовим завданням, що вимагає не лише ретельного проектування структури даних та зручного інтерфейсу користувача, а й точного алгоритмічного формування логіки роботи системи.

Особливу роль у цьому процесі відіграє побудова UML-діаграми алгоритму – інструменту, який дозволяє формалізовано описати всі бізнес-процеси та функціональні взаємодії з точки зору об'єктно-орієнтованого підходу та логічної послідовності дій. UML-діаграма дає змогу розбити складну систему на окремі модулі, чітко окреслити функціональні блоки, їх зв'язки та порядок виконання операцій, що значно спрощує як розробку, так і подальшу підтримку програмного продукту [14].

Насамперед, для забезпечення стабільної та безпечної роботи системи необхідно передбачити імпорт відповідних бібліотек і компонентів, які реалізують взаємодію з базами даних, криптографічні алгоритми хешування паролів, обробку документів у форматах .docx та .pdf, створення зручного графічного інтерфейсу користувача, таймерів безпеки, а також роботу з файловими структурами. Цей базовий крок створює фундамент для подальшої розробки і дозволяє програмі коректно функціонувати у різних операційних середовищах.

Далі слід чітко визначити алгоритмічну структуру системи, яка базується на ідеї модульності. Це означає, що складний процес управління ресурсами розділяється на логічні етапи, кожен із яких відповідає за певний набір функцій: автентифікацію користувачів, управління типами та категоріями майна, облік операцій з майном, генерацію документації, імпорт даних та налаштування параметрів системи [15]. Такий підхід не лише підвищує надійність, а й робить систему гнучкою та зручною у використанні.

Після проектування алгоритмічної структури й оформлення її у вигляді UML-діаграми, розробник отримає чітке уявлення про послідовність операцій, відповідальність кожного модуля та механізми взаємодії між ними. Це сприятиме зниженню ризиків помилок, полегшить тестування і пришвидшить впровадження програмного модуля у виробниче середовище військової частини.

Перш за все, варто звернути увагу на процедуру підготовки середовища, оскільки це забезпечує основу для подальшої роботи з даними. На цьому етапі передбачається імпорт необхідних бібліотек, які надають базові функції для взаємодії

з файлами, базами даних, обробкою документів і графічним інтерфейсом. Також слід визначити шляхи збереження баз даних і допоміжних файлів, адже централізоване й уніфіковане зберігання даних підвищує стабільність і безпеку системи. Налаштування списку таймерів блокування входу – це ще одна ключова складова, що підтримує механізми захисту від несанкціонованого доступу.

Далі необхідно розробити механізм ініціалізації бази даних, який гарантуватиме наявність у системі необхідних структур для збереження інформації про користувачів. Ця процедура включає перевірку існування відповідної бази даних, а у разі її відсутності – створення таблиці користувачів із відповідними полями, які зберігають ідентифікаційні дані, контактні відомості, параметри доступу, а також мультимедійні атрибути, такі як фотографія. Така ініціалізація є критичною, бо без неї подальше управління користувачами і контролю доступу буде неможливим.

Після створення базової структури для користувачів потрібно продумати функції для збереження і завантаження налаштувань системи. Зокрема, важливо мати можливість зберігати у файлі параметри автоматичної синхронізації, що впливає на безперервність і актуальність інформації про майно. Відповідно, функції для запису й зчитування цих налаштувань повинні бути розроблені з урахуванням надійності та зручності використання.

Наступний крок полягає у побудові механізмів формування таблиць із інформацією про майно та операції для кожного користувача. Для цього необхідно реалізувати функціонал, що за логіном користувача витягує його унікальний ідентифікатор із бази даних і на його основі створює або завантажує відповідні таблиці у допоміжних базах даних. Перша таблиця міститиме безпосередньо перелік майна, яке закріплене за користувачем, а друга – історію операцій з цим майном, включаючи прийняття, вилучення, ремонти тощо. Такий поділ на об’єкти та операції дозволяє максимально гнучко відслідковувати стан ресурсів і здійснювати аудит.

Для відображення цих даних у графічному інтерфейсі слід продумати окремі механізми виведення інформації. Це може бути реалізовано у вигляді таблиць, які інтегруються у спеціальні віджети інтерфейсу із можливістю прокрутки, сортування

та фільтрації. Зокрема, список майна користувача повинен виводитися з урахуванням усіх атрибутів об’єктів, а також мати інтерфейс для роботи з останніми операціями, що дасть змогу швидко переглянути та відкрити деталі кожної транзакції. Такий інтерфейс забезпечує оперативний доступ до необхідної інформації без перевантаження користувача.

Важливою частиною алгоритму є забезпечення безпеки при обробці персональних даних користувачів, особливо паролів. Для цього необхідно передбачити функціонал для їх хешування із застосуванням сучасних криптографічних хеш-функцій, які забезпечують одностороннє перетворення пароля і зменшують ризик компрометації. Окрім цього, потрібен механізм реєстрації нових користувачів із збереженням усіх їхніх атрибутів у базі даних, а також автентифікації, яка здійснюватиме перевірку відповідності введених логіну і пароля збереженим даним.

Поряд з цим, алгоритм повинен містити компоненти для управління інформацією про блокування користувачів після певної кількості невдалих спроб входу, що запобігатиме атакам типу brute-force. Для цього слід передбачити функції зчитування та запису стану блокування, які відповідально оновлюватимуть статус доступу у спеціальному сховищі.

Значущим елементом системи є введення зручних інтерфейсів для роботи з даними. Це може бути реалізовано у вигляді полів введення з автозаповненням, що прискорюють і спрощують процес вибору потрібних значень із підказок. Подібні механізми позитивно впливають на швидкість роботи користувача і зменшують кількість помилок при введенні.

Для документування операцій з майном важливо передбачити функціонал автоматичного формування офіційних документів у стандартизованих форматах, таких як DOCX або PDF. Алгоритм повинен підтримувати вставку даних у шаблони документів із коректною граматичною обробкою форм слів залежно від кількості, що забезпечує належну якість оформлення. Важливо, щоб форма для створення таких

документів була інтерактивною та надавала можливість не лише вводити необхідні дані, а й за потреби автоматично оновлювати інформацію у базі даних.

Також необхідно забезпечити можливість імпорту довідкових даних, наприклад, типів майна з файлів Excel. Це дозволить централізовано і зручно оновлювати класифікатори без ручного введення, що зменшує людський фактор і підвищує точність даних. Подальші функції мають забезпечувати отримання списків категорій майна, типів за категоріями, генерацію унікальних кодів для нових об'єктів і їх додавання або видалення, що гарантує повний контроль над каталогом майна.

Окрім цього, варто інтегрувати в алгоритм логіку вибору правильних граматичних форм для назв об'єктів у документах, що значно підвищить якість і природність текстових звітів і документів.

Для забезпечення гнучкості системи необхідно врахувати рівні доступу користувачів, які визначатимуть, які саме додаткові функції їм доступні. Залежно від рівня дозволяється відображати розширені або повні інтерфейси з можливістю додавання, редагування чи видалення користувачів і типів майна, а також імпорту даних та формування груп [16]. Такий підхід відповідає принципам безпеки і контролю доступу.

Завершальним кроком є розробка інтерфейсу налаштувань, де користувач може увімкнути або вимкнути автоматичну синхронізацію, що дозволяє оптимізувати продуктивність системи залежно від потреб конкретного користувача чи військової частини.

В результаті розробки такого алгоритму створюється комплексна, модульна, безпечна і зручна система обліку майна, яка не лише автоматизує процеси реєстрації, звітності та контролю, але й забезпечує високу якість даних і гнучкість у користуванні. UML-діаграма, побудована на основі описаних функціональних блоків, чітко відобразить усі компоненти системи, зв'язки між ними, умови виконання та переходи, що полегшить розуміння і подальшу розробку програмного продукту. Враховуючи вище наведений аналіз отримаємо діаграму на рисунку 2.5, що ілюструє діаграму агоритму програмного модулю обліку майна.

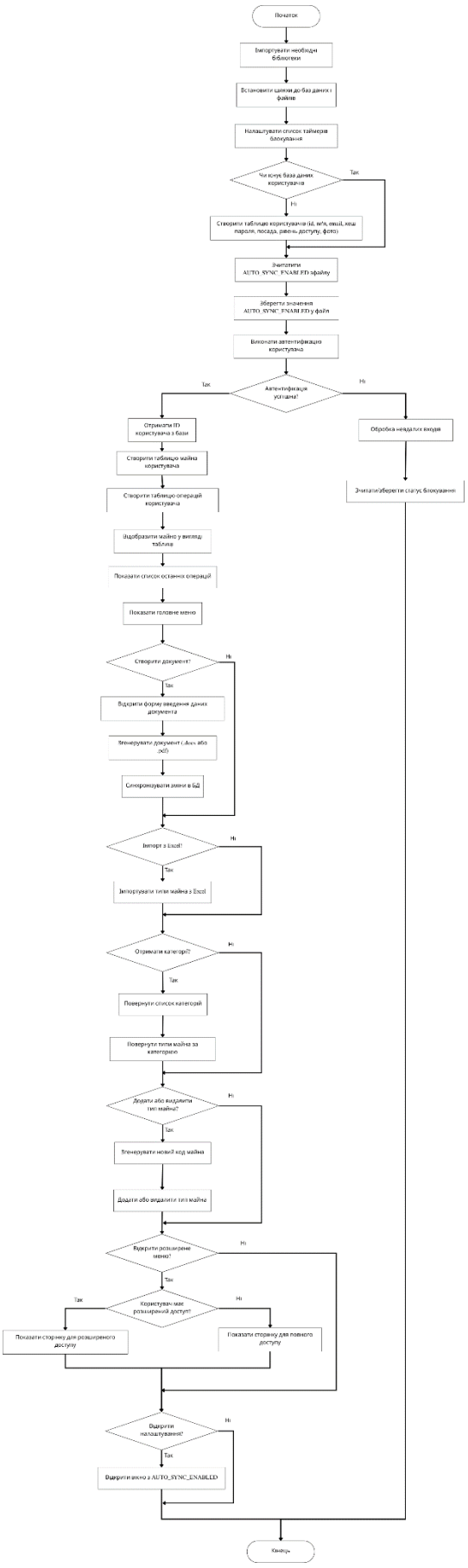

Рисунок 2.5 – Загальний алгоритм програмного модулю обліку майна

2.4 Висновок

Розробка структурного алгоритму для програмного модуля обліку військового майна стала базисом цієї науково-прикладної роботи, що об'єднала низку технологічних, функціональних та безпекових підходів у єдину логічну послідовність дій. У межах представленої концепції було детально опрацьовано структуру алгоритмічної моделі, яка охоплює всі ключові аспекти функціонування майбутнього застосунку – від ініціалізації середовища до генерації та синхронізації документів. Кожен етап цього алгоритму був побудований не лише з міркувань технічної доцільності, а й виходячи з реальних потреб кінцевого користувача – військовослужбовця, який, діючи в умовах підвищеної відповідальності та обмеженого часу, потребує інструмента, здатного гарантувати надійність, швидкість і зрозумілу логіку взаємодії.

Як зазначено в першому підрозділі – на відміну від багатьох традиційних систем, що виконують облік на базовому рівні, запропонована структура від початку враховує необхідність високого рівня безпеки, масштабованості та адаптації до потреб кінцевого користувача. Це виражається, зокрема, у використанні криптографічного хешування для зберігання паролів, механізмів блокування після невдалих спроб входу, багаторівневого управління правами доступу та можливості автоматичного формування звітної документації. В той час як існуючі рішення здебільшого оперують або ручним введенням даних, або обмеженим функціоналом, представлений підхід пропонує розширений спектр функцій, які не лише відповідають сучасним стандартам обліку, але й підтримують автоматизацію повторюваних задач. Вперше у подібних алгоритмах враховано можливість повної інтеграції облікових та звітних механізмів, включно з лінгвістичним модулем відмінювання назв майна, що є нетривіальним, але критично важливим компонентом для підготовки юридично й адміністративно коректних документів.

У другому підрозділі було здійснено комплексну розробку структурних діаграм, що слугували основою для створення програмного модуля обліку майна

військової частини, з акцентом на систематичність, логічність і масштабованість архітектури. Підхід до моделювання системи відрізнявся від традиційних об'єктно-орієнтованих практик, де домінували UML-діаграми класів. Замість цього було обрано більш релевантні для поставлених задач діаграми взаємодії з базами даних, компонентні діаграми та діаграми рівнів доступу, які дозволили чітко відобразити функціональні зв'язки між базами даних, користувачами та функціональними модулями системи. Такий вибір методів моделювання підкреслює особливості архітектури, в якій процедурний та функціональний підходи отримують пріоритет, а взаємодія з інформаційними ресурсами і контроль доступу формують центральні аспекти системи. Відмінність запропонованого рішення від існуючих полягає в адаптації класичних інструментів моделювання під специфіку військового програмного забезпечення, що має суворі вимоги щодо надійності, безпеки та гнучкості, з одночасним забезпеченням прозорості та зрозумілості структури для подальшої реалізації і супроводу.

У третьому підрозділі здійснено всебічний аналіз і систематизацію підходів до розробки алгоритму програмного модуля обліку майна, що є невід'ємною частиною сучасних інформаційних систем управління військовими ресурсами. Основною метою було створення формалізованої моделі, яка відображає логіку функціонування системи, її структурну організацію та взаємодію компонентів у межах об'єктно-орієнтованого підходу. Такий підхід дозволяє не лише забезпечити зрозумілість і прозорість архітектури програмного продукту, а й створити гнучку платформу для подальшого розвитку, модифікації і масштабування системи, що є особливо актуальним у контексті високих вимог до безпеки та надійності військових інформаційних систем.

Розробка структурного алгоритму програмного модуля обліку військового майна об'єднала технологічні, функціональні та безпекові рішення в єдину систему, що враховує потреби військовослужбовців і забезпечує надійність, швидкість і зручність роботи. Запропонована структура відрізняється підвищеною безпекою, масштабованістю та автоматизацією, включно з криптографічним захистом,

багаторівневим доступом і автоматичним формуванням звітів. Моделювання базується на діаграмах, які відображають функціональну взаємодію з базами даних і контролем доступу, відходячи від класичних UML-діаграм класів, що підвищує прозорість, гнучкість і відповідає суворим військовим вимогам.

А, формалізована UML-модель тепер відображає логіку та структуру системи, забезпечуючи можливість її подальшого розвитку і масштабування з урахуванням високих вимог безпеки та надійності.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОГРАМНОГО МОДУЛЮ ОБЛІКУ МАЙНА ВІЙСЬКОВОЇ ЧАСТИНИ

3.1 Обґрунтування вибору мови, середовища програмування та бібліотек для реалізації програмного модуля

Для реалізації програмного модуля обліку майна військової частини, ураховуючи специфіку задачі – створення багатофункціонального, безпечного і зручного в користуванні програмного комплексу для обліку майна, необхідно було застосувати мову та інструментарій, які поєднують високу продуктивність розробки, багатство стандартних і сторонніх бібліотек, гнучкість інтеграції з різноманітними системами, а також забезпечують простоту підтримки і масштабування системи. Python у цьому контексті є найбільш придатним вибором завдяки своїй універсальності, широкій підтримці робочих середовищ і активній спільноті розробників. Надалі будемо розбирати саме найважливіші складові в контексті конкретної праці.

Перш за все, Python [17] відомий своєю читабельністю та простотою синтаксису, що значно скорочує час розробки та полегшує подальшу підтримку коду. Для проєкту, який передбачає інтеграцію з базами даних, генерацію документів, роботу з файлами різних форматів, а також створення інтуїтивно зрозумілого графічного інтерфейсу, мова має бути максимально виразною, щоб розробники могли швидко й без помилок імплементувати функціонал. У порівнянні з такими мовами, як C++ чи Java, де значна частина часу витрачається на деталізацію типів, управління пам’яттю та написання шаблонного коду, Python дозволяє сфокусуватися саме на логіці предметної області. Це критично важливо для складних систем обліку майна, де потрібно враховувати велику кількість бізнес-правил, наприклад, авторизацію користувачів із різними рівнями доступу, роботу з хешуванням паролів, імпорт та експорт даних у різних форматах.

Для зберігання та обробки даних у проекті використовується мова SQL, яка органічно інтегрується з Python і забезпечує локальне збереження даних у вигляді реляційної бази даних. Вибір SQLite [18] для модуля обліку майна військової частини є обґрунтованим, адже це вбудована база, яка не потребує додаткового серверного програмного забезпечення, що значно спрощує розгортання системи в умовах військової інфраструктури з обмеженим доступом до мережі. Перевагою SQLite є її стабільність, швидкість та надійність при обробці локальних запитів, а також підтримка транзакцій, що дозволяє гарантувати цілісність даних під час виконання операцій обліку.

Середовище розробки PyCharm [19] вибране через його високу інтеграцію з Python, широкий спектр інструментів для дебагу, рефакторингу, підтримки систем контролю версій, а також зручний інтерфейс для управління пакетами і бібліотеками. PyCharm сприяє підвищенню продуктивності розробника за рахунок автоматичного завершення коду, швидкого переходу між класами і функціями, інструментів аналізу якості коду та можливості швидкого тестування. В контексті проекту це важливо, оскільки реалізація функціоналу модулю обліку майна передбачає роботу з великою кількістю функцій, складною логікою бізнес-процесів і багаторівневими структурами даних. PyCharm допомагає не лише писати код ефективно, а й підтримувати його якість, що зменшує ризик помилок, особливо в критичних функціях, таких як автентифікація користувачів або генерація офіційних документів.

У порівнянні з іншими середовищами розробки, наприклад Visual Studio Code або Eclipse, PyCharm надає більш глибоку інтеграцію з Python-екосистемою, що проявляється у вдосконаленому автокомпліті, підтримці складних структур даних і зручному дебагу. Це суттєво скорочує час розробки та знижує рівень помилок, що особливо актуально для проєкту з військовим спрямуванням, де надійність і точність мають першорядне значення. У свою чергу, порівняння з мовами, такими як Java або C#, показує, що Python є більш гнучким і менш ресурсозатратним варіантом, що дозволяє легше адаптувати програму під специфічні вимоги військового обліку майна без надмірної складності коду.

В основу цього проєкту покладено потребу в розробці гнучкого, інтуїтивно зрозумілого інтерфейсу користувача, надійної системи збереження і обробки даних, а також інтеграції з популярними форматами документів для автоматизованої звітності. Вибрані бібліотеки відображають комплексний підхід, який поєднує ефективність реалізації, відповідність стандартам безпеки, а також масштабованість системи. У тексті нижче наведено аргументоване обґрунтування вибору кожної бібліотеки з урахуванням специфіки проекту, а також проведено порівняльний аналіз з альтернативними рішеннями.

Бібліотека tkinter [20] є класичним інструментом для створення графічного інтерфейсу користувача в Python і в даному проекті вибрана з огляду на її простоту, кросплатформеність та вбудовану підтримку, що дозволяє розробляти легкі, стабільні і швидкі у виконанні GUI-додатки. Військовий модуль обліку майна вимагає надійного і зрозумілого інтерфейсу, який би не навантажував систему і був доступний на широкому спектрі робочих станцій без додаткових інсталяцій. На відміну від більш складних і ресурсомістких фреймворків, таких як PyQt або wxPython, tkinter має мінімальні системні залежності, що зменшує ризики виникнення проблем із сумісністю. Хоча PyQt пропонує ширші можливості візуального дизайну, його ліцензійні умови та вимоги до встановлення сторонніх компонентів роблять його менш привабливим для проєктів із жорсткими стандартами безпеки та корпоративного контролю. Вибір tkinter також обумовлений активною підтримкою в Python-спільноті, що сприяє швидкому вирішенню технічних питань та адаптації під нові вимоги. Компоненти ttk, які є розширенням tkinter, доповнюють інтерфейс сучасними віджетами з нативним виглядом і покращеною функціональністю, що особливо важливо для створення зручних форм введення та таблиць, необхідних для обліку майна.

Бібліотека Pillow [21] є найпоширенішим рішенням для роботи з графічними зображеннями в Python. Вона забезпечує широкий спектр функцій для обробки растрових зображень, таких як відкриття, збереження, масштабування та конвертація форматів. В контексті обліку майна військової частини це дозволяє ефективно

працювати з фотографіями об’єктів майна, ідентифікаційними зображеннями користувачів і документами. Альтернативи, як OpenCV, мають надмірний функціонал, орієнтований на складну комп’ютерну візію, що не є необхідним у даному випадку і додатково ускладнює інтеграцію. Pillow пропонує інтуїтивний інтерфейс, відносну легкість у навчанні та стабільність, а також підтримує тісну інтеграцію з tkinter через ImageTk, що спрощує відображення зображень у GUI. Це критично важливо для розробки інтерфейсу, де необхідно оперативно відображати фото і не перевантажувати користувача зайвими затримками.

Модуль sqlite3 обрано як легковагове та вбудоване рішення для зберігання даних, що забезпечує високу швидкість доступу і надійність зберігання без потреби у розгортанні складних серверних СУБД. Військова частина зазвичай має обмежені ресурси інфраструктури, тому відсутність необхідності встановлення зовнішнього сервера бази даних суттєво спрощує експлуатацію і обслуговування програмного забезпечення. SQLite підтримує більшість стандартних SQL-запитів, що дозволяє реалізувати гнучку структуру даних для користувачів, операцій і майна, а також легко інтегрується з Python без необхідності додаткових залежностей. Порівняно з PostgreSQL або MySQL, які потребують налаштування та адміністрування серверної частини, sqlite3 є оптимальним варіантом для локального і автономного використання. При цьому можливість імпорту/експорту даних у стандартних форматах забезпечує легку масштабованість і інтеграцію з іншими системами.

Для безпеки аутентифікації користувачів реалізовано використання hashlib [22] із алгоритмом sha256 для хешування паролів. Це стандартний підхід, який відповідає вимогам сучасної криптографічної безпеки, захищаючи дані користувачів від несанкціонованого доступу. Хоча існують більш складні рішення, наприклад, bcrypt або scrypt, які пропонують підвищений рівень захисту завдяки адаптивності та стійкості до атак типу brute-force, в умовах цього проєкту sha256 є достатнім компромісом між безпекою та простотою реалізації. Також використання вбудованої бібліотеки hashlib забезпечує безпеку без додаткових залежностей і гарантує

стабільну роботу на будь-яких платформах, що є ключовим для військової сфери з урахуванням вимог надійності.

Використання pandas [23] у проекті спрямоване на обробку та аналіз табличних даних, особливо під час імпорту інвентаризації майна з Excel-файлів, що часто є стандартним форматом для звітності у військових частинах. Pandas пропонує зручні структури DataFrame, які дозволяють легко і ефективно виконувати фільтрацію, сортування, агрегацію та трансформацію даних. Альтернативи, як openpyxl або xlrd, орієнтовані переважно на читання та запис Excel без просунутої аналітики, що вимагає додаткової реалізації логіки обробки. Pandas дозволяє суттєво скоротити час розробки і підвищити якість обробки даних завдяки своїй потужній функціональності, що безпосередньо впливає на точність і надійність формування звітів.

Для автоматизованої генерації документів у форматах .docx і .pdf обрано бібліотеки python-docx [24] і docx2pdf [25] відповідно. Python-docx дозволяє створювати, редагувати і форматувати документи Word програмно, що забезпечує гнучкість у формуванні офіційної документації, таких як акти прийняття, видачі або вилучення майна. Використання цієї бібліотеки забезпечує підтримку форматування, вставки таблиць, списків і налаштування абзаців, що відповідає вимогам ведення офіційних записів. Docx2pdf, у свою чергу, дозволяє швидко конвертувати Word-документи у PDF для архівації та обміну у незмінному форматі, що є стандартною практикою у військовій документації. Альтернативні варіанти, як ReportLab або LaTeX, хоча й пропонують розширені можливості генерації PDF, мають складніший синтаксис і не надають такого рівня інтеграції з існуючими шаблонами Word, що було б надмірним для поставлених завдань.

Для інтеграції системних операцій, зокрема виклику зовнішніх процесів, застосовано модуль subprocess [26]. Це стандартний і надійний інструмент для асинхронного виконання команд операційної системи, що дозволяє програмно керувати процесами без залучення сторонніх рішень. Його використання гарантує

кросплатформенність і високу сумісність з іншими компонентами проєкту, що є суттєвим у військових IT-середовищах.

Таким чином, поєднання мови Python, SQL-бази даних та середовища PyCharm забезпечує баланс між гнучкістю, продуктивністю і надійністю. Це дозволяє реалізувати програмний модуль обліку майна військової частини з багатим функціоналом, включаючи безпечне збереження та обробку даних, створення документів, автоматичну синхронізацію, а також зручний інтерфейс для користувача різного рівня доступу. А сукупність обраних бібліотек демонструє ідеальний баланс між простотою використання, надійністю, швидкістю розробки та відповідністю вимогам безпеки та стабільності, що є критично важливим для військових інформаційних систем. Вони дозволяють побудувати зручний, функціональний та легко підтримуваний програмний комплекс, який адаптується до специфіки військової служби, де простота інтеграції, автономність роботи, високий рівень захисту даних і можливість автоматизації рутинних процесів обліку є пріоритетами. Такий вибір технологій є обґрунтованим з огляду на сучасні вимоги до програмних систем у військовому секторі, що висувають високі стандарти безпеки, ефективності та підтримки. У перспективі це також відкриває можливості для масштабування та інтеграції з іншими інформаційними системами без необхідності кардинальної перебудови архітектури.

3.2 Розробка програмного модуля обліку майна

Розробка програмного модуля обліку майна військової частини є складним та багатогранним завданням, що вимагає глибокого розуміння не лише програмної інженерії, а й специфіки управління ресурсами в умовах підвищених вимог безпеки, точності та оперативності. Ефективність такого модуля визначається його здатністю інтегруватися у загальну систему управління військовою частиною, забезпечувати надійність збереження даних, простоту користування, а також гнучкість у обробці інформації про майно з урахуванням різноманітних рівнів доступу та ролей

користувачів. У цьому контексті, розробка програмного модуля обліку майна передбачає послідовну реалізацію низки функціональних та організаційних кроків, які визначають його архітектуру, структуру баз даних, логіку взаємодії з користувачем та зовнішніми файлами.

Перш за все, розробник зосередиться на формуванні надійної системи зберігання даних, що є базисом для подальшої роботи модулю. Військові частини часто оперують великими обсягами інформації про власне майно, включно із засобами техніки, обладнанням, матеріальними ресурсами, які підлягають суворому контролю. Відповідно, структура баз даних повинна бути оптимально продуманою, із чітким розмежуванням таблиць, що містять дані про користувачів, їх права доступу, перелік майна, а також журнал операцій із цим майном. Важливо передбачити створення первинних таблиць користувачів, де крім стандартних ідентифікаторів і контактної інформації зберігаються також хешовані паролі та рівні доступу, що забезпечує безпеку та розмежування прав. Концептуально важливим є використання механізму хешування паролів із застосуванням криптографічно стійких функцій, що виключає можливість прямого відновлення пароля з бази, захищаючи тим самим інформацію від несанкціонованого доступу.

Наступним етапом є розробка функціоналу, що реалізує різні рівні доступу та відповідні таблиці для користувачів залежно від їх ролі в системі. Адміністратори, оператори та звичайні користувачі повинні мати окремі зони відповідальності, і це накладає вимоги на динамічне формування таблиць із майном, що їм підпорядковане, та операцій, які вони виконують. Такий підхід дозволяє підвищити точність обліку, унеможливлює випадкові або зловмисні дії з чужим майном і забезпечує прозорість усіх операцій. Для цього у модулі слід реалізувати методи отримання ідентифікаторів користувачів за логінами, автоматичне формування відповідних робочих таблиць у різних базах даних, а також організувати зручне відображення списків майна та історії операцій у графічному інтерфейсі.

Інтерфейс користувача є не менш важливим компонентом, оскільки від його інтуїтивності і зручності залежить продуктивність праці персоналу, а також

мінімізується ризик помилок при введенні чи пошуку інформації. Забезпечення адаптивного та інформативного відображення даних, використання табличних компонентів із підтримкою автозаповнення, фільтрації і сортування сприяє ефективному навігаційному досвіду. Особливу увагу варто приділяти зручним формам для створення, редагування та пошуку документів, а також візуальному відображенню статусу операцій. Впровадження функцій автоматичного оновлення та синхронізації з базою даних дозволяє оперативно відображати зміни, підтримуючи актуальність інформації без зайвих затримок.

Окремо слід відзначити важливість розробки надійного механізму реєстрації та автентифікації користувачів. Це передбачає не лише перевірку логіна та пароля, а й ведення обліку невдалих спроб входу з автоматичним блокуванням доступу при перевищенні певної кількості помилок. Такий підхід гарантує підвищений рівень безпеки та запобігає спробам несанкціонованого доступу шляхом підбору паролів. Паралельно з цим, реалізація можливості відновлення та зміни налаштувань синхронізації й блокування користувачів сприяє гнучкому адмініструванню системи.

Функції роботи з документами мають бути реалізовані з урахуванням потреб військової частини щодо оформлення та обробки типових звітів, актів прийняття, передачі та вилучення майна. Модуль повинен підтримувати створення як текстових документів у форматі .docx, так і їхню конвертацію у PDF для подальшого збереження або друку. Важливо, щоб генерація документів базувалася на шаблонах, які враховують граматичні особливості мови, наприклад, правильне використання форм множини для назв майна, що підвищує якість і коректність офіційних документів. Вбудована функція автоматичного імпорту даних з Excel відкриває можливості для масового завантаження інформації про типи майна, що значно економить час при оновленні бази.

У контексті військового середовища особливої уваги заслуговує реалізація контролю категорій майна та типів, оскільки це сприяє правильній класифікації і швидкому пошуку необхідної інформації. Визначення та підтримка унікальних кодів для кожного одиничного екземпляра майна запобігає дублюванню і дозволяє

відстежувати рух активів з максимальною точністю. Розробка інтуїтивних засобів додавання, редагування та вилучення типів майна і користувачів з урахуванням їх рівнів доступу забезпечує гнучкість системи та контроль над змінами.

Особливо важливою є реалізація багаторівневого меню та динамічної системи прав доступу, що дозволяє відображати різні функціональні сторінки відповідно до ролей користувачів – від простих операторів до адміністраторів із повним набором можливостей. Така архітектурна особливість підвищує безпеку, мінімізує ризик випадкових помилок та сприяє логічній організації робочого процесу.

Тож лише за умови дотримання системного підходу та врахування всіх ключових аспектів – від структури баз даних і безпеки до зручності користування та відповідності офіційній документації — можна створити ефективний, надійний та зручний інструмент, що сприятиме оптимізації ресурсів і підвищенню оперативної здатності військової частини.

3.3 Тестування розробленого програмного модуля

Розпочати варто з входу в аккаунт користувача з повним рівнем доступу, на рисунку 3.1 зображено успішний вхід користувача в модуль.

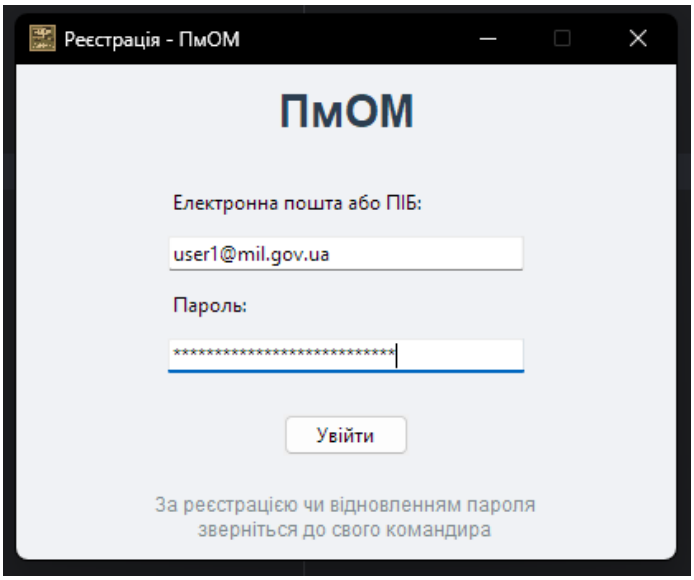

Рисунок 3.1 – Успішний вхід користувача

Після введення правильних даних користувача і натискання кнопки «Увійти» отримуємо, перехід на головну сторінку додатку з міні-профілем та функціональним меню, що зображена на рисунку 3.2.

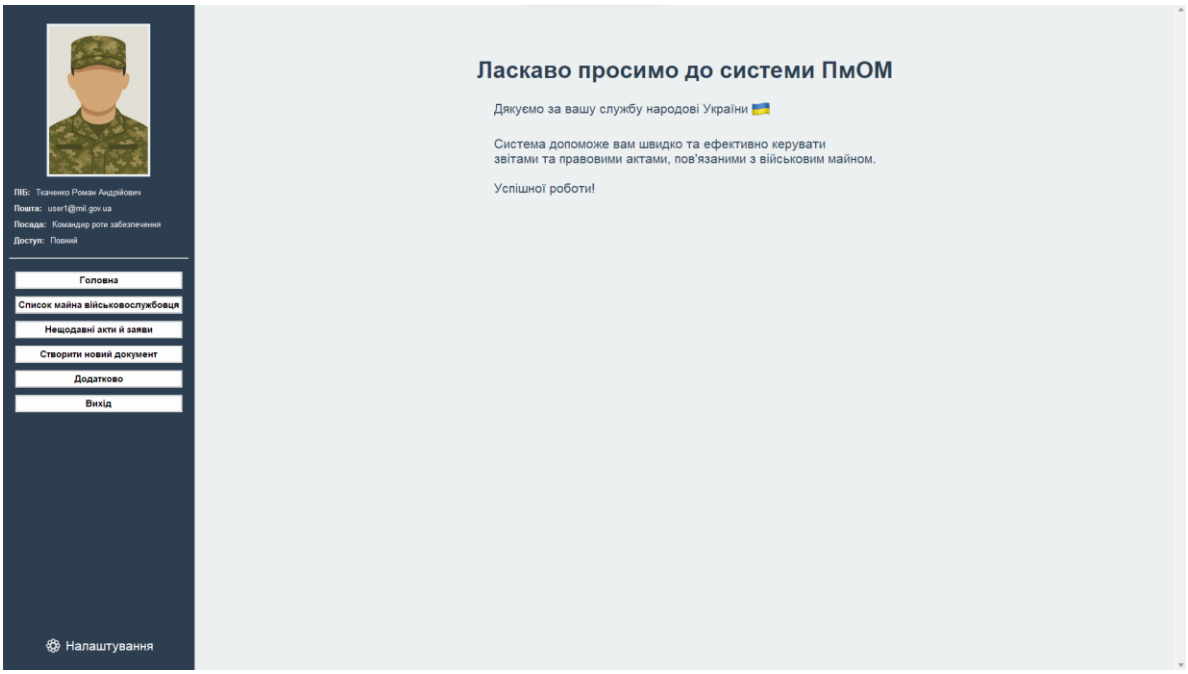

Рисунок 3.2 – Вигляд головної сторінки програмного модулю

Для початку перевіримо додаткові функції, які є унікальними для аккаунтів з повним доступом. Всі доступні функції зображені на рисунках 3.3 – 3.6.

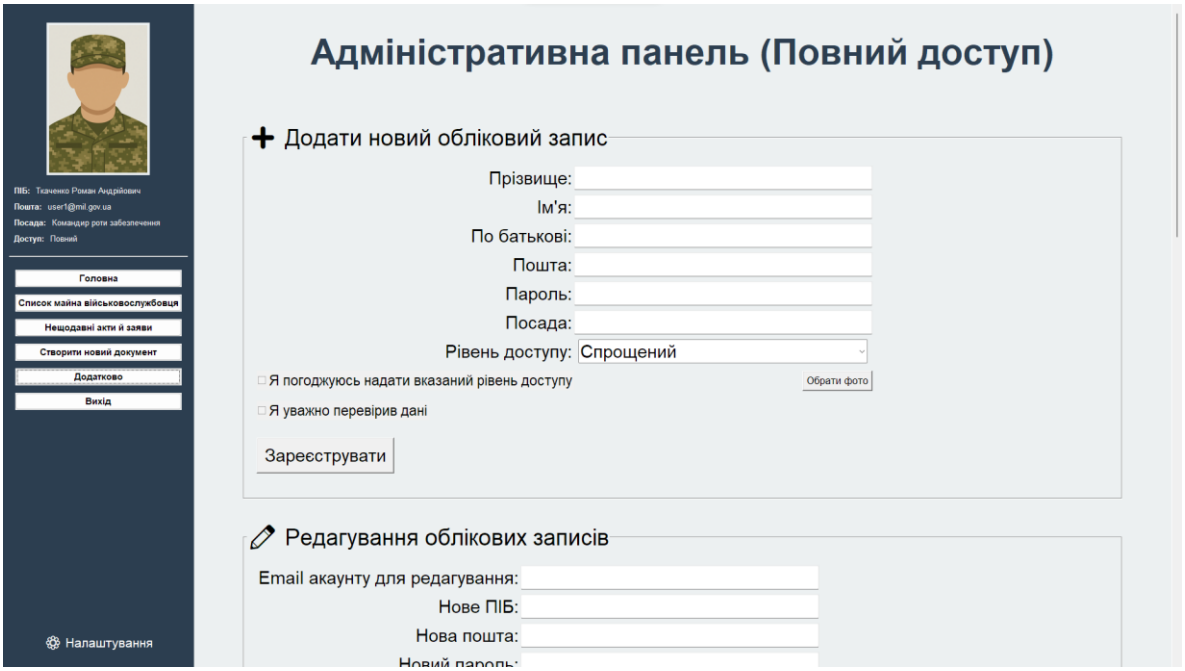

Рисунок 3.3 – Вкладка додатково й функція реєстрації зокрема

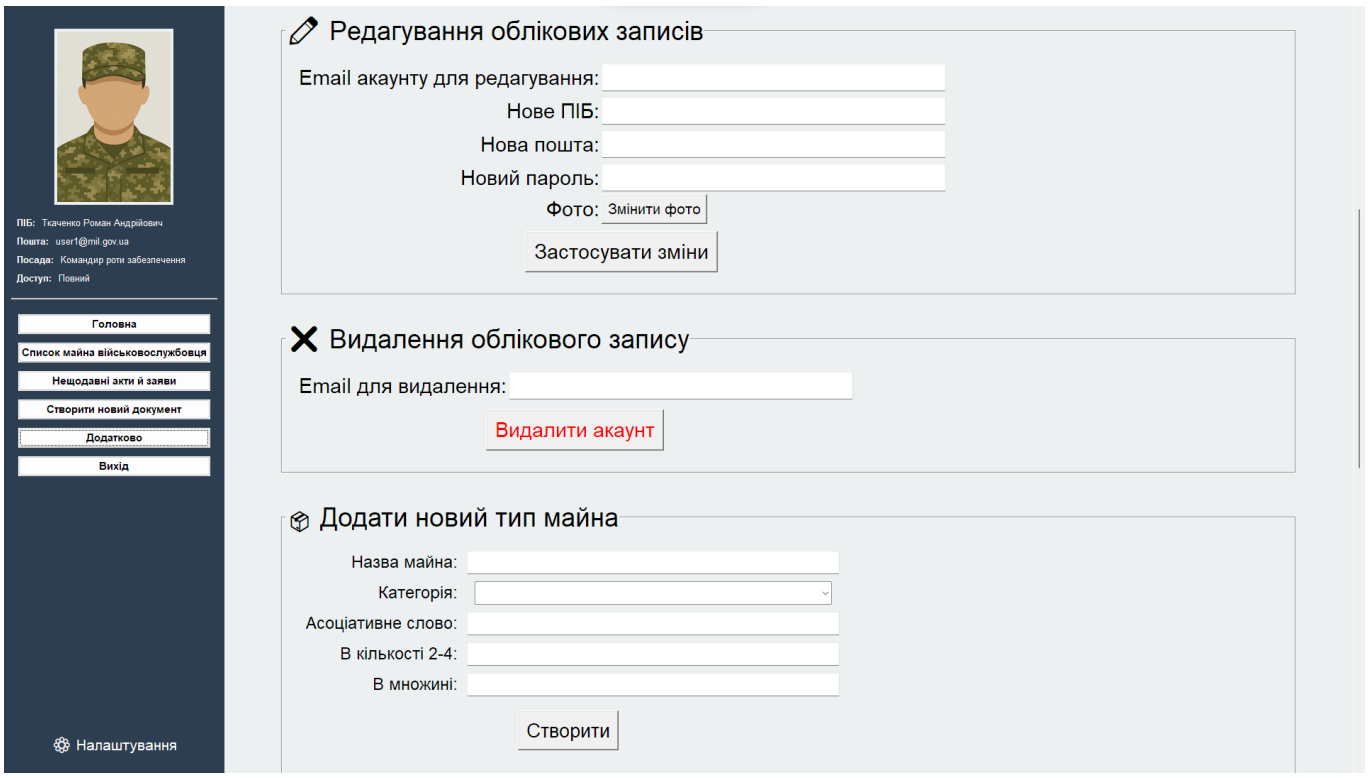

Рисунок 3.4 – Вкладка додатково й функції додавання нового типу майна, редагування та видалення облікових записів

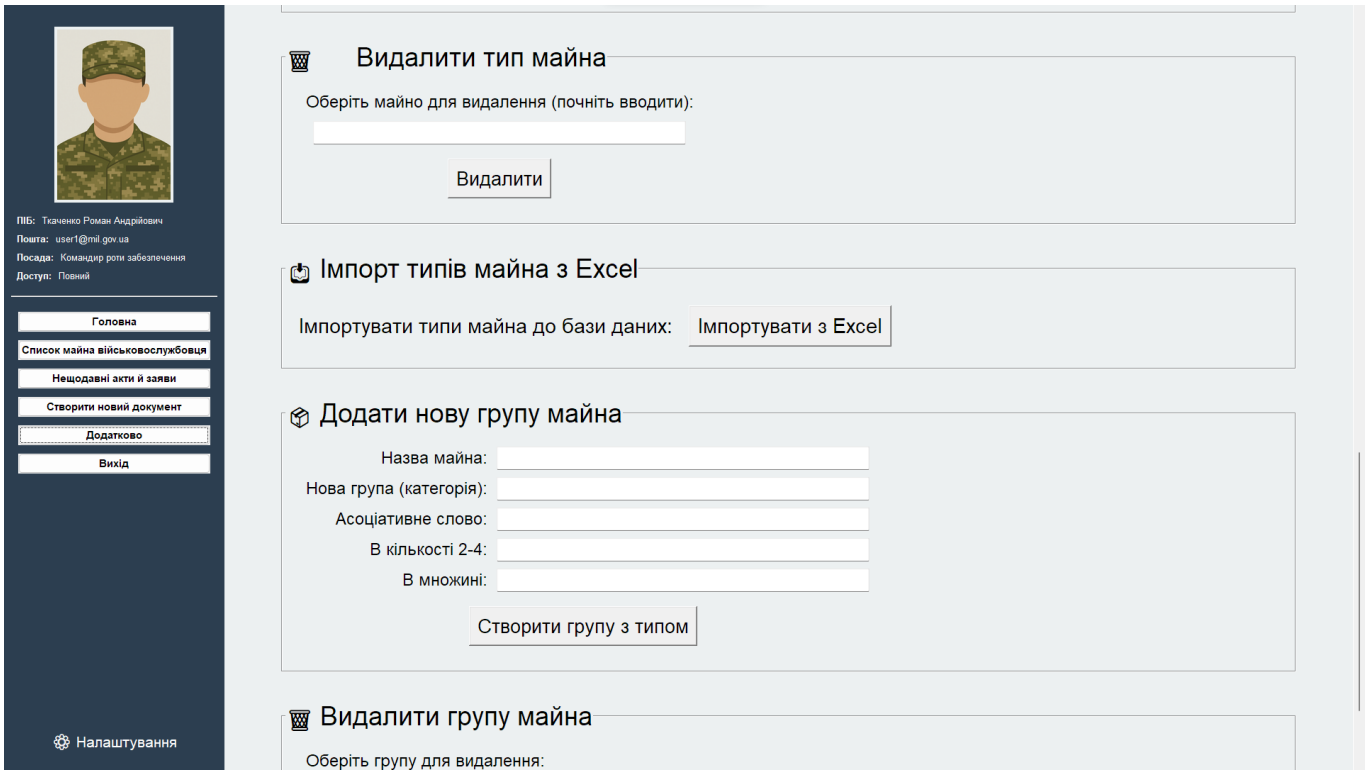

Рисунок 3.5 – Вкладка додатково й функції видалення типу майна, імпорту до бази даних та додавання нової групи майна

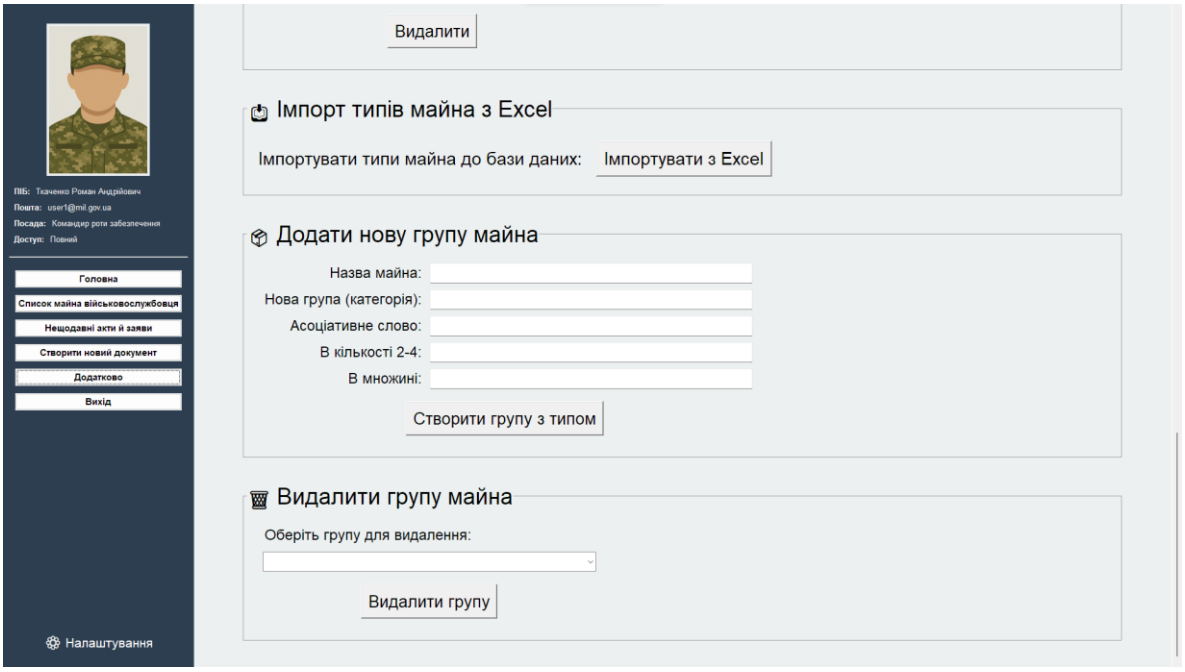

Рисунок 3.6 – Вкладка додатково й функція видалення групи майна

Тож давайте виконаємо кожну функцію й перевіримо їх спрацювання. Додамо користувача з ПІБ 1, почтою 1, й паролем 2, посадою Розробник, встановимо йому зображення профілю як адмін й надамо повний доступ. Відредагуємо профіль змінивши пароль на 1. Створимо анологічний профіль з цифрами 2 та 3. Профіль з ПІБ 3 видалимо. Додамо категорію майна Інструменти, та видалимо її. Додамо категорію майна Інструменти та обладнання. Додамо тип майна Пила, та видалимо його. Додамо тип майна Акумуляторна пила Stihl MSA 120 C-BQ. На рисунках 3.7-3.9 зображено спробу зайти на створені аккаунти. Що означає що всі три функції адміністрування аккаунтів працюють коректно.

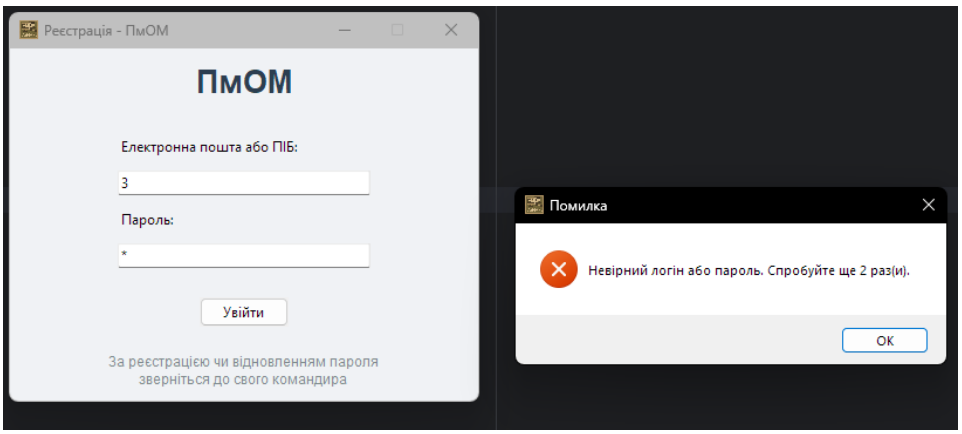

Рисунок 3.7 – Новостворений та видалений аккаунт

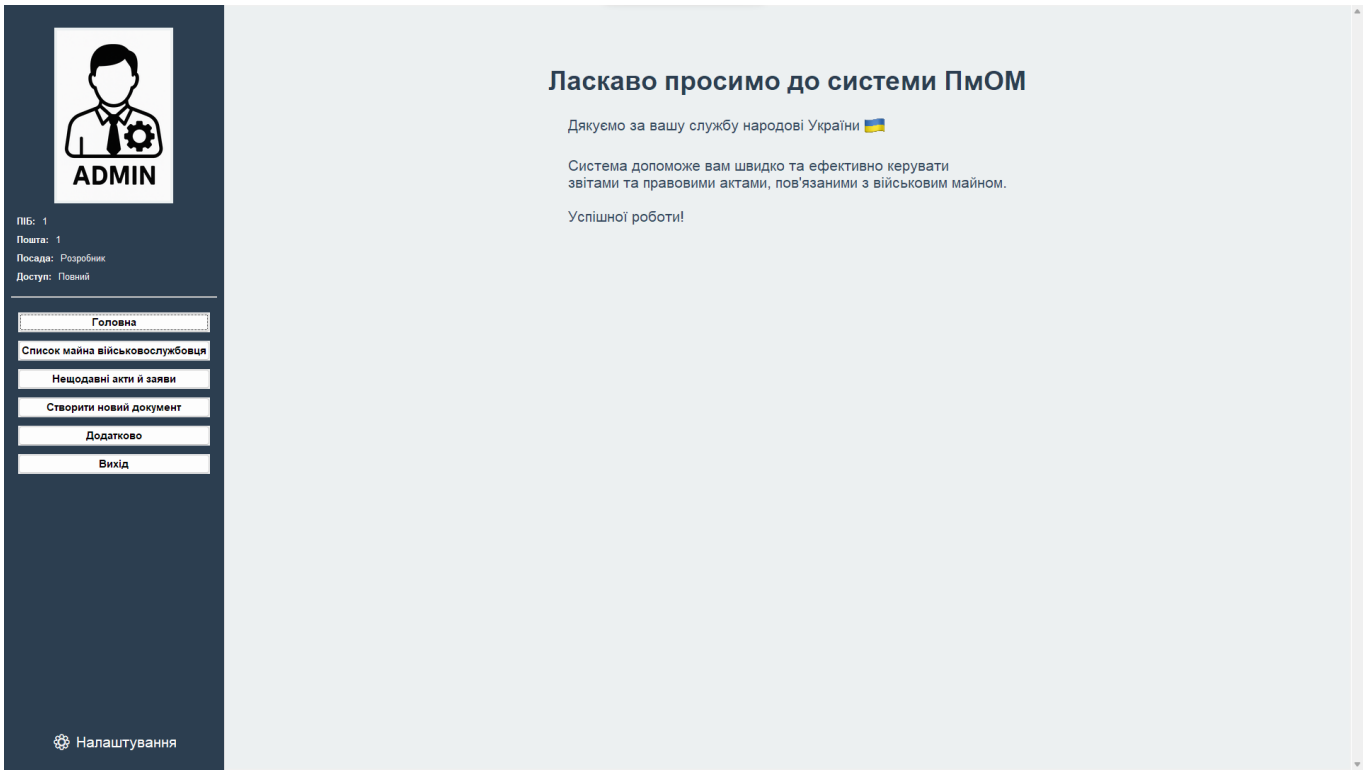

Рисунок 3.8 – Новостворений аккаунт з відредагованим паролем

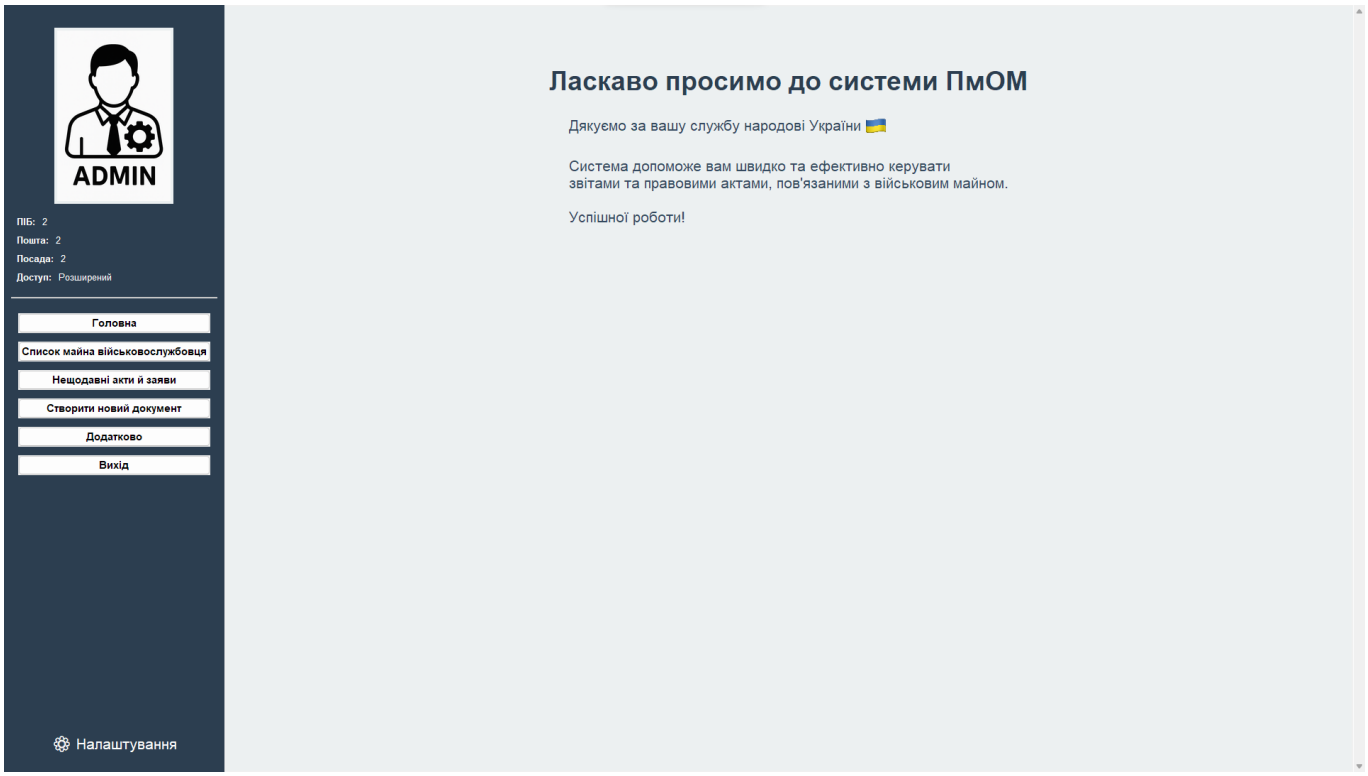

Рисунок 3.9 – Новостворений аккаунт з розширеними правами

У програмі також присутня система блокування входу при спробі перебору паролів, її роботу видно на рисунку 3.10.

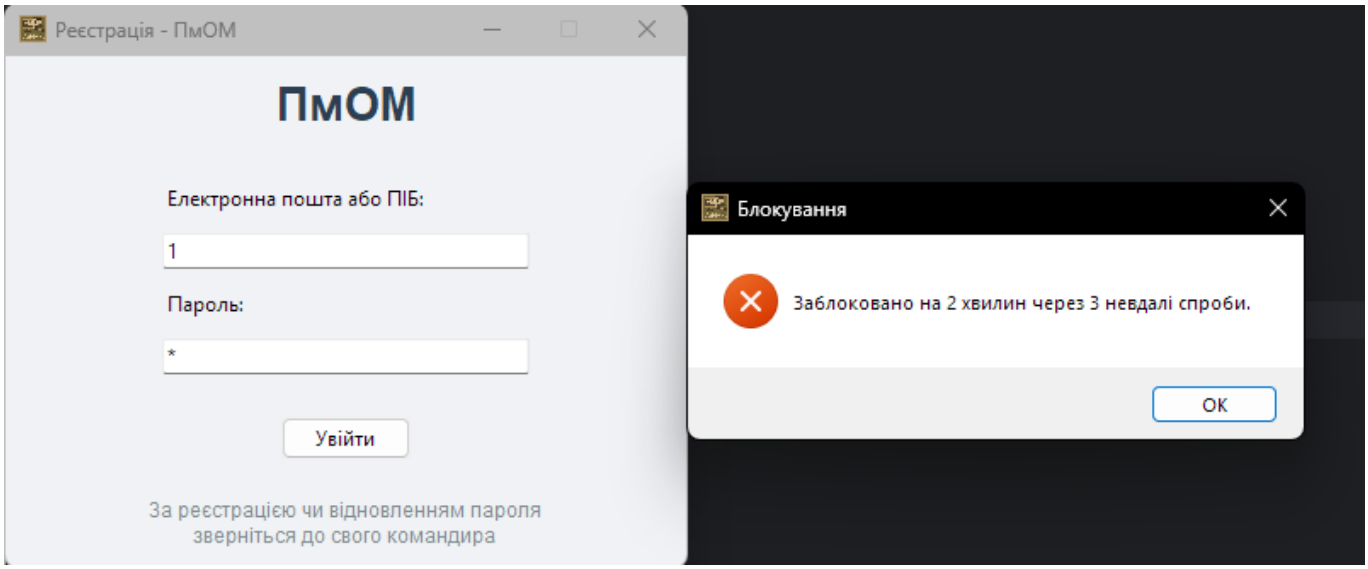

Рисунок 3.10 – Система блокування аккаунтів

У цілях тестування давайте нарешті автоматично створимо документ з новоствореною категорією та типом майна. На рисунку 3.11 зображено вікно для автоматичної генерації документації обліку майна.

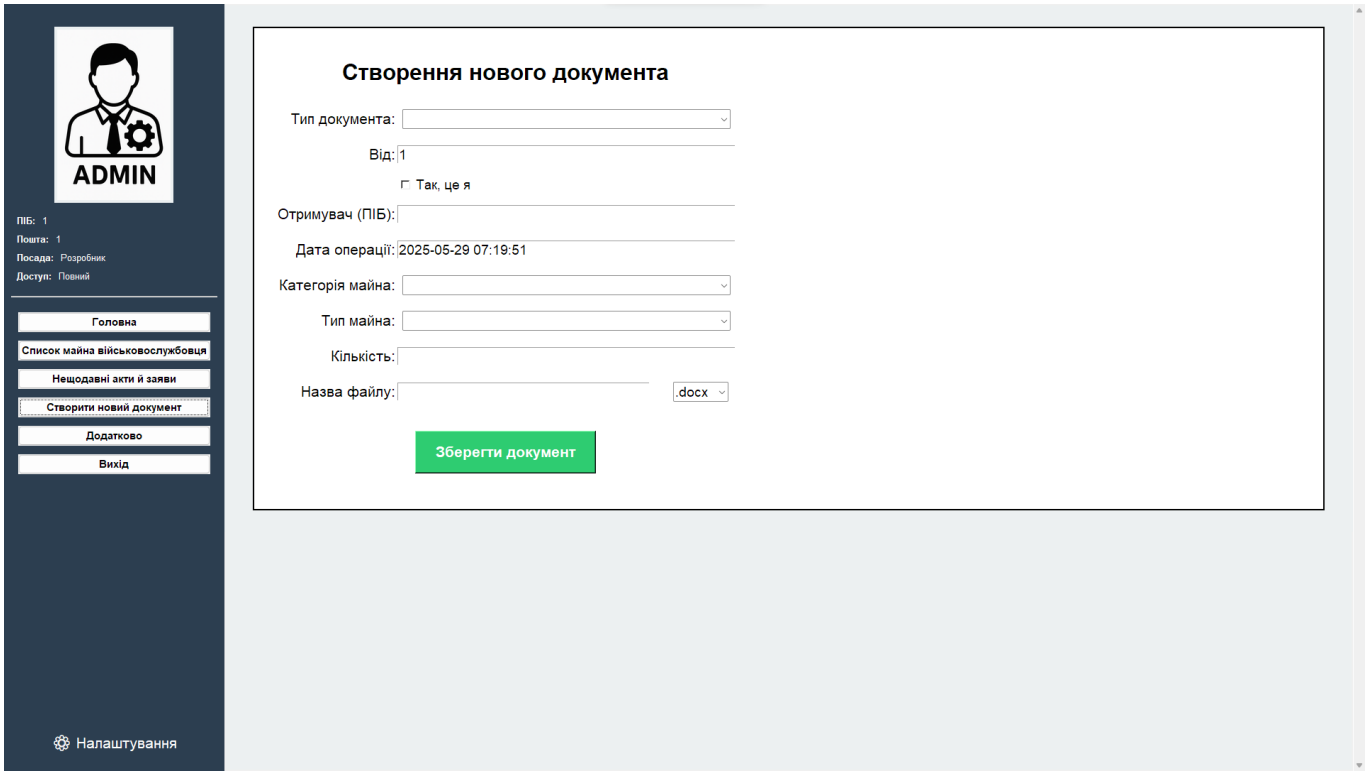

Рисунок 3.11 – Вікно автоматичної генерації документів

Отримаємо згенерований документ та переглянемо його з допомогою вкладки Нещодавні акти та заяви на рисунку 3.12 та у відповідному застосунку 3.13.

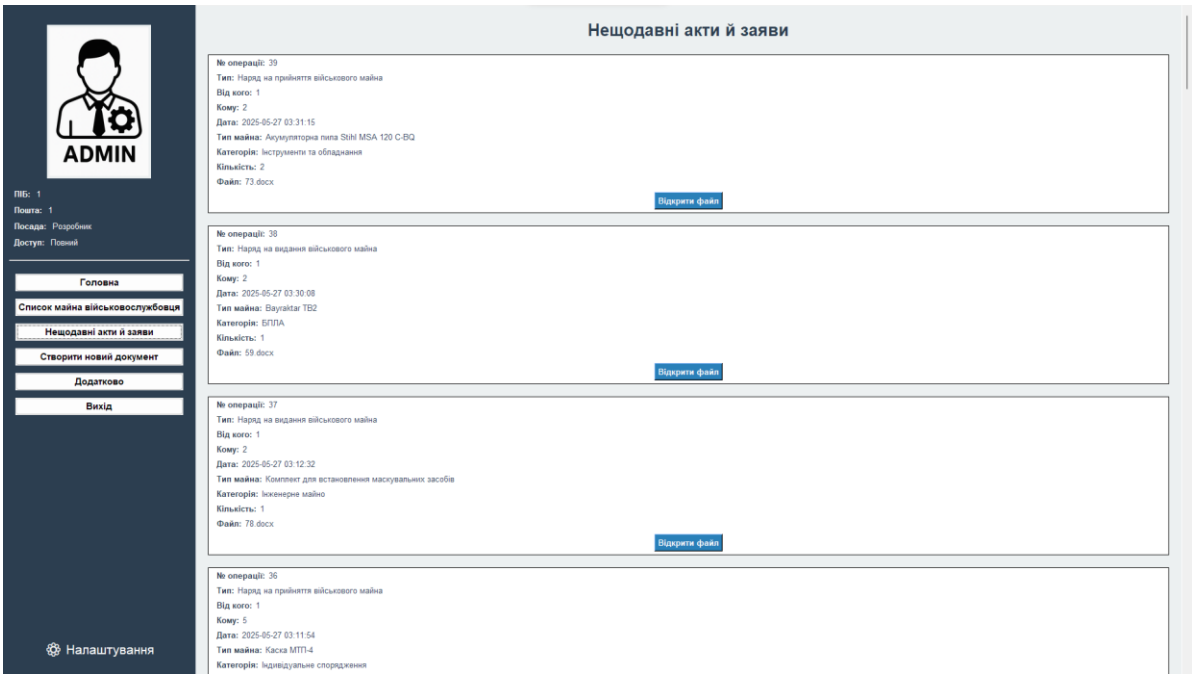

Рисунок 3.12 – Вікно перегляду автоматично згенерованих документів

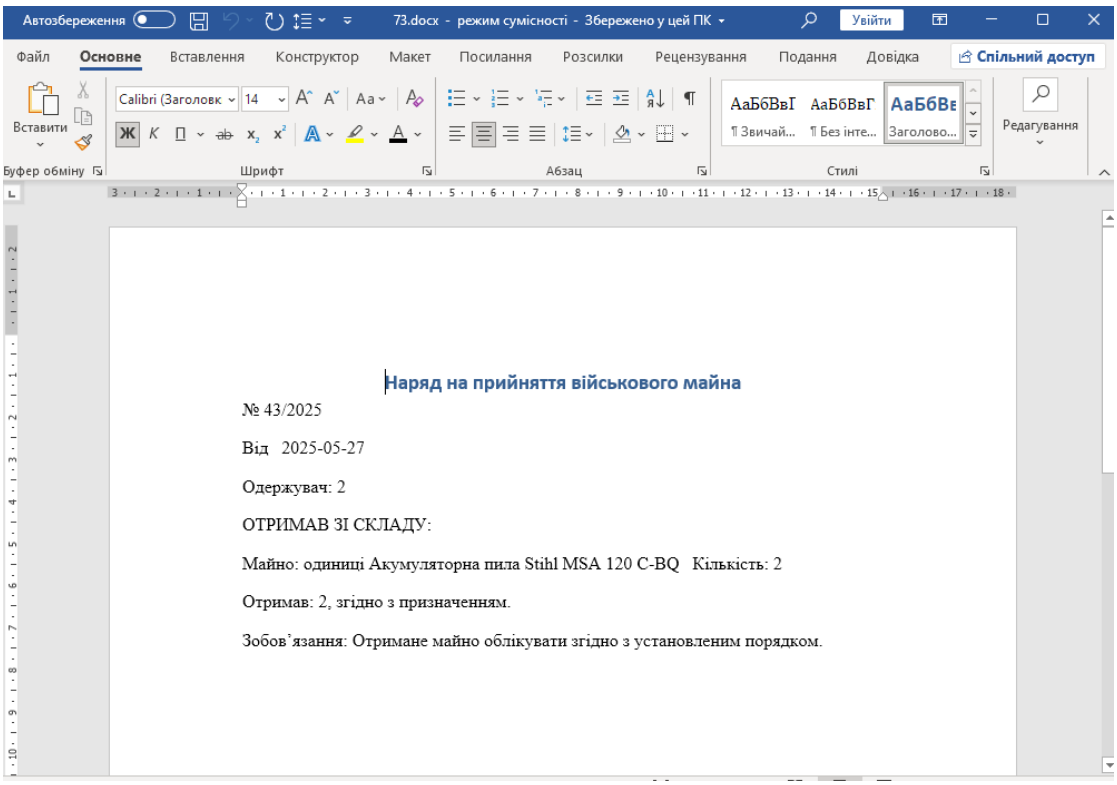

Рисунок 3.13 – Перегляд автоматично згенерованих документів засобами Microsoft Word

Також слід підмітити що система автоматично внесла в облік відповідне майно на баланс військовослужбовця, що й видно на рисунку 3.14.

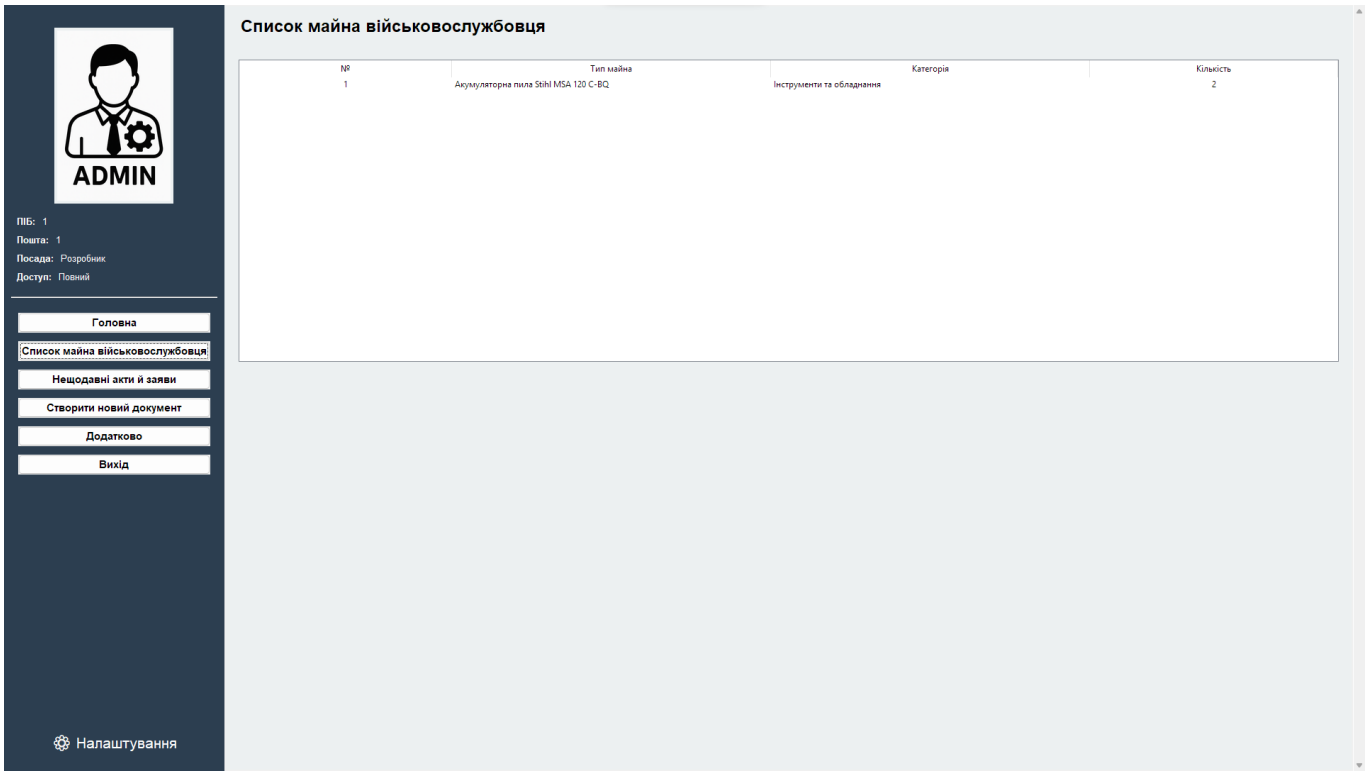

Рисунок 3.14 – Список майна військовослужбовця

За необхідності цю функцію легко можна відключити, що й продемонстровано на рисунку 3.15.

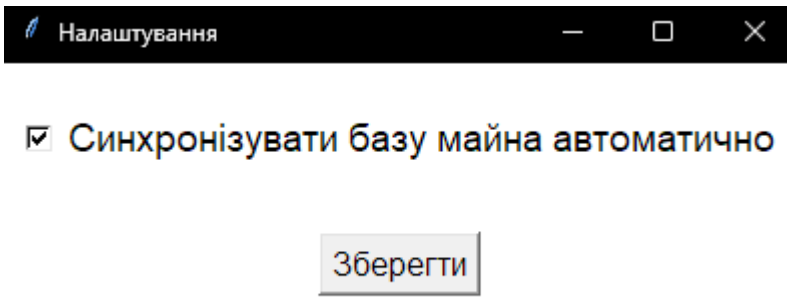

Рисунок 3.15 – Відключення синхронізації бази майна з нормативними документами згенерованими через систему

3.4 Висновок

У процесі реалізації програмного модуля для вибору мови програмування, середовища та бібліотек та практичну розробку, яка ґрунтувалась на поставленій задачі дослідження, розробленому алгоритмі та створених UML-діаграмах. Метою дослідження було розширення функціональних можливостей під час обліку майна, забезпечення функціональної, безпечної, гнучкої і зручної системи, що задовольняє специфічні потреби військової структури. Центральне місце в архітектурі системи зайняла мова Python, яка завдяки своїй простоті, читабельності, широкій бібліотечній екосистемі й здатності до швидкої реалізації складних логічних зв'язків дозволила сфокусувати зусилля на предметній області, а не на деталях реалізації. Обрані технології не були випадковими – кожне рішення підтверджене як технічними міркуваннями ефективності, так і практичними аспектами для експлуатації в умовах обмеженої інфраструктури та відповідних вимог безпеки.

Проведений аналіз середовищ програмування дозволив встановити доцільність використання PyCharm, як основного інструменту розробника. У межах даного проєкту ця IDE проявила себе як ефективний і зручний інструмент, що дозволяє реалізовувати складну логіку, здійснювати рефакторинг коду, швидко тестувати компоненти, відлагоджувати функції, та ефективно управляти проєктом. На відміну від інших популярних середовищ, зокрема Visual Studio Code, PyCharm забезпечує глибшу інтеграцію з Python-інструментарієм, розширену підтримку відлагодження та зручніші засоби рефакторингу, хоча й поступається за швидкодією запуску та споживанням системних ресурсів. Завдяки зручному інтерфейсу й розширеній підтримці Python-середовища, вибір саме PyCharm, зменшив час на розробку й сприяв підвищенню надійності створеного програмного продукту.

Для створений програмного модуля було обрано SQLite для роботи з реляційною базою даних на основі, яка забезпечила б достатній рівень продуктивності, простоту розгортання і підтримки, а також стійкість до помилок. SQLite вибрано оскільки засоби для комфортної взаємодії вже вбудовано в PyCharm,

відсутня залежность від окремого серверного забезпечення, що є критичним у військовому середовищі. Дані зберігаються локально, але повністю підтримують транзакційність, що гарантує цілісність облікових операцій навіть у разі нештатних ситуацій. Синергія SQLite і Python через модуль sqlite3 дозволила створити гнучку й водночас стійку до збоїв систему обліку.

Вибір бібліотек у проекті був виваженим і повністю відповідає як функціональним вимогам, так і контексту застосування. Зокрема, використання tkinter як основи графічного інтерфейсу дозволило створити інтуїтивно зрозуміле й швидке в роботі GUI-рішення, здатне працювати на різних платформах без встановлення додаткових компонентів. Простота використання та низький поріг входження у розробку з tkinter забезпечили гнучкість у реалізації інтерфейсних рішень, які були адаптовані до потреб цільових користувачів – військовослужбовців, які не завжди мають досвід користування складними інформаційними системами. Додаткове використання ttk дозволило реалізувати сучасні компоненти керування, що підвищують зручність користування й спрощують навчання персоналу роботі з програмою.

Бібліотека Pillow – для роботи з графічними зображеннями, підтвердила свою ефективність у контексті вимог проєкту. Потреба у швидкій обробці фотографій майна, користувацьких ідентифікаторів або документів була реалізована з мінімальними витратами системних ресурсів і без складної інтеграції з іншими модулями. Простий API Pillow забезпечив легку реалізацію збереження, конвертації та відображення зображень у вікнах графічного інтерфейсу. Цей підхід дозволив підвищити функціональність програми без ускладнення архітектури й збереженням стабільності роботи.

Результати реалізації свідчать про успішну розробку модуля обліку майна військової частини. Було реалізовано повний цикл обліку майна з можливістю фіксації, оновлення, перегляду та експорту інформації. Модуль автентифікації забезпечує контроль доступу на основі рівня доступу, що є критичним для військової структури. Застосування хешування паролів підвищує рівень захисту персональних

даних користувачів. Система передбачає гнучкий механізм генерації звітів, у тому числі з виводом у різних форматах, що спрощує подальшу роботу зі службовими документами. Проведене тестування розробленого модуля підтвердило його працездатність, стійкість до навантажень, коректність логіки обробки даних і зручність користування для цільової аудиторії.

Таким чином, результати роботи свідчать про високий рівень відповідності створеної програми заданим вимогам, а також про її готовність до впровадження в реальні умови функціонування військової частини. Завдяки обґрунтованим технічним рішенням і точному дотриманню вимог проєктування, було досягнуто балансу між функціональністю, зручністю, ефективністю та надійністю. У майбутньому система має потенціал до масштабування, адаптації під інші підрозділи або розширення функціоналу з урахуванням змін у нормативно-правовому полі чи структурі облікових процедур. Виконана робота є прикладом вдалого поєднання інженерного підходу, раціонального вибору інструментів та практичної реалізації інформаційної системи, що відповідає суворим критеріям галузевого застосування.

ВИСНОВКИ

Під час процесу розробки програмного модулю обліку майна військової частини поставлені перед дослідженням задачі було виконано в повному обсязі.

У межах дослідження здійснено огляд і ґрунтовний аналіз наявних аналогів програмного забезпечення для обліку майна у військових частинах. Було проведено порівняльну оцінку їхніх функціональних можливостей, виявлено основні переваги та недоліки, а також з'ясовано основні особливості використання таких систем у військових умовах. Огляд показав, що аналогам притаманний ряд недоліків: застосування застарілих методів протидії кібератакам, використання недостатньо зрозумілих інтерфейсів користувача, а також низький рівень автоматизації процесів обліку матеріальних засобів.

Здійснено детальну постановку задачі дослідження, яка полягає у розробці інтелектуального модуля обліку майна, що спрямований на автоматизацію процесів, мінімізацію людського фактору, підвищення продуктивності тилових підрозділів і забезпечення оперативного, безпомилкового контролю над матеріальними ресурсами в умовах Збройних Сил України. У межах задачі передбачено реалізацію таких вимог, як: безпека інформації та захист від несанкціонованого доступу; надійність і автономність роботи системи навіть у бойових умовах; гнучке керування рівнями доступу згідно з посадовими обов'язками; можливість масового оновлення даних; простий, інтуїтивно зрозумілий інтерфейс для користувачів із різним рівнем технічної підготовки; автоматичне формування супровідної документації згідно з чинними нормативами.

Розроблено загальний алгоритм функціонування програмного модулю для обліку майна, адаптований до умов специфіки військової діяльності та вимог автоматизації. Даний алгоритм відрізняється від існуючих наявністю додаткових блоків, які реалізують відповідні функції: імпорту/експорту загальної інформації, автоматична генерація документації у форматах DOCX та PDF, автоматичного оновлення довідників, а також автозаповнення у відповідних полях. Такий підхід не

лише забезпечує зручність користування, а й істотно скорочує кількість рутинної роботи, знижує ймовірність помилок та зменшує час, необхідний для ведення облікової діяльності та навчання персоналу.

Створено набір UML-діаграм: діаграми взаємодії з базами даних, компонентні діаграми та діаграми рівнів доступу, які дозволяють структурувати архітектуру модуля, моделювати зв'язки між базами даних, компонентами та ролями користувачів із правами доступу. Це забезпечує логічність збереження й обробки даних, ілюструє взаємодію модулів, контролює доступ, гарантує цілісність даних і підвищує безпеку через моделювання аутентифікації, авторизації та журналювання. Використання UML-діаграм позитивно вплинуло на якість, розуміння поставлених задач і надійність програмного модуля.

Обґрунтовано вибір мови програмування, середовища розробки та бібліотек, що дозволяють забезпечити гарну масштабованість програмного модулю. Використання SQLite – це раціональний компроміс між функціональністю, простотою впровадження і стабільністю. Саме тому обрані інструменти – мова програмування Python, середовище розробки PyCharm та бібліотеки tkinter, Pillow і sqlite3 – забезпечують не лише ефективність реалізації, а й високу масштабованість розробленого модуля. Їх застосування дозволяє адаптувати систему до змін вимог, додавати нові модулі (наприклад, підтримку інших форматів документів чи розширених звітів) без істотної перебудови архітектури. Обрані засоби відповідають критеріям продуктивності, сумісності, безпеки та простоти обслуговування, що є ключовими для стабільної роботи програмного забезпечення в умовах обмеженого доступу до інтернету, підвищених вимог до надійності й критичності збереження даних у військовій інфраструктурі.

Реалізовано програмний модуль обліку майна військової частини, який відрізняється від аналогічних рішень наявністю додаткових функцій: імпорту та експорту загальної інформації, автоматичної генерації документації у форматах DOCX і PDF, автоматичного оновлення довідників, а також функції автозаповнення у відповідних полях. Втілені в модулі технічні й концептуальні рішення дозволили

ефективно використати потенціал сучасних інформаційних технологій для вирішення задач обліку майна у військовому секторі.

Здійснено комплексне тестування розробленого програмного забезпечення. Отримані результати підтвердили відповідність функціоналу заявленим вимогам, продемонстрували стійкість системи до типових помилок та засвідчили доцільність обраних рішень для автоматизації процесу обліку майна.

У ході виконання дослідження було враховано нагальні виклики, пов’язані з активною фазою бойових дій та масштабним розширенням Збройних Сил України, що обумовило потребу у високоефективній системі обліку матеріальних ресурсів. Розроблений програмний модуль дозволяє забезпечити централізоване зберігання та обробку даних, мінімізувати вплив людського фактора, а також підвищити прозорість і контроль у процесах обліку майна військової частини.

Мета дослідження – розширення функціональних можливостей обліку майна – досягнута за рахунок того, що в порівнянні з найближчим аналогом введені додаткові функції, а саме: імпорт та експорт даних, автоматична генерація документації у форматах DOCX і PDF, автоматичне оновлення довідників, функція автозаповнення, гнучке керування рівнями доступу, журналювання дій користувачів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Прудніков Е. Р. Програмний модуль обліку майна для військової частини / Е. Р. Прудніков, І. Р. Арсенюк // Молодь в науці: дослідження, проблеми, перспективи (МН-2025), 15.10.24 – 14.06.25 : збірник матеріалів. – Вінниця: ВНТУ, 2025. – URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/viewFile/25533/21099
2. Forbes - Два роки війни у 18 графіках. Як повномасштабне вторгнення Росії змінило Україну – ЗСУ, держфінанси, демографія, міжнародна допомога і ставлення до перемоги. URL: https://forbes.ua/war-in-ukraine/yak-povnomasshtabne-vtorgnennya-rosii-vplinulo-na-ukrainu-dva-roki-viyni-v-grafikakh-23022024-19442
3. Українська Правда – За лаштунками цифровізації в Міноборони: топ-5 результатів за перші 100 днів URL: https://www.pravda.com.ua/columns/2023/12/22/7434130/
4. CSIS - How Ukraine Rebuilt Its Military Acquisition System Around Commercial Technology URL: https://www.csis.org/analysis/how-ukraine-rebuilt-its-military-acquisition-system-around-commercial-technology
5. Верховна Рада України – Про затвердження Інструкції з обліку військового майна у Збройних Силах України URL: https://zakon.rada.gov.ua/laws/show/z1192-17#Text
6. Верховна Рада України – Про затвердження Змін до деяких нормативно-правових актів Міністерства оборони України URL: https://zakon.rada.gov.ua/laws/show/z1257-24#Text
7. Вернер І. Є. та ін. Автоматизовані інформаційні системи органів управління військами. Київ, 2014.
8. Міністерство Оборони України – Міноборони спрощує облік майна і навчає військові частини. URL: https://mod.gov.ua/news/minoboroni-sproshhue-oblik-majna-i-navchae-vijskovi-chastini

9. RFID4u – The Role of RFID in Military Logistics URL: https://rfid4u.com/role-of-rfid-in-military-logistics/
10. U.S. Army – Predictive Logistics in Data-Driven Sustainment. URL: https://www.army.mil/article/270892/predictive_logistics_in_data_driven_sustainment
11. Побудова та розуміння алгоритмів. URL: https://cloud.itstep.org/blog_3/building-and-understanding-algorithms-a-step-by-step-guide-for-beginners
12. Кормен Т. Х., Лейзерсон Ч. Е., Рівест Р. Л., Штайн К. "Алгоритми: побудова та аналіз". – К.: Видавництво «Вільямс», 2003
13. ISO – Information security, cybersecurity and privacy protection — Information security management systems URL: https://www.iso.org/standard/27001
14. Object Management Group – Unified Modeling Language (UML) Specification, Version 2.5.1. URL: https://www.omg.org/spec/UML/2.5.1/
15. Pressman, R. S., & Maxim, B. R. (2014). *Software Engineering: A Practitioner's Approach* (8th ed.).
16. Sommerville, I. (2016). *Software Engineering* (10th ed.)
17. Python – About URL: https://www.python.org/about/
18. SQLite – About URL: https://www.sqlite.org/about.html
19. JetBrains - PyCharm URL: https://www.jetbrains.com/pycharm/
20. Python – tkinter — Python interface to Tcl/Tk URL: https://docs.python.org/3/library/tkinter.html
21. PyPi – Pillow about URL: https://pypi.org/project/pillow/
22. Python – hashlib — Secure hashes and message digests URL: https://docs.python.org/3/library/hashlib.html
23. Pandas – About us URL: https://pandas.pydata.org/about/
24. Python-docx – Documentation URL: https://python-docx.readthedocs.io/en/latest/
25. PyPi - docx2pdf URL: https://pypi.org/project/docx2pdf/
26. Python – subprocess — Subprocess management URL: https://docs.python.org/3/library/subprocess.html

27.Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп’ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. В. Сілагін, І. В. Богач. – Вінниця : ВНТУ, 2023. – (PDF, 54 с.).

ДОДАТКИ

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Програмний модуль обліку майна військової частини

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,46 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Арсенюк І.Р.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Прудніков Е.Р.
(підпис) (прізвище, ініціали)

ДОДАТОК Б
ІНСТРУКЦІЯ КОРИСТУВАЧА ПРОГРАМНОГО МОДУЛЯ ОБЛІКУ МАЙНА ВІЙСЬКОВОЇ ЧАСТИНИ

Для початку користувачу слід завантажити додаток ПмОМ (Програмний модуль обліку майна) на комп'ютер і отримати від командира логін та пароль для входу в систему. Щоб почати роботу, необхідно коректно ввести логін і пароль у вікні авторизації, яке зображено на рисунку Б.1.

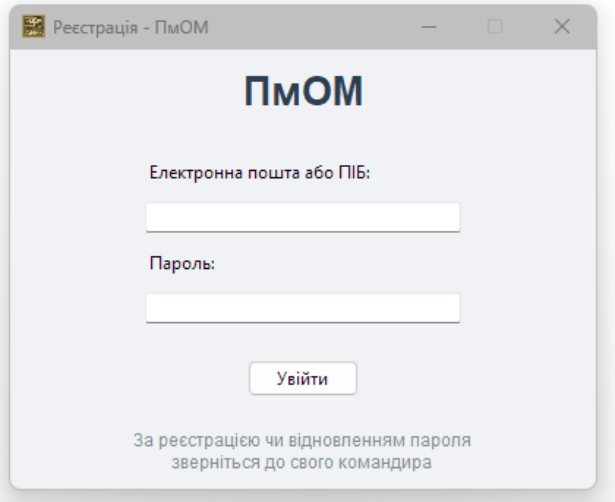

Рисунок Б.1 - Вікно авторизації користувача

У разі невдалої спроби введення даних, ви можете отримати блокування на певний період часу, як показано на рисунку Б.2.

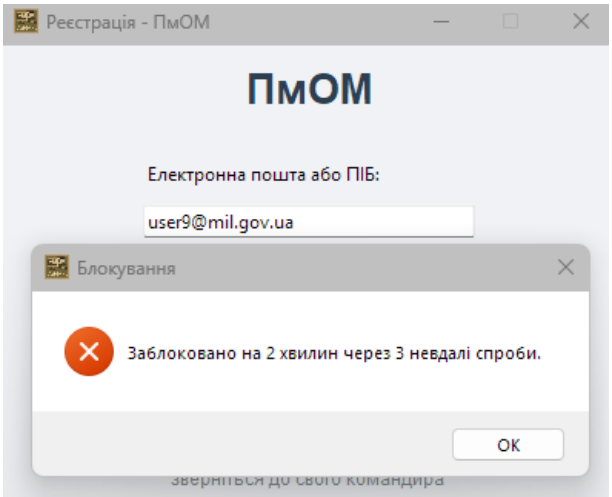

Рисунок Б.2 – Невдала спроба введення даних

Після успішного входу в систему ви потрапите на головну сторінку ПмОМ (див. рисунок Б.3). Вона, як і будь-яка інша, поділена на дві частини: мініпрофіль і меню з кнопками для переходу до інших розділів, а також праву оперативну частину, де й буде відбуватися більшість взаємодій із програмою.

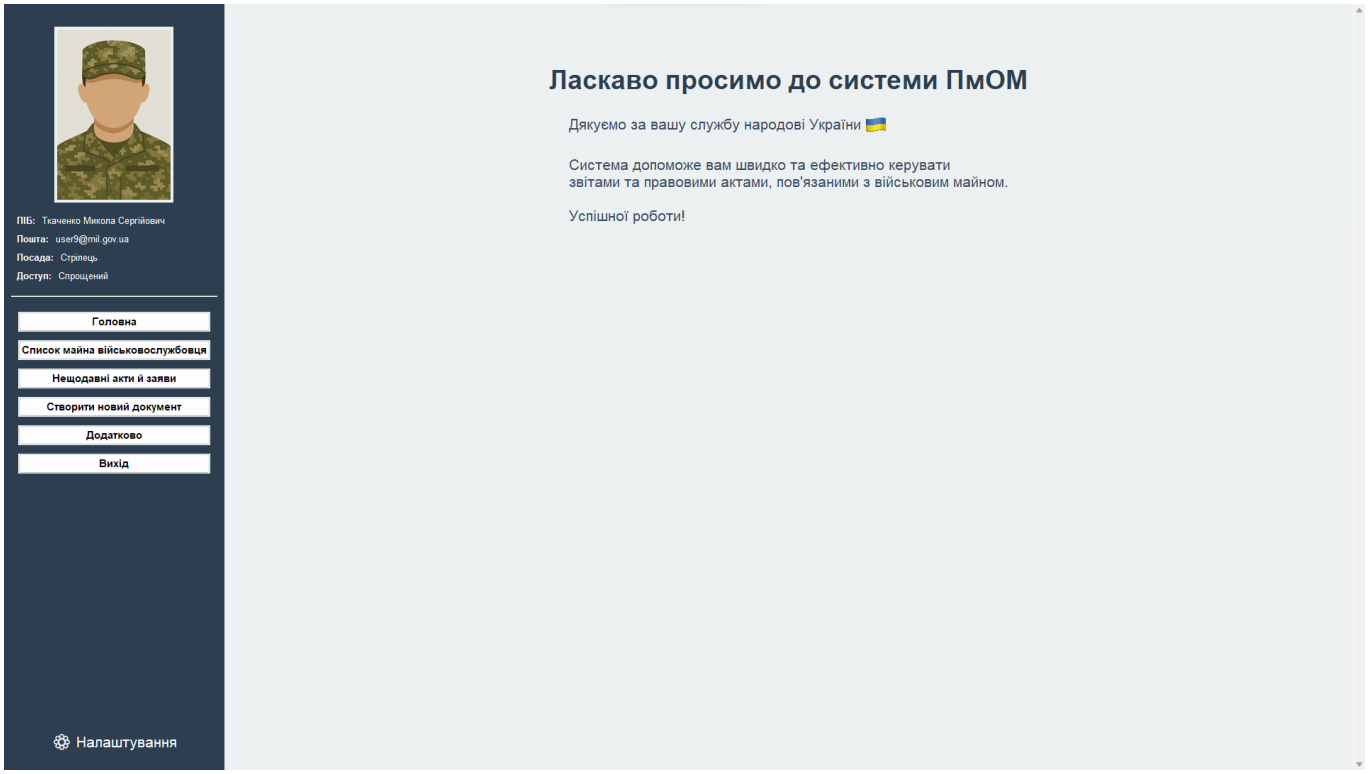

Рисунок Б.3 – Головна сторінка ПмОМ

Перший розділ – це список майна військовослужбовця. Тут у табличній формі чітко перелічено все майно, яке перебуває у вашій власності (див. рисунок 3.14). У наступному розділі можна переглядати нещодавно сформовані акти, створені за допомогою системи ПмОМ (див. рисунок 3.12).

Безпосередньо для створення актів і заяв слугує третій розділ. Весь процес створення акта проходить дуже швидко й полягає у заповненні відповідних полів, як у прикладі на рисунку 3.11.

Додаткові функції доступні лише користувачам з рівнями доступу "Розширений" та "Повний". Увесь їх перелік і вигляд зображено на рисунках 3.3–3.6.

Щоб вийти з системи, слід натиснути відповідну кнопку й підтвердити операцію (див. рисунок Б.4).

.

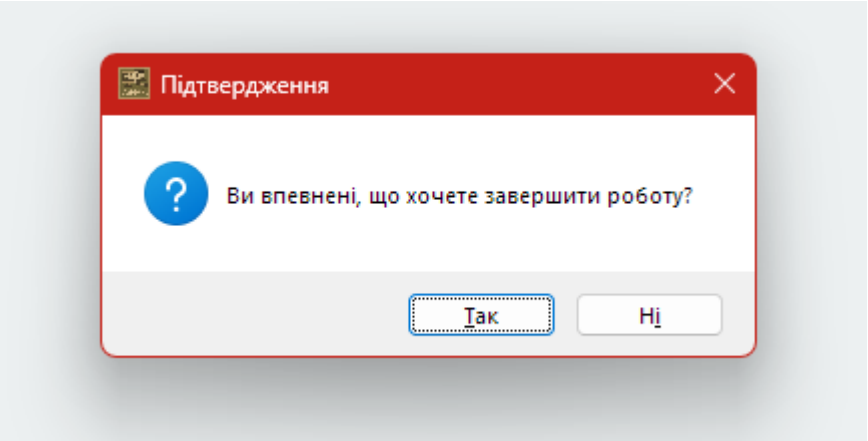

Рисунок Б.4 – Підтвердження операції виходу з ПмОМ

ДОДАТОК В

ЛІСТИНГ ПРОГРАМИ

```
import tkinter as tk
from tkinter import ttk, messagebox
from PIL import Image, ImageTk
import sys
import sqlite3
import hashlib
import os
from tkinter import filedialog
import pandas as pd
import time
import datetime
from docx import Document
from docx.shared import Pt
from docx.enum.text import WD_PARAGRAPH_ALIGNMENT
from docx2pdf import convert as docx2pdf_convert
import subprocess

def set_window_icon(window):
    try:
        icon_path = os.path.join("data", "img", "icon.png")
        icon_image = ImageTk.PhotoImage(Image.open(icon_path))
        window.iconphoto(False, icon_image)
        window._icon_ref = icon_image
    except Exception as e:
        print(f"Не вдалося встановити іконку: {e}")

DB_FILE = "data/users.db"
INVENTORY_DB = "data/inventory.db"
BLOCK_FILE = "data/block_status.txt"
SETTINGS_PATH = "data/settings.txt"
BLOCK_TIMES = [0, 2*60, 5*60, 15*60, 60*60, 2*60*60, 24*60*60]
AUTO_SYNC_ENABLED = True

def init_db():
    if not os.path.exists(DB_FILE):
        conn = sqlite3.connect(DB_FILE)
        cursor = conn.cursor()
        cursor.execute("""
        CREATE TABLE users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            full_name TEXT NOT NULL,
            email TEXT NOT NULL UNIQUE,
            password_hash TEXT NOT NULL,
            position TEXT,
            access_level TEXT,
            photo_filename TEXT
        )
        """)
        conn.commit()
        conn.close()

def save_settings_to_file():
```

```
    os.makedirs("data", exist_ok=True)
    with open(SETTINGS_PATH, "w") as f:
        f.write(f"AUTO_SYNC_ENABLED={AUTO_SYNC_ENABLED}\n")

def load_settings_from_file():
    global AUTO_SYNC_ENABLED
    try:
        with open(SETTINGS_PATH, "r") as f:
            for line in f:
                if line.startswith("AUTO_SYNC_ENABLED="):
                    value = line.strip().split("=")[1]
                    AUTO_SYNC_ENABLED = value == "True"
    except FileNotFoundError:
        AUTO_SYNC_ENABLED = True  # значення за замовчуванням

def setup_user_info_table_by_login(login):
    users_db_path = "data/users.db"
    helpinfo_db_path = "data/helpinfo.db"

    # Отримуємо id користувача
    conn = sqlite3.connect(users_db_path)
    cursor = conn.cursor()

    cursor.execute("""
        SELECT id FROM users
        WHERE full_name = ? OR email = ?
    """, (login, login))
    row = cursor.fetchone()
    conn.close()

    if not row:
        print("Користувача не знайдено.")
        return

    user_id = row[0]
    table_name = f"{int(user_id):05}info"

    # Створюємо папку, якщо потрібно
    if not os.path.exists("data"):
        os.makedirs("data")

    # Підключаємось до helpinfo.db
    conn = sqlite3.connect(helpinfo_db_path)
    cursor = conn.cursor()

    # Перевірка, чи існує таблиця
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name=?", (table_name,))
    result = cursor.fetchone()

    if not result:
        # Створення таблиці
        cursor.execute(f"""
            CREATE TABLE "{table_name}" (
                item_number INTEGER PRIMARY KEY AUTOINCREMENT,
                asset_type TEXT NOT NULL,
```

```
            asset_category TEXT NOT NULL,
            quantity INTEGER NOT NULL
        )
    """)
    conn.commit()
    print(f"Таблиця '{table_name}' створена.")
else:
    print(f"Таблиця '{table_name}' вже існує.")

conn.close()

# Глобальна змінна — назва таблиці для зручності
global USER_INFO_TABLE_NAME
USER_INFO_TABLE_NAME = table_name

# Глобальна змінна — шлях до бази
global HELPINFO_DB_PATH
HELPINFO_DB_PATH = helpinfo_db_path

def setup_user_operation_table_by_login(login):
    users_db_path = "data/users.db"
    operation_db_path = "data/operation.db"

    # Отримуємо id користувача
    conn = sqlite3.connect(users_db_path)
    cursor = conn.cursor()

    cursor.execute("""
        SELECT id FROM users
        WHERE full_name = ? OR email = ?
    """, (login, login))
    row = cursor.fetchone()
    conn.close()

    if not row:
        print("Користувача не знайдено.")
        return

    user_id = row[0]
    table_name = f"{int(user_id):05}operation"

    # Створюємо папку, якщо потрібно
    if not os.path.exists("data"):
        os.makedirs("data")

    # Підключаємось до operation.db
    conn = sqlite3.connect(operation_db_path)
    cursor = conn.cursor()

    # Перевірка, чи існує таблиця
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name=?", (table_name,))
    result = cursor.fetchone()

    if not result:
        # Створення таблиці
```

```
        cursor.execute(f"""
            CREATE TABLE "{table_name}" (
                operation_number INTEGER PRIMARY KEY AUTOINCREMENT,
                type_of_operation TEXT NOT NULL,
                from_user TEXT NOT NULL,
                to_user TEXT NOT NULL,
                date_of_operation TEXT NOT NULL,
                asset_type TEXT NOT NULL,
                asset_category TEXT NOT NULL,
                quantity INTEGER NOT NULL,
                file_name TEXT NOT NULL
            )
        """)
        conn.commit()
        print(f"Таблиця '{table_name}' створена.")
    else:
        print(f"Таблиця '{table_name}' вже існує.")

    conn.close()

    # Глобальна змінна — назва таблиці для зручності
    global OPERATION_TABLE_NAME
    OPERATION_TABLE_NAME = table_name

    # Глобальна змінна — шлях до бази
    global OPERATION
    OPERATION = operation_db_path

def show_user_asset_list(login, parent_frame, canvas):
    # Очистка попереднього вмісту
    for widget in parent_frame.winfo_children():
        widget.destroy()

    users_db_path = "data/users.db"
    helpinfo_db_path = "data/helpinfo.db"

    conn = sqlite3.connect(users_db_path)
    cursor = conn.cursor()
    cursor.execute("""
        SELECT id FROM users
        WHERE full_name = ? OR email = ?
    """, (login, login))
    row = cursor.fetchone()
    conn.close()

    if not row:
        tk.Label(parent  frame, text="Користувача не знайдено.", font=("Arial", 14)).pack(pady=10)
        return

    user_id = row[0]
    table_name = f"{int(user_id):05}info"

    conn = sqlite3.connect(helpinfo_db_path)
    cursor = conn.cursor()
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name=?", (table_name,))
```

```
    result = cursor.fetchone()

    if not result:
        tk.Label(parent_frame, text=f"Таблиця '{table_name}' не знайдена.", font=("Arial", 14)).pack(pady=10)
        conn.close()
        return

    cursor.execute(f"SELECT item_number, asset_type, asset_category, quantity FROM \"{table_name}\"")
    assets = cursor.fetchall()
    conn.close()

    if not assets:
        tk.Label(parent_frame, text="Список майна порожній.", font=("Arial", 14)).pack(pady=10)
        return

    # === Заголовок ===
    title = tk.Label(parent_frame, text="Список майна військовослужбовця", font=("Arial", 18, "bold"),
bg="#ecf0f1", anchor="w")
    title.pack(pady=(10, 20), padx=20, anchor="w")

    # === Таблиця з майном ===
    columns = ("item_number", "asset_type", "asset_category", "quantity")
    tree = ttk.Treeview(parent_frame, columns=columns, show="headings", height=20)

    tree.heading("item_number", text="№")
    tree.heading("asset_type", text="Тип майна")
    tree.heading("asset_category", text="Категорія")
    tree.heading("quantity", text="Кількість")

    tree.column("item_number", width=50, anchor="center")
    tree.column("asset_type", width=200)
    tree.column("asset_category", width=200)
    tree.column("quantity", width=100, anchor="center")

    for item in assets:
        tree.insert("", "end", values=item)

    tree.pack(padx=20, pady=10, fill="x")

    # Оновити прокрутку
    canvas.update_idletasks()
    canvas.configure(scrollregion=canvas.bbox("all"))

    canvas.yview_moveto(0)

def show_recent_operations(content_frame, canvas, user_full_name):
    # Очистити поточний вміст
    for widget in content_frame.winfo_children():
        widget.destroy()

    # Підключення до бази operation
    conn = sqlite3.connect(OPERATION)
    cursor = conn.cursor()

    # Запит до таблиці користувача
```

```
    try:
        cursor.execute(f"SELECT * FROM '{OPERATION_TABLE_NAME}' ORDER BY date_of_operation
DESC")
        operations = cursor.fetchall()
    except sqlite3.Error as e:
        tk.Label(content_frame, text=f"Помилка бази даних: {e}", fg="red", font=("Arial", 12)).pack(pady=20)
        conn.close()
        return

    conn.close()

    if not operations:
        tk.Label(content_frame, text="Немає жодного акту або заяви.", font=("Arial", 14), fg="#7f8c8d",
bg="#ecf0f1").pack(pady=20)
        return

    # Заголовок
    tk.Label(content_frame, text="Нещодавні акти й заяви", font=("Arial", 20, "bold"), bg="#ecf0f1",
fg="#2c3e50").pack(pady=(20, 10))

    for op in operations:
        frame = tk.Frame(content_frame, bg="#ffffff", bd=1, relief="solid")
        frame.pack(padx=20, pady=10, fill="x")

        fields = {
            "№ операції": op[0],
            "Тип": op[1],
            "Від кого": op[2],
            "Кому": op[3],
            "Дата": op[4],
            "Тип майна": op[5],
            "Категорія": op[6],
            "Кількість": op[7],
            "Файл": op[8]
        }

        for label, value in fields.items():
            inner = tk.Frame(frame, bg="#ffffff")
            inner.pack(anchor="w", padx=10, pady=1)
            tk.Label(inner, text=f"{label}:", font=("Arial", 10, "bold"), bg="#ffffff", fg="#2c3e50").pack(side="left")
            tk.Label(inner, text=value, font=("Arial", 10), bg="#ffffff", fg="#34495e").pack(side="left")

        # Кнопка відкриття файлу
        def open_file(file=op[8]):
            file_path = os.path.join("documents", file)
            if os.path.exists(file_path):
                try:
                    if os.name == "nt":  # Windows
                        os.startfile(file_path)
                    elif os.name == "posix":  # macOS / Linux
                        subprocess.call(["xdg-open", file_path])
                except Exception as e:
                    tk.messagebox.showerror("Помилка", f"Не вдалося відкрити файл:\n{e}")
            else:
                tk.messagebox.showwarning("Файл не знайдено", f"Файл не знайдено:\n{file_path}")
```

```
        tk.Button(frame, text="Відкрити файл", font=("Arial", 10, "bold"), command=open_file, bg="#2980b9",
fg="white").pack(pady=5)

        canvas.yview_moveto(0)

def hash_password(password):
    return hashlib.sha256(password.encode('utf-8')).hexdigest()

def load_block_status():
    if not os.path.exists(BLOCK_FILE):
        return {"fail_count": 0, "block_start": 0, "block_level": 0}
    with open(BLOCK_FILE, "r") as f:
        data = f.read().strip().split(",")
        if len(data) != 3:
            return {"fail_count": 0, "block_start": 0, "block_level": 0}
        return {
            "fail_count": int(data[0]),
            "block_start": float(data[1]),
            "block_level": int(data[2])
        }

def save_block_status(fail_count, block_start, block_level):
    with open(BLOCK_FILE, "w") as f:
        f.write(f"{fail_count},{block_start},{block_level}")

class AutocompleteEntry(ttk.Entry):
    def __init__(self, suggestions, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.suggestions = sorted(suggestions)
        self.var = self["textvariable"] = tk.StringVar()
        self.var.trace("w", self.on_change)
        self.lb = None

    def on_change(self, *args):
        value = self.var.get().lower()
        if value == "":
            self.hide_listbox()
            return
        matching = [s for s in self.suggestions if value in s.lower()]
        if matching:
            if self.lb is None:
                self.lb = tk.Listbox(self.master, height=3, width=40, font=self["font"])
                self.lb.bind("<<ListboxSelect>>", self.on_select)
                self.lb.place(x=self.winfo_x() - 23, y=self.winfo_y() + self.winfo_height() - 65)
            self.lb.delete(0, tk.END)
            for item in matching:
                self.lb.insert(tk.END, item)
        else:
            self.hide_listbox()

    def on_select(self, event):
        if self.lb:
            selection = self.lb.get(self.lb.curselection())
            self.var.set(selection)
```

```
            self.hide_listbox()

    def hide_listbox(self):
        if self.lb:
            self.lb.destroy()
            self.lb = None

def register_user(full_name, email, password, position, access_level, photo_filename):
    conn = sqlite3.connect(DB_FILE)
    cursor = conn.cursor()
    try:
        cursor.execute("""
            INSERT INTO users (full_name, email, password_hash, position, access_level, photo_filename)
            VALUES (?, ?, ?, ?, ?, ?)
        """, (full_name, email, hash_password(password), position, access_level, photo_filename))
        conn.commit()

        return True, "Реєстрація успішна!"
    except sqlite3.IntegrityError:
        return False, "Користувач із такою поштою вже існує."
    finally:
        conn.close()

def authenticate_user(login, password):
    conn = sqlite3.connect(DB_FILE)
    cursor = conn.cursor()
    password_hash = hash_password(password)
    # Пошук по email або full_name
    cursor.execute("""
        SELECT full_name, email, position, access_level, photo_filename FROM users
        WHERE (email = ? OR full_name = ?) AND password_hash = ?
    """, (login, login, password_hash))
    user = cursor.fetchone()
    conn.close()
    return user

def handle_additional_menu(user, content_frame, canvas):
    access_level = user[3]

    for widget in content_frame.winfo_children():
        widget.destroy()

    if access_level == "Спрощений":
        show_main_content(content_frame, canvas)
        messagebox.showerror("Недостатньо прав", "У вашого облікового запису недостатньо прав для
перегляду додаткових функцій.")
    elif access_level == "Розширений":
        show_additional_page_for_extended(content_frame, canvas)
    elif access_level == "Повний":
        show_additional_page_for_full(content_frame, canvas)
    else:
        messagebox.showerror("Помилка", "Невідомий рівень доступу.")

def import_inventory_from_excel():
    excel_path = filedialog.askopenfilename(title="Оберіть Excel-файл", filetypes=[("Excel файли", "*.xlsx")])
```

```
    if not excel_path:
        return

    try:
        df = pd.read_excel(excel_path, engine='openpyxl', header=1, usecols="B:G")
        df.dropna(how="all", inplace=True)
        df.columns = ["code", "name", "category", "keyword", "plural_2_4", "plural"]

        conn = sqlite3.connect("data/inventory.db")
        cursor = conn.cursor()
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS inventory (
                code TEXT PRIMARY KEY,
                name TEXT,
                category TEXT,
                keyword TEXT,
                plural_2_4 TEXT,
                plural TEXT
            )
        ''')

        for _, row in df.iterrows():
            cursor.execute('''
                INSERT OR REPLACE INTO inventory (code, name, category, keyword, plural_2_4, plural)
                VALUES (?, ?, ?, ?, ?, ?)
            ''', (
                row["code"],
                row["name"],
                row["category"],
                row["keyword"],
                row["plural_2_4"],
                row["plural"]
            ))

        conn.commit()
        conn.close()

        messagebox.showinfo("Успіх", "Типи майна імпортовано з Excel у базу даних.")
    except Exception as e:
        messagebox.showerror("Помилка", f"Помилка при імпорті:\n{e}")

def get_categories():
    """Повертає список унікальних категорій з БД."""
    conn = sqlite3.connect(INVENTORY_DB)
    cursor = conn.cursor()
    cursor.execute("SELECT DISTINCT category FROM inventory ORDER BY category")
    categories = [row[0] for row in cursor.fetchall()]
    conn.close()
    return categories

def get_asset_types_by_category(category):
    if not category:
        tk.messagebox.showerror("Помилка", "Оберіть категорію майна перед вибором типу.")
        return []
    conn = sqlite3.connect(INVENTORY_DB)
```

```
    cursor = conn.cursor()
    cursor.execute("SELECT DISTINCT name FROM inventory WHERE category = ? ORDER BY name",
(category,))
    asset_types = [row[0] for row in cursor.fetchall()]
    conn.close()
    return asset_types

def get_next_code():
    """Повертає код для нового майна, на 1 більше останнього за номером."""
    conn = sqlite3.connect(INVENTORY_DB)
    cursor = conn.cursor()
    cursor.execute("SELECT code FROM inventory ORDER BY code DESC LIMIT 1")
    row = cursor.fetchone()
    conn.close()
    if row:
        last  code = row[0]  # Приклад: "AB0002"
        prefix = ''.join(filter(str.isalpha, last_code))
        number = int(''.join(filter(str.isdigit, last_code)))
        new_number = number + 1
        return f"{prefix}{new_number:04d}"
    else:
        return "AB0001"  # Якщо БД порожня

def add_inventory_type(name, category, keyword, plural_2_4, plural):
    conn = sqlite3.connect(INVENTORY_DB)
    cursor = conn.cursor()
    code = get_next_code()
    try:
        cursor.execute("""
            INSERT INTO inventory (code, name, category, keyword, plural_2_4, plural)
            VALUES (?, ?, ?, ?, ?, ?)
        """, (code, name, category, keyword, plural_2_4, plural))
        conn.commit()
    except sqlite3.IntegrityError as e:
        messagebox.showerror("Помилка", f"Не вдалося додати майно: {e}")
        conn.close()
        return False
    conn.close()
    return True

def delete_inventory_type_by_name(name):
    conn = sqlite3.connect(INVENTORY_DB)
    cursor = conn.cursor()
    cursor.execute("DELETE FROM inventory WHERE name = ?", (name,))
    deleted = cursor.rowcount
    conn.commit()
    conn.close()
    return deleted > 0

def get_plural_form(quantity):

    quantity = int(quantity)
    last_two = quantity % 100
    last_one = quantity % 10
```

```
    if 11 <= last_two <= 14:
        return 'plural'        # 11-14: п’ять і більше, форма 3
    elif last_one == 1:
        return 'keyword'       # закінчується на 1 (окрім 11)
    elif last_one in [2, 3, 4]:
        return 'plural_2_4'    # закінчується на 2, 3, 4 (окрім 12-14)
    else:
        return 'plural'

def generate_document(data, save_path, extension):
    doc = Document()

    # Встановлення стилю документа
    style = doc.styles['Normal']
    font = style.font
    font.name = 'Times New Roman'
    font.size = Pt(12)

    # Заголовок
    title = doc.add_heading(data['type_of_operation'], level=1)
    title.alignment = WD_PARAGRAPH_ALIGNMENT.CENTER

    conn = sqlite3.connect('data/inventory.db')
    cursor = conn.cursor()

    # Визначаємо потрібну колонку для запиту в залежності від quantity
    plural_form = get_plural_form(data['quantity'])

    query_map = {
        'keyword': "SELECT keyword FROM inventory WHERE name = ?",
        'plural_2_4': "SELECT plural_2_4 FROM inventory WHERE name = ?",
        'plural': "SELECT plural FROM inventory WHERE name = ?"
    }

    cursor.execute(query_map[plural_form], (data['asset_type'],))
    result = cursor.fetchone()
    # result повертається у вигляді кортежу, беремо перший елемент
    word = result[0] if result else data['asset_type']

    # Основний вміст
    if data['type_of_operation'] == "Наряд на видання військового майна":
        doc.add_paragraph(f"№ 47/2025")
        doc.add_paragraph(f"Від  {data['date_of_operation'][:10]}")
        doc.add_paragraph(f"Одержувач: {data['to_user']}")
        doc.add_paragraph(f"ВИДАТИ ЗІ СКЛАДУ:")
        doc.add_paragraph(f"Майно: {word} {data['asset_type']}  Кількість: {data['quantity']}")
        doc.add_paragraph(f"Видати: {data['to_user']}, згідно з призначенням.")
        doc.add_paragraph(f"Зобов’язання: Прийняте майно облікувати згідно з установленим порядком.")
        doc.add_paragraph(f"")
        doc.add_paragraph(f"")
        doc.add_paragraph(f"")
        doc.add_paragraph(f"Матеріально відповідальна особа:")
        doc.add_paragraph(f"{data['from_user']}  Підпис _____")
        doc.add_paragraph(f"Одержав:")
        doc.add_paragraph(f"{data['to_user']}  Підпис _____")
```

```
    elif data['type_of_operation'] == "Наряд на прийняття військового майна":
        doc.add_paragraph(f"№ 43/2025")
        doc.add_paragraph(f"Від  {data['date_of_operation'][:10]}")
        doc.add_paragraph(f"Одержувач: {data['to_user']}")
        doc.add_paragraph(f"ОТРИМАВ ЗІ СКЛАДУ:")
        doc.add_paragraph(f"Майно: {word} {data['asset_type']}  Кількість: {data['quantity']}")
        doc.add_paragraph(f"Отримав: {data['to_user']}, згідно з призначенням.")
        doc.add_paragraph(f"Зобов'язання: Отримане майно облікувати згідно з установленим порядком.")
        doc.add_paragraph(f"")
        doc.add_paragraph(f"")
        doc.add_paragraph(f"")
        doc.add_paragraph(f"Матеріально відповідальна особа:")
        doc.add_paragraph(f"{data['from_user']}  Підпис                    ")
        doc.add_paragraph(f"Одержав:")
        doc.add_paragraph(f"{data['to_user']}  Підпис                    ")

    elif data['type_of_operation'] == "Акт вилучення майна":
        doc.add_paragraph(f"№ 12/2025")
        doc.add_paragraph(f"Від  {data['date_of_operation'][:10]}")
        doc.add_paragraph(f"Вилучає: {data['to_user']}")
        doc.add_paragraph(f"ВИЛУЧЕНО ЗІ СКЛАДУ:")
        doc.add_paragraph(f"Майно: {word} {data['asset_type']}  Кількість: {data['quantity']}")
        doc.add_paragraph(f"Вилучено: {data['to_user']}, згідно з призначенням.")
        doc.add_paragraph(f"Зобов'язання: Вилучене майно облікувати згідно з установленим порядком.")
        doc.add_paragraph(f"")
        doc.add_paragraph(f"")
        doc.add_paragraph(f"")
        doc.add_paragraph(f"Матеріально відповідальна особа:")
        doc.add_paragraph(f"{data['from_user']}  Підпис _____")
        doc.add_paragraph(f"Вилучив:")
        doc.add_paragraph(f"{data['to_user']}  Підпис _____")

    # Збереження документа
    docx_path = f"{save_path}.docx"
    doc.save(docx_path)

    if extension == '.pdf':
        try:
            docx2pdf_convert(docx_path, f"{save_path}.pdf")
            os.remove(docx_path)  # Видалення проміжного DOCX файлу
        except Exception as e:
            messagebox.showerror("Помилка", f"Не вдалося конвертувати в PDF: {e}")

def open_create_document_form(content_frame, user_full_name, canvas):
    for widget in content_frame.winfo_children():
        widget.destroy()

    form_frame = tk.Frame(content_frame, bg="#ffffff", bd=2, relief="solid", padx=30, pady=20)
    form_frame.pack(pady=30, padx=40, fill="both", expand=True)

    title = tk.Label(form_frame, text="Створення нового документа", font=("Arial", 24, "bold"), bg="#ffffff")
    title.grid(row=0, column=0, columnspan=2, pady=20)

    entries = {}
```

```
    # Тип операції
    tk.Label(form_frame, text="Тип документа:", bg="#ffffff", font=("Arial", 16)).grid(row=1, column=0,
sticky="e", pady=10)
    type_values = [
        "Наряд на видання військового майна",
        "Наряд на прийняття військового майна",
        "Акт вилучення майна",
    ]
    type_var = tk.StringVar()
    entries["type_of_operation"] = ttk.Combobox(form_frame, textvariable=type_var, values=type_values,
font=("Arial", 14), width=40, state="readonly")
    entries["type_of_operation"].grid(row=1, column=1, pady=5)

    # From user (автозаповнення)
    tk.Label(form  frame, text="Від:", bg="#ffffff", font=("Arial", 16)).grid(row=2, column=0, sticky="e", pady=10)
    from_user_var = tk.StringVar(value=user_full_name)
    tk.Entry(form_frame, textvariable=from_user_var, font=("Arial", 14), width=43).grid(row=2, column=1,
pady=5)

    # Галочка підтвердження
    confirm_var = tk.BooleanVar(value=False)
    confirm  check = tk.Checkbutton(form  frame, text="Так, це я", variable=confirm  var, bg="#ffffff",
font=("Arial", 14))
    confirm_check.grid(row=3, column=1, sticky="w")

    # To user
    tk.Label(form_frame, text="Отримувач (ПІБ):", bg="#ffffff", font=("Arial", 16)).grid(row=4, column=0,
sticky="e", pady=10)
    to_user_var = tk.StringVar()
    entries["to_user"] = tk.Entry(form_frame, textvariable=to_user_var, font=("Arial", 14), width=43)
    entries["to_user"].grid(row=4, column=1, pady=5)

    # Дата операції
    tk.Label(form_frame, text="Дата операції:", bg="#ffffff", font=("Arial", 16)).grid(row=5, column=0,
sticky="e", pady=10)
    date_var = tk.StringVar(value=datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
    entries["date_of_operation"] = tk.Entry(form_frame, textvariable=date_var, font=("Arial", 14), width=43)
    entries["date_of_operation"].grid(row=5, column=1, pady=5)

    # Категорія
    tk.Label(form_frame, text="Категорія майна:", bg="#ffffff", font=("Arial", 16)).grid(row=6, column=0,
sticky="e", pady=10)
    cat_var = tk.StringVar()
    entries["asset_category"] = ttk.Combobox(form_frame, textvariable=cat_var, values=get_categories(),
font=("Arial", 14), width=40, state="readonly")
    entries["asset_category"].grid(row=6, column=1, pady=5)

    # Тип майна — оновлюється після вибору категорії
    tk.Label(form  frame, text="Тип майна:", bg="#ffffff", font=("Arial", 16)).grid(row=7, column=0, sticky="e",
pady=10)
    asset_type_var = tk.StringVar()
    asset_type_combo = ttk.Combobox(form_frame, textvariable=asset_type_var, font=("Arial", 14), width=40,
state="readonly")
    entries["asset_type"] = asset_type_combo
```

```
    asset_type_combo.grid(row=7, column=1, pady=5)

    def update_asset_types(*args):
        selected_category = cat_var.get()
        types = get_asset_types_by_category(selected_category)
        asset_type_combo["values"] = types
        asset_type_var.set("")

    cat_var.trace("w", update_asset_types)

    # Кількість
    tk.Label(form  frame, text="Кількість:", bg="#ffffff", font=("Arial", 16)).grid(row=8, column=0, sticky="e",
pady=10)
    quantity_var = tk.StringVar()
    entries["quantity"] = tk.Entry(form_frame, textvariable=quantity_var, font=("Arial", 14), width=43)
    entries["quantity"].grid(row=8, column=1, pady=5)

    # Назва файлу
    tk.Label(form  frame, text="Назва файлу:", bg="#ffffff", font=("Arial", 16)).grid(row=9, column=0, sticky="e",
pady=10)
    file_name_input = tk.StringVar()
    entries["file_name_text"] = tk.Entry(form_frame, textvariable=file_name_input, font=("Arial", 14), width=32)
    entries["file_name_text"].grid(row=9, column=1, sticky="w", pady=5)

    extension_var = tk.StringVar(value=".docx")
    extension_combo = ttk.Combobox(form_frame, textvariable=extension_var, values=[".docx", ".pdf"],
font=("Arial", 14), width=5)
    extension_combo.grid(row=9, column=1, sticky="e", pady=5, padx=(0, 10))


    # ======= Зберегти ==========
    def submit_form():
        if not os.path.exists("documents"):
            os.makedirs("documents")

        if not confirm_var.get():
            tk.messagebox.showwarning("Підтвердження", "Підтвердьте, що ви відправник.")
            return

        required_fields = ["type_of_operation", "to_user", "date_of_operation", "asset_category", "asset_type",
"quantity"]
        for field in required_fields:
            if not entries[field].get():
                tk.messagebox.showerror("Помилка", f"Поле '{field}' не заповнене.")
                return

        file_name = file_name_input.get() + extension_var.get()

        data = (
            type_var.get(),
            user_full_name,
            to_user_var.get(),
            date_var.get(),
            asset_type_var.get(),
            cat_var.get(),
```

```
        int(quantity_var.get()),
        file_name
    )

    try:
        conn = sqlite3.connect(OPERATION)
        cursor = conn.cursor()
        cursor.execute(f"""
            INSERT INTO "{OPERATION_TABLE_NAME}" (
                type_of_operation, from_user, to_user, date_of_operation,
                asset_type, asset_category, quantity, file_name
            ) VALUES (?, ?, ?, ?, ?, ?, ?, ?)
        """, data)
        conn.commit()
        conn.close()

        # Генерація документа
        document_data = {
            "type_of_operation": type_var.get(),
            "from_user": user_full_name,
            "to_user": to_user_var.get(),
            "date_of_operation": date_var.get(),
            "asset_type": asset_type_var.get(),
            "asset_category": cat_var.get(),
            "quantity": quantity_var.get()
        }
        file_base = os.path.join("documents", file_name_input.get())
        generate_document(document_data, file_base, extension_var.get())

        doc_type = entries["type_of_operation"].get()
        asset_category = entries["asset_category"].get()
        asset_type = entries["asset_type"].get()
        quantity = int(entries["quantity"].get())

        if AUTO_SYNC_ENABLED:
            conn = sqlite3.connect(HELPINFO_DB_PATH)
            cursor = conn.cursor()

            if doc_type == "Наряд на прийняття військового майна":
                cursor.execute(f"""
                        INSERT INTO "{USER_INFO_TABLE_NAME}" (asset_type, asset_category, quantity)
                        VALUES (?, ?, ?)
                    """, (asset_type, asset_category, quantity))
            elif doc_type in ["Наряд на видання військового майна", "Акт вилучення майна"]:
                cursor.execute(f"""
                        SELECT quantity FROM "{USER_INFO_TABLE_NAME}"
                        WHERE asset_type=? AND asset_category=?
                    """, (asset_type, asset_category))
                result = cursor.fetchone()
                if result:
                    new_quantity = result[0] - quantity
                    if new_quantity <= 0:
                        cursor.execute(f"""
                                DELETE FROM "{USER_INFO_TABLE_NAME}"
                                WHERE asset_type=? AND asset_category=?
```

```
                    """, (asset_type, asset_category))
                else:
                    cursor.execute(f"""
                        UPDATE "{USER_INFO_TABLE_NAME}"
                        SET quantity=?
                        WHERE asset_type=? AND asset_category=?
                    """, (new_quantity, asset_type, asset_category))
            # Якщо немає в базі — нічого не робимо
        conn.commit()
        conn.close()

        tk.messagebox.showinfo("Успіх", "Документ успішно створено.")
    except Exception as e:
        tk.messagebox.showerror("Помилка", f"Не вдалося записати у БД: {e}")

    submit_btn = tk.Button(form_frame, text="Зберегти документ", command=submit_form, font=("Arial", 16,
"bold"), bg="#2ecc71", fg="white", padx=20, pady=10)
    submit_btn.grid(row=10, column=0, columnspan=2, pady=30)

    canvas.yview_moveto(0)

def show_additional_page_for_extended(parent, canvas):
    for widget in parent.winfo_children():
        widget.destroy()

    title = tk.Label(parent, text="Додаткові функції", font=("Arial", 44, "bold"), bg="#ecf0f1", fg="#2c3e50")
    title.pack(pady=40)

    container = tk.Frame(parent, bg="#ecf0f1")
    container.pack(padx=80, pady=20, fill="both", expand=True)

    # --- Реєстрація облікового запису ---
    register_frame = tk.LabelFrame(container, text="➕ Додати новий обліковий запис",
                                   font=("Arial", 28), bg="#ecf0f1", padx=20, pady=20)
    register_frame.pack(fill="x", pady=20)

    labels = [
        "Прізвище:", "Ім'я:", "По батькові:", "Пошта:", "Пароль:", "Посада:"
    ]

    for i, label in enumerate(labels):
        tk.Label(register_frame, text=label, bg="#ecf0f1", font=("Arial", 22)).grid(row=i, column=0, sticky="e",
pady=4)

    entry_surname = ttk.Entry(register_frame, width=30, font=("Arial", 22))
    entry_name = ttk.Entry(register_frame, width=30, font=("Arial", 22))
    entry_patronymic = ttk.Entry(register_frame, width=30, font=("Arial", 22))
    entry_email = ttk.Entry(register_frame, width=30, font=("Arial", 22))
    entry_password = ttk.Entry(register_frame, show="*", width=30, font=("Arial", 22))
    positions_list = [
        " Начальник служби речового забезпечення", " Начальник служби озброєння", " Начальник тилу",
        " Командир роти", " Командир взводу", " Командир батальйону", " Інтендант", " Комендант складу",
        " Начальник служби ПММ", " Військовий бухгалтер", " Фінансист", " Оператор автоматизованої
системи обліку",
        " Інспектор тилового забезпечення", " Ревізор матеріального забезпечення",
```

```
        " Відповідальний за мобілізаційне забезпечення", " Старшина підрозділу", " Заступник командира з озброєння",
        " Начальник речового складу", " Інженер з експлуатації озброєння", " Адміністратор системи обліку",
        " Аналітик логістики", " Офіцер з логістики", " Помічник начальника речслужби",
        " Інженер технічного забезпечення", " Старший технік", " Сержант зі складу", " Технік-інвентаризатор",
        " Офіцер з матеріального забезпечення", " Військовий логіст", " Контролер обліку майна",
        " Інструктор складу", " Офіцер тилу", " Керівник складу", " Тиловик", " Офіцер з експлуатації майна",
        " Інженер з нормування постачання", " Начальник постачання підрозділу", " Відповідальний за зброю",
        " Обліковець майна", " Діловод", " Старший оператор", " Оператор технічного забезпечення",
        " Армійський інженер", " Технік з обліку", " Спеціаліст з логістичних даних",
        " Офіцер служби забезпечення", " Заступник начальника складу", " Інженер з автоматизованих систем",
        " Помічник інтенданта"
    ]
    entry_position = AutocompleteEntry(positions_list, register_frame, width=30, font=("Arial", 22))

    entries = [entry_surname, entry_name, entry_patronymic, entry_email, entry_password, entry_position]
    for i, entry in enumerate(entries):
        entry.grid(row=i, column=1, pady=4)

    photo_path_var = tk.StringVar()

    def choose_photo():
        file_path = filedialog.askopenfilename(title="Виберіть фото", filetypes=[("Зображення", "*.jpg *.png *.jpeg")])
        if file_path:
            filename = os.path.basename(file_path)
            save_path = os.path.join("data/img", filename).replace("\\", "/")
            os.makedirs("data/img", exist_ok=True)
            photo_path_var.set(save_path)

    photo_button = tk.Button(register_frame, text="Обрати фото", command=choose_photo, font=("Arial", int(22 * 0.6)))
    photo_button.grid(row=6, column=1, sticky="e", pady=8)

    style = ttk.Style()
    style.configure("Big.TCheckbutton", font=("Arial", 17))

    agree_var = tk.BooleanVar()
    agree_check = ttk.Checkbutton(register_frame, text="Я уважно перевірив дані", variable=agree_var, style="Big.TCheckbutton")
    agree_check.grid(row=7, column=0, sticky="w", pady=8, padx=(0, 20))

    def submit_registration():
        if not agree_var.get():
            messagebox.showerror("Помилка", "Підтвердіть, що ви перевірили дані.")
            return

        full_name = f"{entry_surname.get().strip()} {entry_name.get().strip()} {entry_patronymic.get().strip()}"
        email = entry_email.get().strip()
        password = entry_password.get().strip()
        position = entry_position.get().strip()
        photo = photo_path_var.get()
```

```
        if not all([full_name, email, password, position]):
            messagebox.showerror("Помилка", "Будь ласка, заповніть усі поля.")
            return

        success, msg = register_user(full_name, email, password, position, "Спрощений", photo)
        if success:
            messagebox.showinfo("Успіх", msg)
            for entry in entries:
                entry.delete(0, tk.END)
            agree_var.set(False)
            photo_path_var.set("")
        else:
            messagebox.showerror("Помилка", msg)

    tk.Button(register_frame, text="Зареєструвати", command=submit_registration, font=("Arial", 22)).grid(
        row=8, column=0, sticky="w", pady=20, padx=(0, 20)
    )

    # --- Додати новий тип майна ---
    add_frame = tk.LabelFrame(container, text=" 📦 Додати новий тип майна",
                              font=("Arial", 28), bg="#ecf0f1", padx=20, pady=20)
    add_frame.pack(fill="x", pady=20)

    labels = ["Назва майна:", "Категорія:", "Асоціативне слово:", "В кількості 2-4:", "В множині:"]
    entries = {}

    for i, label_text in enumerate(labels):
        tk.Label(add_frame, text=label_text, font=("Arial", 18), bg="#ecf0f1").grid(row=i, column=0, sticky="e",
padx=10, pady=5)

    entries["name"] = ttk.Entry(add_frame, font=("Arial", 18), width=40)
    entries["name"].grid(row=0, column=1, pady=5)

    categories = get_categories()
    cat_var = tk.StringVar()
    entries["category"] = ttk.Combobox(add_frame, textvariable=cat_var, values=categories, font=("Arial", 18),
width=37)
    entries["category"].grid(row=1, column=1, pady=5)

    entries["keyword"] = ttk.Entry(add_frame, font=("Arial", 18), width=40)
    entries["keyword"].grid(row=2, column=1, pady=5)

    entries["plural_2_4"] = ttk.Entry(add_frame, font=("Arial", 18), width=40)
    entries["plural_2_4"].grid(row=3, column=1, pady=5)

    entries["plural"] = ttk.Entry(add_frame, font=("Arial", 18), width=40)
    entries["plural"].grid(row=4, column=1, pady=5)

    def on_add():
        name = entries["name"].get().strip()
        category = entries["category"].get().strip()
        keyword = entries["keyword"].get().strip()
        plural_2_4 = entries["plural_2_4"].get().strip()
        plural = entries["plural"].get().strip()
```

```
    if not all([name, category, keyword, plural_2_4, plural]):
        messagebox.showerror("Помилка", "Будь ласка, заповніть усі поля.")
        return

    success = add_inventory_type(name, category, keyword, plural_2_4, plural)
    if success:
        messagebox.showinfo("Успіх", f"Тип майна '{name}' додано успішно!")
        show_additional_page_for_extended(parent)
        for e in entries.values():
            e.delete(0, tk.END)
        entries["category"]["values"] = get_categories()
    else:
        messagebox.showerror("Помилка", "Не вдалося додати тип майна.")

add_btn = tk.Button(add_frame, text="Створити", font=("Arial", 20), command=on_add)
add_btn.grid(row=5, column=0, columnspan=2, pady=15)

# --- Видалити тип майна ---
del_frame = tk.LabelFrame(container, text=" 🗑 Видалити тип майна",
                          font=("Arial", 28), bg="#ecf0f1", padx=20, pady=20)
del_frame.pack(fill="x", pady=20)

tk.Label(del_frame, text="Оберіть майно для видалення (почніть вводити):",
         font=("Arial", 18), bg="#ecf0f1").grid(row=0, column=0, sticky="w", padx=10, pady=5)

conn = sqlite3.connect(INVENTORY_DB)
cursor = conn.cursor()
cursor.execute("SELECT name FROM inventory ORDER BY name")
all_names = [row[0] for row in cursor.fetchall()]
conn.close()

name_var = tk.StringVar()
del_entry = AutocompleteEntry(all_names, del_frame, font=("Arial", 18), width=40, textvariable=name_var)
del_entry.grid(row=1, column=0, padx=10, pady=5)

def on_delete():
    name = del_entry.get().strip()
    print(f"DEBUG: name='{name}'")
    if not name:
        messagebox.showerror("Помилка", "Введіть назву майна для видалення.")
        return

    if name not in all_names:
        messagebox.showerror("Помилка", f"Тип майна '{name}' не знайдено.")
        return

    if messagebox.askyesno("Підтвердження", f"Ви дійсно хочете видалити тип майна '{name}'?"):
        success = delete_inventory_type_by_name(name)
        if success:
            messagebox.showinfo("Успіх", f"Тип майна '{name}' видалено.")
            name_var.set("")
            show_additional_page_for_extended(parent)
        else:
            messagebox.showerror("Помилка", "Не вдалося видалити тип майна.")
```

```
    del_btn = tk.Button(del_frame, text="Видалити", font=("Arial", 20), command=on_delete)
    del_btn.grid(row=2, column=0, pady=15)

    canvas.yview_moveto(0)

def show_additional_page_for_full(parent, canvas):
    for widget in parent.winfo_children():
        widget.destroy()

    title = tk.Label(parent, text="Адміністративна панель (Повний доступ)", font=("Arial", 44, "bold"),
bg="#ecf0f1", fg="#2c3e50")
    title.pack(pady=40)

    container = tk.Frame(parent, bg="#ecf0f1")
    container.pack(padx=80, pady=20, fill="both", expand=True)

    # ═══ Форма реєстрації нового користувача ═══
    register_frame = tk.LabelFrame(container, text=" ➕ Додати новий обліковий запис", font=("Arial", 28),
bg="#ecf0f1", padx=20, pady=20)
    register_frame.pack(fill="x", pady=20)

    labels = ["Прізвище:", "Ім'я:", "По батькові:", "Пошта:", "Пароль:", "Посада:", "Рівень доступу:"]

    for i, label in enumerate(labels):
        tk.Label(register_frame, text=label, bg="#ecf0f1", font=("Arial", 22)).grid(row=i, column=0, sticky="e",
pady=4)

    entry_surname = ttk.Entry(register_frame, width=30, font=("Arial", 22))
    entry_name = ttk.Entry(register_frame, width=30, font=("Arial", 22))
    entry_patronymic = ttk.Entry(register_frame, width=30, font=("Arial", 22))
    entry_email = ttk.Entry(register_frame, width=30, font=("Arial", 22))
    entry_password = ttk.Entry(register_frame, show="*", width=30, font=("Arial", 22))
    positions_list = [
        " Начальник служби речового забезпечення", " Начальник служби озброєння", " Начальник тилу", "
Командир роти", " Командир взводу", " Командир батальйону", " Інтендант", " Комендант складу", "
Начальник служби ПММ", " Військовий бухгалтер", " Фінансист", " Оператор автоматизованої системи
обліку", " Інспектор тилового забезпечення", " Ревізор матеріального забезпечення", " Відповідальний за
мобілізаційне забезпечення", " Старшина підрозділу", " Заступник командира з озброєння", " Начальник
речового складу", " Інженер з експлуатації озброєння", " Адміністратор системи обліку", " Аналітик
логістики", " Офіцер з логістики", " Помічник начальника речслужби", " Інженер технічного забезпечення",
" Старший технік", " Сержант зі складу", " Технік-інвентаризатор", " Офіцер з матеріального забезпечення",
" Військовий логіст", " Контролер обліку майна", " Інструктор складу", " Офіцер тилу", " Керівник складу",
" Тиловик", " Офіцер з експлуатації майна", " Інженер з нормування постачання", " Начальник постачання
підрозділу", " Відповідальний за зброю", " Обліковець майна", " Діловод", " Старший оператор", " Оператор
технічного забезпечення", " Армійський інженер", " Технік з обліку", " Спеціаліст з логістичних даних", "
Офіцер служби забезпечення", " Заступник начальника складу", " Інженер з автоматизованих систем", "
Помічник інтенданта"
    ]
    entry_position = AutocompleteEntry(positions_list, register_frame, width=30, font=("Arial", 22))

    confirm_var = tk.BooleanVar()
    confirm_check = ttk.Checkbutton(register_frame, text="Я погоджуюсь надати вказаний рівень доступу",
variable=confirm_var, style="Big.TCheckbutton")
    confirm_check.grid(row=7, column=0, sticky="w", pady=4)
```

```
    access_var = tk.StringVar(value="Спрощений")
    access_combo = ttk.Combobox(register_frame, textvariable=access_var, values=["Спрощений",
"Розширений", "Повний"], font=("Arial", 22), state="readonly", width=28)

    entries = [entry_surname, entry_name, entry_patronymic, entry_email, entry_password, entry_position,
access_combo]
    for i, entry in enumerate(entries):
        entry.grid(row=i, column=1, pady=4)

    # === Кнопка вибору фото ===
    photo_path_var = tk.StringVar()

    def choose_photo():
        file  path = filedialog.askopenfilename(title="Виберіть фото", filetypes=[("Зображення", "*.jpg *.png
*.jpeg")])
        if file_path:
            filename = os.path.basename(file_path)
            save_path = os.path.join("data/img", filename).replace("\\", "/")
            os.makedirs("data/img", exist_ok=True)
            photo_path_var.set(save_path)

    photo_button = tk.Button(register_frame, text="Обрати фото", command=choose_photo, font=("Arial", int(22 *
0.6)))
    photo_button.grid(row=7, column=1, sticky="e", pady=8)

    # === Підтвердження введення ===
    agree_var = tk.BooleanVar()
    ttk.Style().configure("Big.TCheckbutton", font=("Arial", 17))
    agree_check = ttk.Checkbutton(register_frame, text="Я уважно перевірив дані", variable=agree_var,
style="Big.TCheckbutton")
    agree_check.grid(row=8, column=0, sticky="w", pady=8, padx=(0, 20))

    def submit_registration():
        if not agree_var.get():
            messagebox.showerror("Помилка", "Підтвердіть, що ви перевірили дані.")
            return

        full_name = f"{entry_surname.get().strip()} {entry_name.get().strip()} {entry_patronymic.get().strip()}"
        email = entry_email.get().strip()
        password = entry_password.get().strip()
        position = entry_position.get().strip()
        access_level = access_var.get()
        photo = photo_path_var.get()
        confirm_grant = confirm_var.get()

        if not confirm_grant:
            messagebox.showerror("Помилка", "Потрібно підтвердити надання доступу.")
            return

        if not all([full_name, email, password, position]):
            messagebox.showerror("Помилка", "Будь ласка, заповніть усі поля.")
            return

        success, msg = register_user(full_name, email, password, position, access_level, photo)
        if success:
```

```
            messagebox.showinfo("Успіх", msg)
            for entry in entries[:-1]:  # combobox окремо не має .delete
                entry.delete(0, tk.END)
            access_combo.set("Спрощений")
            agree_var.set(False)
            photo_path_var.set("")
        else:
            messagebox.showerror("Помилка", msg)

    tk.Button(register_frame, text="Зареєструвати", command=submit_registration, font=("Arial", 22)).grid(row=9, column=0, sticky="w", pady=20, padx=(0, 20))

    edit_frame = tk.LabelFrame(container, text="🖍️ Редагування облікових записів", font=("Arial", 28), bg="#ecf0f1", padx=20, pady=20)
    edit_frame.pack(fill="x", pady=20)

    tk.Label(edit_frame, text="Email акаунту для редагування:", bg="#ecf0f1", font=("Arial", 22)).grid(row=0, column=0, pady=4, sticky="e")
    edit_email_entry = ttk.Entry(edit_frame, width=30, font=("Arial", 22))
    edit_email_entry.grid(row=0, column=1, pady=4)

    tk.Label(edit_frame, text="Нове ПІБ:", bg="#ecf0f1", font=("Arial", 22)).grid(row=1, column=0, pady=4, sticky="e")
    new_name_entry = ttk.Entry(edit_frame, width=30, font=("Arial", 22))
    new_name_entry.grid(row=1, column=1, pady=4)

    tk.Label(edit_frame, text="Нова пошта:", bg="#ecf0f1", font=("Arial", 22)).grid(row=2, column=0, pady=4, sticky="e")
    new_email_entry = ttk.Entry(edit_frame, width=30, font=("Arial", 22))
    new_email_entry.grid(row=2, column=1, pady=4)

    tk.Label(edit_frame, text="Новий пароль:", bg="#ecf0f1", font=("Arial", 22)).grid(row=3, column=0, pady=4, sticky="e")
    new_password_entry = ttk.Entry(edit_frame, width=30, font=("Arial", 22), show="*")
    new_password_entry.grid(row=3, column=1, pady=4)

    photo_path_var = tk.StringVar()
    # Ініціалізуєш photo_path_var з існуючим шляхом до фото

    tk.Label(edit_frame, text="Фото:", font=("Arial", 22), bg="#ecf0f1").grid(row=4, column=0, sticky="e")
    tk.Button(edit_frame, text="Змінити фото", command=lambda: choose_new_photo(photo_path_var), font=("Arial", 16)).grid(row=4, column=1, sticky="w")

    def apply_changes():
        email = edit_email_entry.get().strip()
        new_name = new_name_entry.get().strip()
        new_email = new_email_entry.get().strip()
        new_password = new_password_entry.get().strip()
        new_photo_path = photo_path_var.get().strip()

        if not email:
            messagebox.showerror("Помилка", "Вкажіть email для редагування.")
            return

        conn = sqlite3.connect(DB_FILE)
```

```
        cursor = conn.cursor()

        if new_name:
            cursor.execute("UPDATE users SET full_name = ? WHERE email = ?", (new_name, email))
        if new_email:
            cursor.execute("UPDATE users SET email = ? WHERE email = ?", (new_email, email))
        if new_password:
            cursor.execute("UPDATE users SET password_hash = ? WHERE email = ?",
(hash_password(new_password), email))
        if new_photo_path:
            cursor.execute("UPDATE users SET photo_filename = ? WHERE email = ?", (new_photo_path, email))

        conn.commit()
        conn.close()
        messagebox.showinfo("Успіх", "Дані оновлено!")

    def choose_new_photo(photo_var):
        file_path = filedialog.askopenfilename(title="Виберіть нове фото", filetypes=[("Зображення", "*.jpg *.png
*.jpeg")])
        if file_path:
            filename = os.path.basename(file_path)
            save_path = os.path.join("data/img", filename).replace("\\", "/")
            os.makedirs("data/img", exist_ok=True)
            photo_var.set(save_path)
            messagebox.showinfo("Успіх", "Фото успішно обрано.")

    tk.Button(edit_frame, text="Застосувати зміни", command=apply_changes, font=("Arial", 22)).grid(row=5,
column=0, columnspan=2, pady=10)

    delete_frame = tk.LabelFrame(container, text="❌ Видалення облікового запису", font=("Arial", 28),
bg="#ecf0f1", padx=20, pady=20)
    delete_frame.pack(fill="x", pady=20)

    tk.Label(delete_frame, text="Email для видалення:", bg="#ecf0f1", font=("Arial", 22)).grid(row=0, column=0,
pady=4,
                                                                                              sticky="e")
    delete_email_entry = ttk.Entry(delete_frame, width=30, font=("Arial", 22))
    delete_email_entry.grid(row=0, column=1, pady=4)

    def delete_account():
        email = delete_email_entry.get().strip()
        if not email:
            messagebox.showerror("Помилка", "Вкажіть email для видалення.")
            return

        def confirm_and_delete_user(user_id):
            answer = messagebox.askyesno("Підтвердження", "Ви впевнені, що хочете видалити цей обліковий
запис?")
            if answer:
                conn = sqlite3.connect(DB_FILE)
                cursor = conn.cursor()
                cursor.execute("DELETE FROM users WHERE id = ?", (user_id,))
                conn.commit()
                conn.close()
                messagebox.showinfo("Успіх", "Обліковий запис видалено.")
```

```
            else:
                messagebox.showinfo("Скасовано", "Видалення скасовано.")

        confirm = messagebox.askyesno("Підтвердження", f"Ви впевнені, що хочете видалити акаунт: {email}?")
        if confirm:
            conn = sqlite3.connect(DB_FILE)
            cursor = conn.cursor()
            cursor.execute("SELECT id FROM users WHERE email = ?", (email,))
            result = cursor.fetchone()
            conn.close()

            if result:
                user_id = result[0]
                confirm_and_delete_user(user_id)
            else:
                messagebox.showerror("Помилка", "Користувача з такою поштою не знайдено.")

    tk.Button(delete_frame, text="Видалити акаунт", command=delete_account, font=("Arial", 22),
fg="red").grid(row=1, column=0, columnspan=2, pady=10)

    # --- Додати новий тип майна ---
    add_frame = tk.LabelFrame(container, text=" 📦 Додати новий тип майна",
                              font=("Arial", 28), bg="#ecf0f1", padx=20, pady=20)
    add_frame.pack(fill="x", pady=20)

    labels = ["Назва майна:", "Категорія:", "Асоціативне слово:", "В кількості 2-4:", "В множині:"]
    entries = {}

    for i, label_text in enumerate(labels):
        tk.Label(add_frame, text=label_text, font=("Arial", 18), bg="#ecf0f1").grid(row=i, column=0, sticky="e",
                                                                                    padx=10, pady=5)

    entries["name"] = ttk.Entry(add_frame, font=("Arial", 18), width=40)
    entries["name"].grid(row=0, column=1, pady=5)

    categories = get_categories()
    cat_var = tk.StringVar()
    entries["category"] = ttk.Combobox(add_frame, textvariable=cat_var, values=categories, font=("Arial", 18),
width=37)
    entries["category"].grid(row=1, column=1, pady=5)

    entries["keyword"] = ttk.Entry(add_frame, font=("Arial", 18), width=40)
    entries["keyword"].grid(row=2, column=1, pady=5)

    entries["plural_2_4"] = ttk.Entry(add_frame, font=("Arial", 18), width=40)
    entries["plural_2_4"].grid(row=3, column=1, pady=5)

    entries["plural"] = ttk.Entry(add_frame, font=("Arial", 18), width=40)
    entries["plural"].grid(row=4, column=1, pady=5)

    def on_add():
        name = entries["name"].get().strip()
        category = entries["category"].get().strip()
        keyword = entries["keyword"].get().strip()
        plural_2_4 = entries["plural_2_4"].get().strip()
```

```
    plural = entries["plural"].get().strip()

    if not all([name, category, keyword, plural_2_4, plural]):
        messagebox.showerror("Помилка", "Будь ласка, заповніть усі поля.")
        return

    success = add_inventory_type(name, category, keyword, plural_2_4, plural)
    if success:
        messagebox.showinfo("Успіх", f"Тип майна '{name}' додано успішно!")
        show_additional_page_for_full(parent)
        for e in entries.values():
            e.delete(0, tk.END)
        entries["category"]["values"] = get_categories()
    else:
        messagebox.showerror("Помилка", "Не вдалося додати тип майна.")

add_btn = tk.Button(add_frame, text="Створити", font=("Arial", 20), command=on_add)
add_btn.grid(row=5, column=0, columnspan=2, pady=15)

# --- Видалити тип майна ---
del_frame = tk.LabelFrame(container, text=" 🗑 Видалити тип майна",
                          font=("Arial", 28), bg="#ecf0f1", padx=20, pady=20)
del_frame.pack(fill="x", pady=20)

tk.Label(del  frame, text="Оберіть майно для видалення (почніть вводити):",
         font=("Arial", 18), bg="#ecf0f1").grid(row=0, column=0, sticky="w", padx=10, pady=5)

conn = sqlite3.connect(INVENTORY_DB)
cursor = conn.cursor()
cursor.execute("SELECT name FROM inventory ORDER BY name")
all_names = [row[0] for row in cursor.fetchall()]
conn.close()

name_var = tk.StringVar()
del_entry = AutocompleteEntry(all_names, del_frame, font=("Arial", 18), width=40, textvariable=name_var)
del_entry.grid(row=1, column=0, padx=10, pady=5)

def on_delete():
    name = del_entry.get().strip()
    print(f"DEBUG: name='{name}'")
    if not name:
        messagebox.showerror("Помилка", "Введіть назву майна для видалення.")
        return

    if name not in all_names:
        messagebox.showerror("Помилка", f"Тип майна '{name}' не знайдено.")
        return

    if messagebox.askyesno("Підтвердження", f"Ви дійсно хочете видалити тип майна '{name}'?"):
        success = delete_inventory_type_by_name(name)
        if success:
            messagebox.showinfo("Успіх", f"Тип майна '{name}' видалено.")
            name_var.set("")
            show_additional_page_for_full(parent)
        else:
```

```
            messagebox.showerror("Помилка", "Не вдалося видалити тип майна.")

    del_btn = tk.Button(del_frame, text="Видалити", font=("Arial", 20), command=on_delete)
    del_btn.grid(row=2, column=0, pady=15)

    import_frame = tk.LabelFrame(container, text=" 📥 Імпорт типів майна з Excel", font=("Arial", 28),
bg="#ecf0f1", padx=20, pady=20)
    import_frame.pack(fill="x", pady=20)

    tk.Label(import_frame, text="Імпортувати типи майна до бази даних:", bg="#ecf0f1", font=("Arial",
22)).grid(row=0, column=0, pady=10, sticky="w")
    tk.Button(import_frame, text="Імпортувати з Excel", command=import_inventory_from_excel, font=("Arial",
22)).grid(row=0, column=1, pady=10, padx=20)

    # --- Додати нову групу майна ---
    type_add_frame = tk.LabelFrame(container, text=" 📦 Додати нову групу майна", font=("Arial", 28),
bg="#ecf0f1",
                                   padx=20, pady=20)
    type_add_frame.pack(fill="x", pady=20)

    labels_group = ["Назва майна:", "Нова група (категорія):", "Асоціативне слово:", "В кількості 2-4:", "В
множині:"]
    entries_group = {}

    for i, label_text in enumerate(labels_group):
        tk.Label(type_add_frame, text=label_text, font=("Arial", 18), bg="#ecf0f1").grid(row=i, column=0,
sticky="e",
                                                                                         padx=10, pady=5)

    entries_group["name"] = ttk.Entry(type_add_frame, font=("Arial", 18), width=40)
    entries_group["name"].grid(row=0, column=1, pady=5)

    entries_group["category"] = ttk.Entry(type_add_frame, font=("Arial", 18), width=40)  # нова група вводиться
текстом
    entries_group["category"].grid(row=1, column=1, pady=5)

    entries_group["keyword"] = ttk.Entry(type_add_frame, font=("Arial", 18), width=40)
    entries_group["keyword"].grid(row=2, column=1, pady=5)

    entries_group["plural_2_4"] = ttk.Entry(type_add_frame, font=("Arial", 18), width=40)
    entries_group["plural_2_4"].grid(row=3, column=1, pady=5)

    entries_group["plural"] = ttk.Entry(type_add_frame, font=("Arial", 18), width=40)
    entries_group["plural"].grid(row=4, column=1, pady=5)

    def on_add_group():
        name = entries_group["name"].get().strip()
        category = entries_group["category"].get().strip()
        keyword = entries_group["keyword"].get().strip()
        plural_2_4 = entries_group["plural_2_4"].get().strip()
        plural = entries_group["plural"].get().strip()

        if not all([name, category, keyword, plural_2_4, plural]):
            messagebox.showerror("Помилка", "Будь ласка, заповніть усі поля.")
            return
```

```
        success = add_inventory_type(name, category, keyword, plural_2_4, plural)
        if success:
            messagebox.showinfo("Успіх", f"Група майна '{category}' з типом '{name}' додано успішно!")
            show_additional_page_for_full(parent)
            # Оновлення сторінки або список категорій
            for e in entries_group.values():
                e.delete(0, tk.END)
            entries["category"]["values"] = get_categories()  # оновити Combobox категорій у іншому місці, якщо
треба
        else:
            messagebox.showerror("Помилка", "Не вдалося додати групу майна.")

    add_group_btn = tk.Button(type_add_frame, text="Створити групу з типом", font=("Arial", 20),
command=on_add_group)
    add_group_btn.grid(row=5, column=0, columnspan=2, pady=15)

    # --- Видалити групу майна ---
    type_delete_frame = tk.LabelFrame(container, text=" 🗑 Видалити групу майна", font=("Arial", 28),
bg="#ecf0f1",
                                      padx=20, pady=20)
    type_delete_frame.pack(fill="x", pady=20)

    tk.Label(type_delete_frame, text="Оберіть групу для видалення:",
             font=("Arial", 18), bg="#ecf0f1").grid(row=0, column=0, sticky="w", padx=10, pady=5)

    categories = get_categories()  # список категорій із БД
    group_var = tk.StringVar()
    group_combobox = ttk.Combobox(type_delete_frame, textvariable=group_var, values=categories, font=("Arial",
18),
                                  width=40)
    group_combobox.grid(row=1, column=0, padx=10, pady=5)

    def delete_inventory_group_by_category(category):
        conn = sqlite3.connect(INVENTORY_DB)
        cursor = conn.cursor()
        cursor.execute("DELETE FROM inventory WHERE category = ?", (category,))
        deleted = cursor.rowcount
        conn.commit()
        conn.close()
        return deleted > 0

    def on_delete_group():
        category = group_var.get().strip()
        if not category:
            messagebox.showerror("Помилка", "Оберіть групу для видалення.")
            return

        if category not in categories:
            messagebox.showerror("Помилка", f"Групу '{category}' не знайдено.")
            return

        if messagebox.askyesno("Підтвердження",
                               f"Ви дійсно хочете видалити групу майна '{category}' та всі типи в ній?"):
            success = delete_inventory_group_by_category(category)
```

```
            if success:
                messagebox.showinfo("Успіх", f"Групу майна '{category}' видалено.")
                show_additional_page_for_full(parent)
                group_var.set("")
                # Оновити Combobox з категоріями після видалення
                new_categories = get_categories()
                group_combobox['values'] = new_categories
            else:
                messagebox.showerror("Помилка", "Не вдалося видалити групу майна.")

    delete_group_btn = tk.Button(type_delete_frame, text="Видалити групу", font=("Arial", 20),
command=on_delete_group)
    delete_group_btn.grid(row=2, column=0, pady=15)

    canvas.yview_moveto(0)

def open_settings_window():
    settings_window = tk.Toplevel()
    settings  window.title("Налаштування")
    settings_window.geometry("400x200")
    settings_window.config(bg="#ffffff")

    global AUTO_SYNC_ENABLED
    sync_var = tk.BooleanVar(value=AUTO_SYNC_ENABLED)

    def save_settings():
        global AUTO_SYNC_ENABLED
        AUTO_SYNC_ENABLED = sync_var.get()
        save_settings_to_file()
        settings_window.destroy()

    tk.Checkbutton(settings_window, text="Синхронізувати базу майна автоматично",
                   variable=sync_var, font=("Arial", 14), bg="#ffffff").pack(pady=20)

    tk.Button(settings_window, text="Зберегти", font=("Arial", 12),
              command=save_settings).pack(pady=10)

def open_main_window(user):
    main_window = tk.Tk()
    set_window_icon(main_window)
    main_window.title("ПмОМ - Головна панель")
    main_window.attributes("-fullscreen", True)

    photo_path = user[4] if user[4] else "data/img/soldier.png"

    # Ліве меню (розширено до 375 пікселів)
    menu_frame = tk.Frame(main_window, width=375, height=1080, bg="#2c3e50")
    menu_frame.pack(side="left", fill="y")

    # === Scrollable frame ===
    canvas_container = tk.Frame(main_window, bg="#ecf0f1")
    canvas_container.pack(side="right", fill="both", expand=True)

    canvas = tk.Canvas(canvas_container, bg="#ecf0f1")
    scrollbar = ttk.Scrollbar(canvas_container, orient="vertical", command=canvas.yview)
```

```
canvas.configure(yscrollcommand=scrollbar.set)

scrollbar.pack(side="right", fill="y")
canvas.pack(side="left", fill="both", expand=True)

scrollable_frame = tk.Frame(canvas, bg="#ecf0f1")
canvas.create_window((0, 0), window=scrollable_frame, anchor="nw", tags="scrollable_window")

scrollable_frame.bind(
    "<Configure>",
    lambda e: canvas.configure(scrollregion=canvas.bbox("all"))
)

# Зберігаємо посилання на scrollable_frame як content_frame
content_frame = scrollable_frame

def resize_scrollable_frame(event):
    canvas.itemconfig("scrollable_window", width=event.width)

canvas.bind("<Configure>", resize_scrollable_frame)

def on_mousewheel(event):
    canvas.yview_scroll(int(-1 * (event.delta / 120)), "units")

canvas.bind_all("<MouseWheel>", on_mousewheel)  # для Windows
canvas.bind_all("<Button-4>", lambda e: canvas.yview_scroll(-1, "units"))  # Linux scroll up
canvas.bind_all("<Button-5>", lambda e: canvas.yview_scroll(1, "units"))  # Linux scroll down

# ======== Меню: інформація про користувача ========
# Фото користувача
try:
    img = Image.open(photo_path)
    img = img.resize((160, 240), Image.Resampling.LANCZOS)
    border_size = 4
    bordered_img = Image.new("RGB", (img.width + 2 * border_size, img.height + 2 * border_size), "#ecf0f1")
    bordered_img.paste(img, (border_size, border_size))
    photo = ImageTk.PhotoImage(bordered_img)
    photo_label = tk.Label(menu_frame, image=photo, bg="#2c3e50")
    photo_label.image = photo
    photo_label.pack(pady=(30, 10))
except:
    photo_label = tk.Label(menu_frame, text="[Фото]", bg="#2c3e50", fg="white", font=("Arial", 12))
    photo_label.pack(pady=(30, 10))

# Створення інформації з бази даних
user_info_raw = {
    "ПІБ": user[0],        # full_name
    "Пошта": user[1],      # email
    "Посада": user[2],     # position
    "Доступ": user[3]      # access_level
}

def truncate_text(text, max_chars=32):
    return text if len(text) <= max_chars else text[:max_chars - 3] + "..."
```

```
    # Відображення полів користувача
    for label, value in user_info_raw.items():
        frame = tk.Frame(menu_frame, bg="#2c3e50")
        frame.pack(anchor="w", padx=15, pady=2, fill="x")

        tk.Label(
            frame,
            text=f"{label}:",
            bg="#2c3e50",
            fg="white",
            font=("Arial", 10, "bold")
        ).pack(side="left")

        tk.Label(
            frame,
            text=truncate_text(value),
            bg="#2c3e50",
            fg="white",
            font=("Arial", 10)
        ).pack(side="left", padx=(4, 0))


    ttk.Separator(menu_frame, orient="horizontal").pack(fill="x", pady=15, padx=10)

    # ======== Кнопки меню (не змінились) ========
    def add_menu_button(text, command):
        ttk.Button(
            menu_frame,
            text=text,
            command=command,
            style="Menu.TButton"
        ).pack(pady=6, padx=20, fill="x")

    style = ttk.Style()
    style.configure("Menu.TButton", font=("Arial", 11, "bold"))

    # Кнопки
    menu_buttons = [
        ("Головна", lambda: show_main_content(content_frame, canvas)),
        ("Список майна військовослужбовця", lambda: show_user_asset_list(user[0], content_frame, canvas)),
        ("Нещодавні акти й заяви", lambda: show_recent_operations(content_frame, canvas, user[0])),
        ("Створити новий документ", lambda: open_create_document_form(content_frame, user[0], canvas)),
        ("Додатково", lambda: handle_additional_menu(user, content_frame, canvas)),
        ("Вихід", lambda: exit_confirmation(main_window))
    ]

    for text, command in menu_buttons:
        add_menu_button(text, command)

    # ======== Налаштування (внизу) ========
    try:
        settings_img = Image.open("settings.png").resize((24, 24))
        settings_icon = ImageTk.PhotoImage(settings_img)
        settings_btn = tk.Button(menu_frame, image=settings_icon, bg="#2c3e50", bd=0, command=lambda:
dummy_action("Налаштування"))
```

```
        settings_btn.image = settings_icon
        settings_btn.pack(side="bottom", pady=20)
    except:
        tk.Button(
            menu_frame,
            text="⚙ Налаштування",
            font=("Arial", 16),
            bg="#2c3e50",
            fg="white",
            bd=0,
            command=open_settings_window
        ).pack(side="bottom", pady=20)

    show_main_content(content_frame, canvas)
    main_window.mainloop()

    user_info_raw = {
        "ПІБ": user[0],
        "Пошта": user[1],
        "Посада": user[2],
        "Доступ": user[3]
    }

load_settings_from_file()

# Псевдофункція
def dummy_action(name):
    messagebox.showinfo("Дія", f"Функція '{name}' ще не реалізована.")

# Головна сторінка
def show_main_content(parent, canvas):
    for widget in parent.winfo_children():
        widget.destroy()

    greeting = tk.Label(
        parent,
        text="Ласкаво просимо до системи ПмОМ",
        font=("Arial", 28, "bold"),
        bg="#ecf0f1",
        fg="#2c3e50"
    )
    greeting.pack(pady=(80, 20))

    message_frame = tk.Frame(parent, bg="#ecf0f1")
    message_frame.pack(padx=50)

    first_line_frame = tk.Frame(message_frame, bg="#ecf0f1")
    first_line_frame.pack(anchor="w")

    first_line_text = tk.Label(
        first_line_frame,
        text="Дякуємо за вашу службу народові України",
        font=("Arial", 16),
        bg="#ecf0f1",
        fg="#34495e"
```

```
    )
    first_line_text.pack(side="left")

    try:
        flag_img = Image.open("data/img/flag.png")
        flag_img = flag_img.resize((32, 32), Image.Resampling.LANCZOS)
        flag_photo = ImageTk.PhotoImage(flag_img)
        flag_label = tk.Label(first_line_frame, image=flag_photo, bg="#ecf0f1")
        flag_label.image = flag_photo
        flag_label.pack(side="left", padx=(0, 0))
    except Exception as e:
        print("Не вдалося завантажити flag.png:", e)

    additional_text = tk.Label(
        message_frame,
        text=(
            "\nСистема допоможе вам швидко та ефективно керувати\n"
            "звітами та правовими актами, пов'язаними з військовим майном.\n\n"
            "Успішної роботи!"
        ),
        font=("Arial", 16),
        bg="#ecf0f1",
        fg="#34495e",
        justify="left"
    )
    additional_text.pack(anchor="w")

    canvas.yview_moveto(0)

# Підтвердження виходу
def exit_confirmation(root):
    if messagebox.askyesno("Підтвердження", "Ви впевнені, що хочете завершити роботу?"):
        root.destroy()
        sys.exit()

# Вікно логіну
def create_login_window():
    root = tk.Tk()
    set_window_icon(root)
    root.title("Реєстрація - ПмОМ")
    root.geometry("400x300")
    root.resizable(False, False)
    root.configure(bg="#f0f2f5")

    tk.Label(root, text="ПмОМ", font=("Arial", 20, "bold"), bg="#f0f2f5", fg="#2c3e50").pack(pady=10)

    form_frame = tk.Frame(root, bg="#f0f2f5")
    form_frame.pack(pady=10)

    tk.Label(form_frame, text="Електронна пошта або ПІБ:", bg="#f0f2f5").grid(row=0, column=0, sticky="w",
padx=10, pady=5)
    entry_login = ttk.Entry(form_frame, width=35)
    entry_login.grid(row=1, column=0, padx=10, pady=5)

    tk.Label(form_frame, text="Пароль:", bg="#f0f2f5").grid(row=2, column=0, sticky="w", padx=10, pady=5)
```

```
entry_password = ttk.Entry(form_frame, show="*", width=35)
entry_password.grid(row=3, column=0, padx=10, pady=5)

def login_action():
    login = entry_login.get().strip()
    password = entry_password.get().strip()

    status = load_block_status()
    now = time.time()

    # Перевіряємо, чи триває блокування
    if status["block_start"] > 0:
        block_duration = BLOCK_TIMES[min(status["block_level"], len(BLOCK_TIMES) - 1)]
        elapsed = now - status["block_start"]
        if elapsed < block_duration:
            remaining = int(block_duration - elapsed)
            minutes = remaining // 60
            seconds = remaining % 60
            messagebox.showerror(
                "Блокування",
                f"Заблоковано. Спробуйте через {minutes} хв {seconds} сек."
            )
            return
        else:
            # Блокування закінчилося — скидаємо лічильники
            status["fail_count"] = 0
            status["block_start"] = 0
            save_block_status(status["fail_count"], status["block_start"], status["block_level"])

    user = authenticate_user(login, password)
    if user:
        # Успішний вхід — скидаємо всі лічильники
        save_block_status(0, 0, 0)
        # ДОДАТИ ВИКЛИК ПЕРЕВІРКИ ТАБЛИЦІ
        setup_user_info_table_by_login(login)
        setup_user_operation_table_by_login(login)
        root.destroy()
        open_main_window(user)
    else:
        # Невдала спроба
        status["fail_count"] += 1
        if status["fail_count"] >= 3:
            # Активуємо блокування
            status["fail_count"] = 0
            status["block_start"] = now
            status["block_level"] = min(status["block_level"] + 1, len(BLOCK_TIMES) - 1)
            save_block_status(status["fail_count"], status["block_start"], status["block_level"])
            block_duration = BLOCK_TIMES[status["block_level"]]
            minutes = block_duration // 60
            messagebox.showerror(
                "Блокування",
                f"Заблоковано на {minutes} хвилин через 3 невдалі спроби."
            )
        else:
            save_block_status(status["fail_count"], status["block_start"], status["block_level"])
```

```
            remaining_attempts = 3 - status["fail_count"]
            messagebox.showerror(
                "Помилка",
                f"Невірний логін або пароль. Спробуйте ще {remaining_attempts} раз(и)."
            )

    ttk.Button(root, text="Увійти", command=login_action).pack(pady=10)

    tk.Label(
        root,
        text="За реєстрацією чи відновленням пароля\nзверніться до свого командира",
        font=("Arial", 9),
        bg="#f0f2f5",
        fg="#7f8c8d",
        justify="center"
    ).pack(side="bottom", pady=10)

    root.mainloop()

# Запуск
if __name__ == "__main__":
    create_login_window()
```

ДОДАТОК Г

ГРАФІЧНА ЧАСТИНА

«ПРОГРАМНИЙ МОДУЛЬ ОБЛІКУ МАЙНА ВІЙСЬКОВОЇ ЧАСТИНИ»

Виконав: студент 4 курсу, групи 1КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Прудніков Е. Р.
(прізвище та ініціали)

Керівник: к. т. н., доцент каф. КН
Арсенюк І. Р.
(прізвище та ініціали)

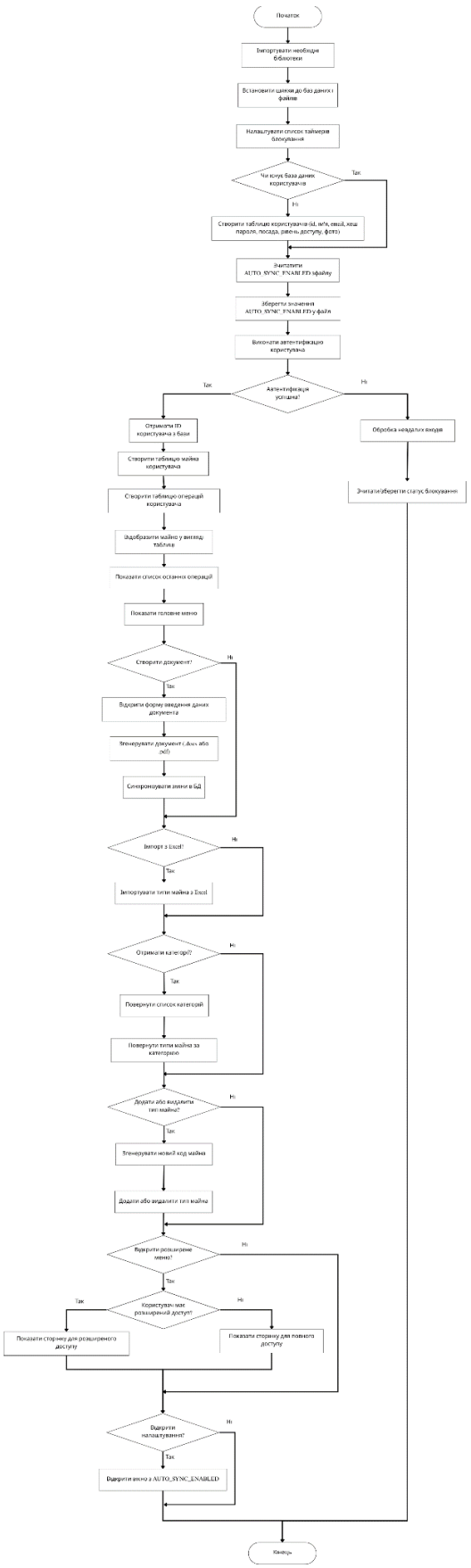

Рисунок Г.1 – Алгоритм функціонування програмного модуля обліку майна військової частини

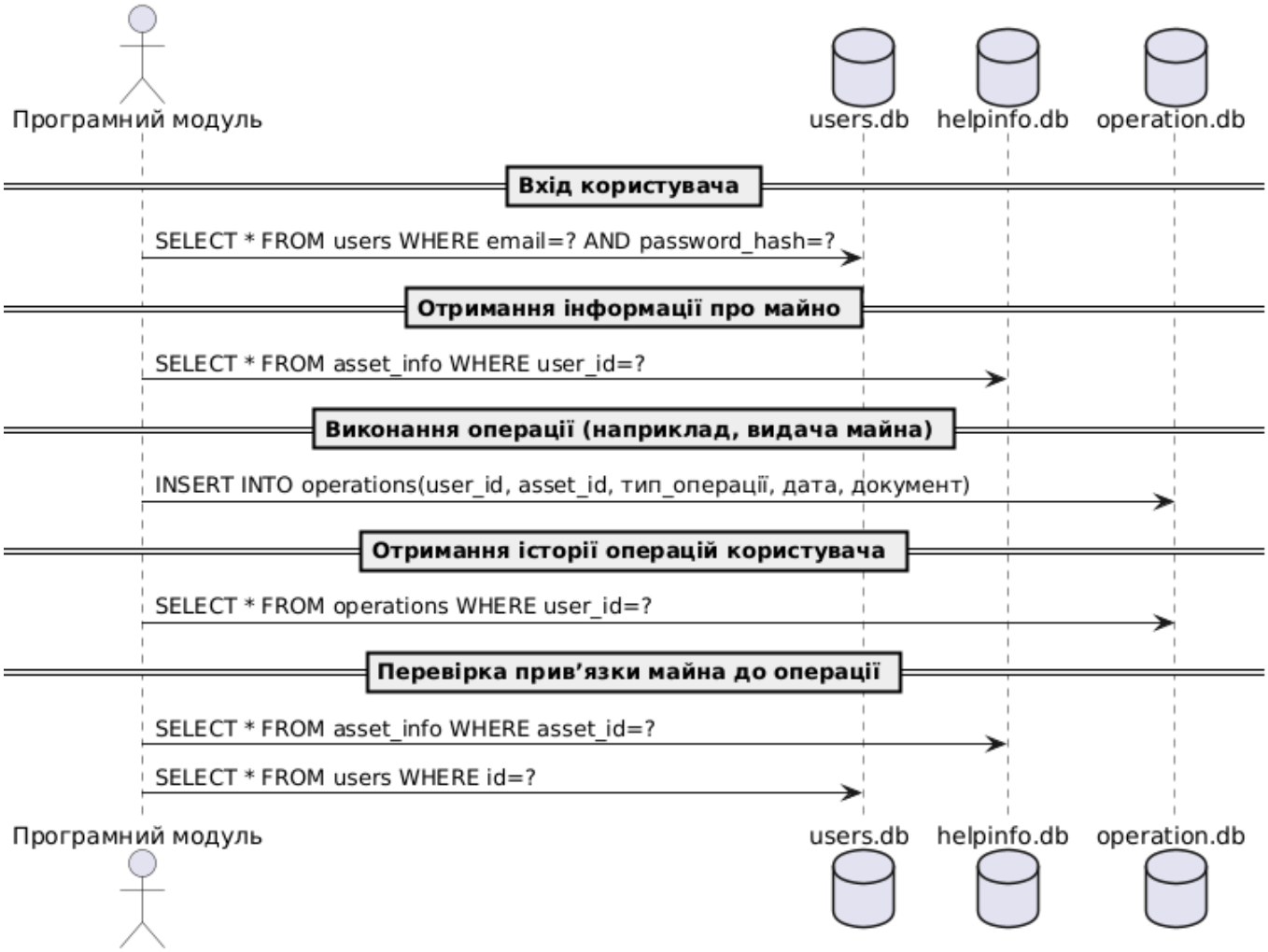

Рисунок Г.2 – Діаграма взаємодії програмного модуля з базами даних

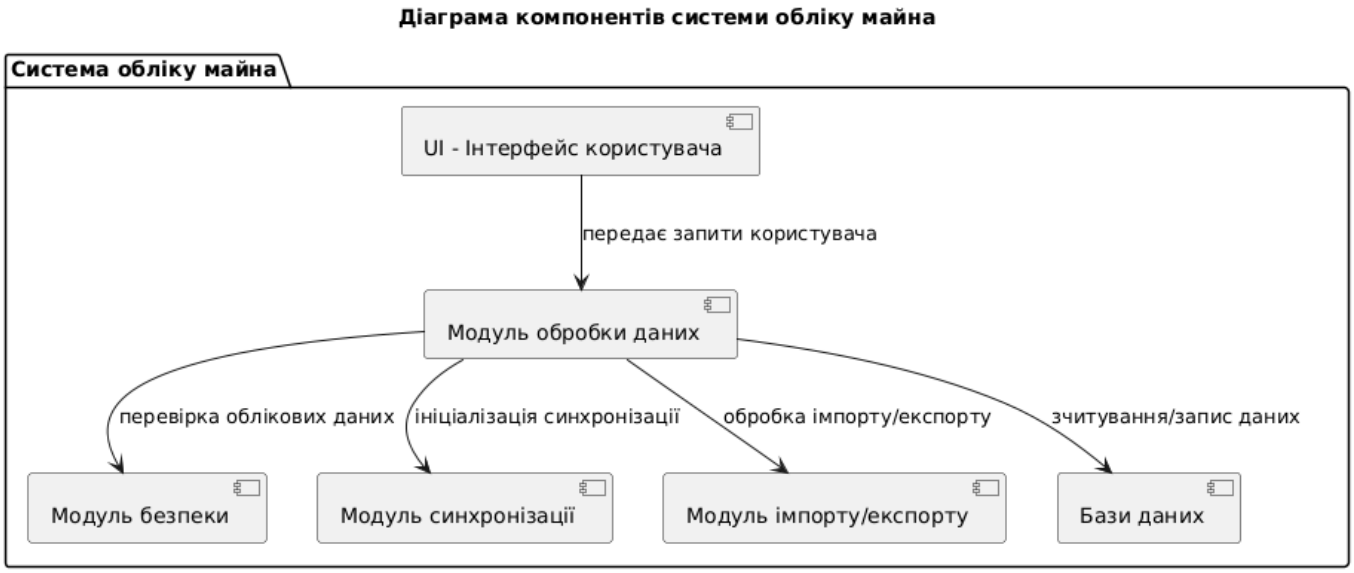

Рисунок Г.3 – Діаграма компонентів програмного модулю

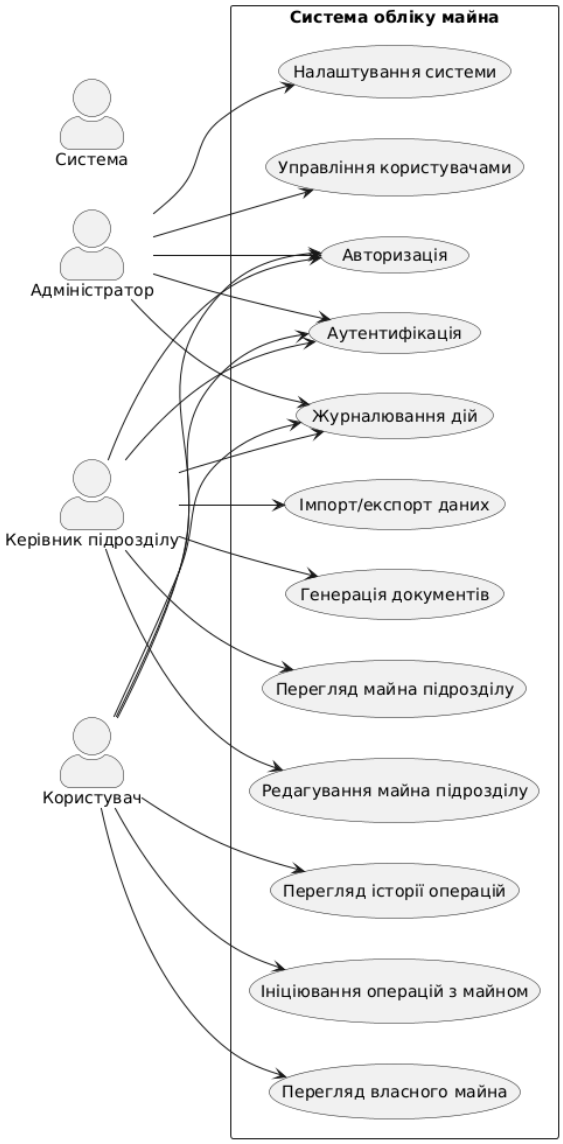

Рисунок Г.4 – Діаграма рівнів користувачів та їх прав

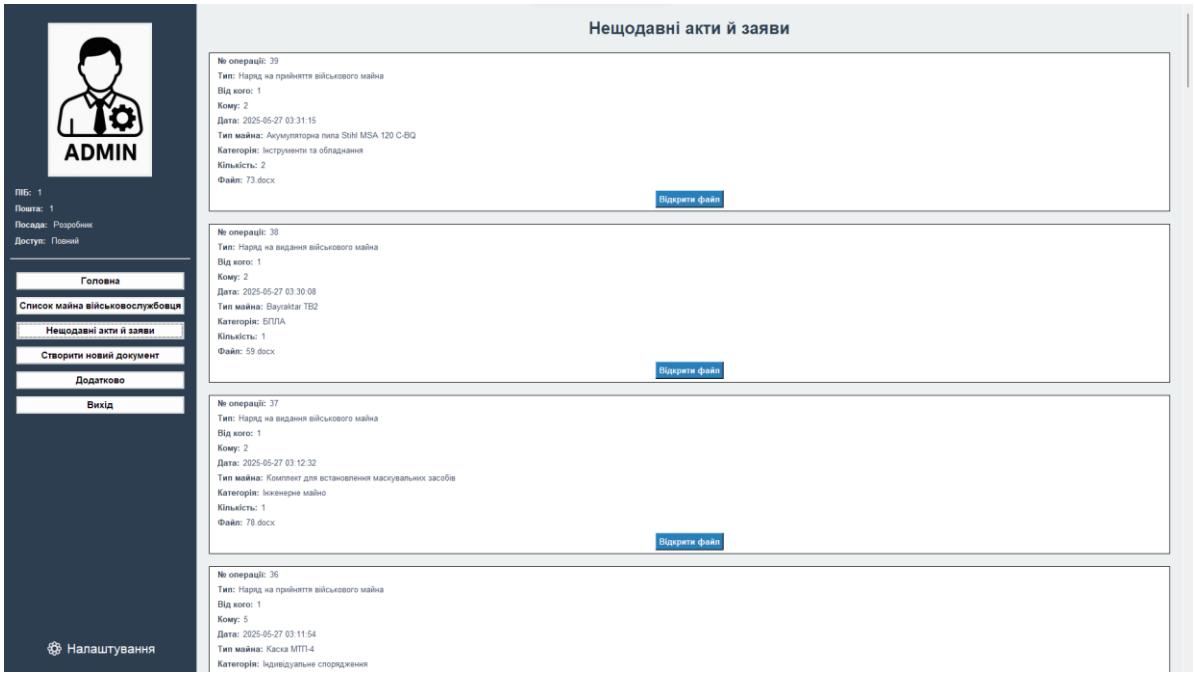

Рисунок Г.5 – Приклад роботи розробленого програмного модуля