Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп’ютерних наук

(повна назва)

Кафедра Програмної інженерії

(повна назва)

**КВАЛІФІКАЦІЙНА РОБОТА**

**Пояснювальна записка**

Рівень вищої освіти \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_перший (бакалаврський)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Програмна система для обліку основних засобів приватного підприємства

.

(тема)

Виконав:

студент 4 курсу, групи ПЗПІ-21-7

Максим ТКАЧЕНКО

(ім’я, прізвище)

Спеціальність 121 – Інженерія програмного

забезпечення

(код і повна назва спеціальності)

Тип програми\_\_\_\_\_освітньо-професійна\_\_ Освітня програма Програмна інженерія

( повна назва освітньої програми)

Керівник доц. Олексій ТУРУТА

(посада, ім’я, прізвище)

Допускається до захисту

|  |  |  |
| --- | --- | --- |
| Зав. кафедри | \_\_\_\_\_\_\_\_\_\_\_\_  (підпис) | Кирило СМЕЛЯКОВ  (ім’я, прізвище) |

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп’ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти Спеціальність

перший (бакалаврський)

121 – Інженерія програмного забезпечення

(код і повна назва)

Тип програми освітньо-професійна

Освітня програма

Програмна інженерія

(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри

(підпис)

« » 2025 р.

**ЗАВДАННЯ**

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Ткаченко Максим Андрійович \_

(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система для обліку основних засобів приватного

підприємства\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

затверджена наказом університету від 19 05 2025р. №\_\_\_\_\_397Ст\_\_\_\_\_

2. Термін подання студентом роботи до екзаменаційної комісії 10 06 2025 р.

3. Вихідні дані до роботи *У межах програмного рішення передбачити розробку сучасної системи обліку основних засобів для приватного \_\_\_ підприємства, що гарантуватиме підвищену точність, оптимізацію процесів і відповідність встановленим нормам і вимогам. Microsoft Visual Studio як середовище розробки, платформу .NET з фреймворком Blazor для створення браузерних додатків та СУБД MS SQL для зберігання і обробки даних.*

4. Перелік питань, що потрібно опрацювати у роботі *Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проєктування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів роботи | Терміни  виконання етапів  роботи | Примітка |
| 1 | Аналіз предметної галузі | 06.04.2025 | *виконано* |
| 2 | Створення специфікації програмного забезпечення | 10.04.2025 | *виконано* |
| 3 | Проєктування програмного забезпечення | 13.04.2025 | *виконано* |
| 4 | Розробка програмного забезпечення | 18.04.2025 | *виконано* |
| 5 | Тестування програмного забезпечення | 02.05.2025 | *виконано* |
| 6 | Оформлення пояснювальної записки | 09.06.2025 | *виконано* |
| 7 | Підготовка доповіді та презентації | 10.06.2025 | *виконано* |
| 8 | Перевірка на нормоконтроль | 06.06.2025 | *виконано* |
| 9 | Попередній захист | 08.06.2025 | *виконано* |
| 10 | Здача роботи у електронний архів | 10.06.2025 | *виконано* |
| 11 | Допуск до захисту у зав. кафедри | 10.06.2025 | *виконано* |

Дата видачі завдання 4 квітня 2025 р.

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Керівник роботи доц. Олексій ТУРУТА\_\_\_\_

(підпис) (посада, ім’я, прізвище)

**РЕФЕРАТ / ABSTRACT**

Пояснювальна записка до кваліфікаційної роботи бакалавра, 52 стор., 5 рис., 11 джерел.

БУХГАЛТЕРІЯ, ОБЛІК ОСНОВНИХ ЗАСОБІВ, ПРИВАТНЕ ПІДПРИЄМСТВО, ПРОГРАМНА СИСТЕМА, BLAZOR, MS SQL, .NET.

Об’єкт розробки – сучасна програмна система обліку основних засобів для приватного підприємства. Вона дозволить оперативно фіксувати придбання, нараховувати амортизацію, реєструвати вибуття та планувати техобслуговування активів. Завдяки цьому підприємство отримає точну картину вартості й стану основних засобів у будь-який момент і зможе обґрунтовано планувати подальші інвестиції. Автоматизація рутинних бухгалтерських операцій мінімізує помилки, пришвидшує обробку даних та вивільнить ресурси для стратегічних завдань бізнесу.

**Мета розробки** – створити надійний інструмент управління та моніторингу основних засобів, що відповідатиме вимогам бухгалтерських стандартів і регуляторних норм, підвищить точність обліку та ефективність процесів.

**Технічний стек** – Microsoft Visual Studio, платформа .NET із Blazor-фреймворком для веб-інтерфейсу та СУБД MS SQL для зберігання й обробки даних.

У підсумку реалізовано функціональну систему, яка забезпечує повний цикл обліку основних засобів від придбання до списання.

ACCOUNTING, BLAZOR, FIXED ASSETS ACCOUNTING, MS SQL, PRIVATE ENTERPRISE, SOFTWARE SYSTEM, .NET.

Development Object – an advanced software system for fixed-assets management tailored to a private enterprise. It enables real-time recording of acquisitions, automated depreciation, disposal tracking, and maintenance planning. This gives the company a clear, up-to-date view of asset value and condition, supporting data-driven investment decisions. Automating routine accounting tasks reduces errors, speeds up processing, and frees up resources for core business activities.

Project Goal – to deliver a robust tool for managing and monitoring fixed assets that fully complies with accounting standards and regulations, enhances accuracy, and streamlines workflows.

Technology Stack – Microsoft Visual Studio as the IDE, the .NET platform with Blazor for browser-based interfaces, and MS SQL for database management.

Result – a comprehensive fixed-assets accounting system covering the entire lifecycle from purchase through disposal.

Я, Ткаченко Максим Андрійович, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система для обліку основних засобів приватного підприємства», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

**ЗМІСТ**

Вступ 7

1 Аналіз предметної області 8

1.1 Аналіз предметної галузі 8

1.2 Виявлення проблем та актуалізація рішень 10

1.3 Постановка задачі 11

2 Формування вимог до програмної системи 13

3 Архітектура та проєктування програмного забезпечення 15

3.1 UML проєктування програмного забезпечення 15

3.2 Проєктування архітектури програмного забезпечення 20

3.3 Проєктування структури зберігання даних 21

3.4 Приклади найцікавіших алгоритмів та методів 24

4 Опис прийнятих програмних рішень 26

5 Тестування розробленого програмного забезпечення 39

6 Впровадження програмного забезпечення 42

Висновки 44

Перелік джерел посилання 46

Додаток А Створений UI / UX дизайн системи 47

Додаток Б Слайди презентації 50

Додаток В Звіт результатів перевірки на унікальність тексту 51

Додаток Г Експертний висновок з нормоконтролю 52

**ВСТУП**

Ведення бухгалтерії – це ключовий елемент будь-якої комерційної діяльності, а основні засоби становлять один із найважливіших активів організації. Коректний моніторинг цих активів сприяє оптимальному управлінню, чіткому плануванню та обґрунтованому прийняттю рішень. В умовах жорсткої конкуренції надважливо мати достовірні й актуальні дані про основні засоби, щоб випереджати конкурентів.

Щоб подолати труднощі ручного обліку, буде створено програмне рішення для автоматизації процесу обліку основних засобів, що гарантуватиме його точність та результативність. Метою проєкту є розробити таку систему для приватного підприємства та провести аналіз її ефективності.

Відстеження основних засобів – критично важливий процес для будь-якого приватного бізнесу. Ручний облік займає багато часу і часто призводить до помилок, від чого страждає якість інформації і зростає ризик невдалих управлінських рішень. Впровадження програмної системи дозволить підприємству оптимізувати облікові операції та зменшити ймовірність помилок.

Ключова задача цієї роботи – створити інтуїтивно зрозумілу, продуктивну та максимально точну систему обліку основних засобів. Крім того, вона повинна вміти формувати звіти, необхідні для оперативних управлінських рішень. Розроблена система може бути інтегрована та протестована на базі приватного підприємства, а результати дослідження міститимуть оцінку того, наскільки вона підвищує точність і ефективність облікових процесів.

**1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ**

1.1 Аналіз предметної галузі

Облік основних засобів є надзвичайно важливою складовою в багатьох секторах економіки, зокрема у виробництві, будівництві, транспорті та енергетиці. У цих сферах такі активи формують помітну частину сумарних ресурсів і мають вирішальне значення для основної господарської діяльності підприємства [1].

Наприклад, у переробних виробництвах машини, устаткування й будівлі відіграють ключову роль у технологічному процесі. Точність їх обліку критична для раціонального завантаження, своєчасного сервісу та дотримання регулятивних норм.

Схожа ситуація спостерігається й у будівельному секторі, де важка техніка й транспорт необхідні, щоб завершувати проєкти вчасно та без перевитрат [2]. Правильне ведення обліку допомагає скоротити простої, оптимізувати витрати на підтримку та підвищити маржу.

У транспортній індустрії активи на кшталт літаків, суден і поїздів забезпечують стабільність перевезень. Коректна реєстрація дає змогу гарантувати безпеку, дотримання нормативів і максимальну продуктивність парку.

В енергетичній галузі труби, електростанції чи бурові платформи слугують критичною інфраструктурою для генерації й постачання енергії [3]. Точний облік тут підтримує безпеку, екологічні стандарти й ефективне використання потужностей.

У підсумку, якісний облік основних засобів істотно впливає на результативність компаній у багатьох галузях. Впровадження спеціалізованих програмних комплексів підвищує точність і швидкість обробки даних, що сприяє кращим управлінським рішенням і більшій прибутковості.

Попри значущість обліку, у практиці зустрічаються кілька типових проблем:

* неточна чи неповна інформація: ручний ввід даних часто супроводжується помилками, що спотворює картину та призводить до нераціонального розподілу ресурсів і фінансових втрат;
* трудомісткість та тривалість процедур: паперові або табличні методики потребують багато часу і людських ресурсів для підтримки актуальності;
* відсутність уніфікації: нестача стандартизованих підходів веде до несумісності або прогалин у даних щодо різних активів;
* ризики безпеки й конфіденційності: записи містять чутливу інформацію (вартість, строк служби, амортизаційний метод), тож виникає потреба надійного захисту від несанкціонованого доступу та шахрайства.

Далі розвиток обліку основних засобів повинен зосереджуватися на створенні програмних систем, що усувають зазначені недоліки. Це означає проектування рішень, які будуть зручними, точними та ефективними у відображенні всіх релевантних відомостей. Також важливо забезпечити інтеграцію з іншими бухгалтерськими модулями, наприклад, із підсистемами дебіторської та кредиторської заборгованості.

Не менш суттєвим є врахування питань безпеки й конфіденційності: системи мають реалізовувати шифрування даних і багаторівневий контроль доступу, орієнтований на ролі та обов’язки працівників.

Подальший прогрес слід спрямувати на розробку гнучких рішень, здатних адаптуватися до галузевих особливостей. Йдеться про широкі можливості кастомізації та взаємодію з іншими галузевими ПЗ, що забезпечить безперервний потік даних.

Підсумовуючи, створення сучасної системи обліку основних засобів може суттєво збільшити точність і продуктивність облікових процедур. У майбутньому необхідно зосередитися на усуненні наявних проблем і формуванні користувацько-орієнтованих, ефективних і безпечних рішень, адаптованих під конкретні потреби кожної окремої галузі.

1.2 Виявлення проблем та актуалізація рішень

Як було наголошено у попередньому пункті, система обліку основних засобів є вирішальною для успішної діяльності підприємств у різних галузях. Проте на практиці організації часто зустрічаються з низкою проблем, здатних спричинити перекручення даних, хибні управлінські рішення та фінансові збитки. У цьому підрозділі розглянуто ключові «больові точки» та окреслено способи їх усунення шляхом упровадження спеціалізованої програмної платформи.

Неточність і фрагментарність відомостей. Одним із головних недоліків ручного ведення обліку є ризик помилок, через які дані стають неповними або суперечливими. Наслідком може бути некоректний розподіл ресурсів та неправильні фінансові висновки. Розв’язання: автоматизована система, що збирає й вводить інформацію, мінімізує людський фактор і забезпечує повноту та узгодженість даних завдяки вбудованим перевіркам.

Тривалі та ресурсомісткі процедури. Паперові або табличні методики потребують значного часу та сил, що затягує звітність і погіршує оперативність управління. Розв’язання: програмна платформа автоматизує рутинні операції, скорочуючи часові витрати; крім того, модуль аналітики дає змогу отримувати звіти в реальному часі й швидко ухвалювати рішення.

Брак уніфікованих підходів. Відсутність єдиних стандартів веде до розбіжностей або прогалин у даних щодо різних активів, ускладнюючи їх порівняння. Розв’язання: впровадження єдиної системи з типізованими полями та валідаторами гарантує, що вся інформація фіксується послідовно й однаково для кожного об’єкта.

Загрози безпеці й конфіденційності. Відомості про вартість, строк експлуатації та спосіб амортизації мають бути захищені від стороннього доступу та шахрайства. Розв’язання: платформа повинна підтримувати шифрування, багаторівневий контроль доступу за ролями та регулярне резервне копіювання даних для їх збереження і відновлення.

Отже, створення цілісної програмної системи обліку основних засобів дає змогу ліквідувати вказані недоліки: автоматизувати процеси, підвищити точність і повноту відомостей, забезпечити миттєву звітність, а також зміцнити захист та конфіденційність фінансових даних.

1.3 Постановка задачі

Завдання полягає у створенні програмної платформи для обліку основних засобів, що розв’язує питання, описані в попередніх підрозділах. Розробка спрямована на підвищення точності й продуктивності обліку, уніфікацію даних, посилення захисту та конфіденційності, а також формування звітності у режимі реального часу. Нижче наведено головні орієнтири системи

* автоматизація бухгалтерських процедур: майбутня система спростить облік основних засобів, скоротивши витрати часу й ресурсів на підтримку документації; при цьому гарантовано коректний та повний збір необхідних відомостей, що мінімізує ризик неточностей;
* стандартизація відомостей: рішення забезпечить єдиний порядок фіксації даних про основні засоби. Передбачено типові поля та перевірки, аби інформація збиралась правильно й послідовно, що полегшить порівняння активів і сприятиме обґрунтованим висновкам;
* зростання точності та повноти: система міститиме вбудовані контролі достовірності, які гарантуватимуть узгодженість і правильність введених даних, відтак зменшаться ризики похибок та розбіжностей;
* звітність у режимі online: платформа формуватиме поточні звіти, дозволяючи оперативно ухвалювати рішення. Користувачі отримуватимуть актуальні відомості про стан активів і зможуть раціонально розподіляти ресурси;
* посилення безпеки й конфіденційності: програмний комплекс включатиме потужні механізми захисту, що охоронятимуть чутливі відомості (вартість, строки, методи амортизації). Реалізовано шифрування, рольовий доступ і регулярні резервні копії, аби забезпечити збереження й відновлення інформації

Для реалізації цих цілей система міститиме такі можливості:

* + автоматизований ввід і збір відомостей;
  + заздалегідь визначені поля та перевірки даних;
  + аналітику й звітність у реальному часі;
  + систему прав доступу й дозволів;
  + захищене зберігання та резервування інформації.

Отже, впровадження такої програмної платформи істотно підвищить точність та результативність облікових процедур, уніфікує дані, поліпшить захист та конфіденційність, а також надаватиме звіти online. Система автоматизує облік, знизить імовірність помилок і забезпечить користувачів актуальною інформацією щодо стану активів.

**2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ**

Вимоги до програмної системи обліку основних засобів мають ґрунтуватися на цілях і особливостях, викладених у попередньому розділі. Тому під час формування технічного завдання необхідно врахувати такі пункти:

* дружній інтерфейс користувача: програмний комплекс зобов’язаний мати інтуїтивно зрозумілий і ергономічний UI, що дозволятиме співробітникам без зусиль орієнтуватися у меню та виконувати базові дії, зокрема додавання, коригування й видалення об’єктів обліку;
* автоматизований імпорт даних: система повинна вміти автоматично підхоплювати інформацію з первинних документів — таких, як замовлення на поставку, рахунки-фактури чи касові квитанції, — що мінімізує ризик помилок і скорочує час ручного введення;
* попередньо задані реквізити: ПЗ має містити набір стандартних полів (тип активу, первісна вартість, очікуваний строк служби, метод амортизації тощо), аби забезпечити однорідний і систематизований збір критично важливих відомостей по всіх позиціях;
* перевірка коректності вводів: модуль валідації даних повинен відстежувати точність і логічну цілісність інформації, істотно зменшуючи ймовірність розбіжностей або неузгодженостей;
* звітність та аналітика «на льоту»: програмне середовище зобов’язане генерувати оперативні звіти й інтерактивні аналітичні панелі у реальному часі, що дає змогу менеджерам своєчасно приймати обґрунтовані рішення стосовно розподілу ресурсів і керування активами;
* рольовий доступ: система має підтримувати гнучкий механізм авторизації, базований на ролях та посадових обов’язках, щоб кожен працівник бачив тільки ті дані, які йому дозволені;
* захищене сховище: усі відомості слід зберігати у надійному середовищі зі шифруванням конфіденційної інформації, багатоступеневим контролем доступу та регламентованим резервним копіюванням для гарантії відновлення;
* масштабованість: архітектура повинна легко підлаштовуватися під зростання бази активів і кількості користувачів у разі розширення бізнесу;
* інтеграційні можливості: необхідна здатність до взаємодії з іншими обліковими модулями, зокрема підсистемами щодо кредиторської й дебіторської заборгованості, що мінімізує дублювання інформації і забезпечує цілісний обмін даними;
* кастомізація: програмний продукт має дозволяти підлаштування під специфіку конкретного підприємства, включаючи додавання унікальних полів, створення індивідуальних звітів і конфігурацію власних бізнес-процесів.

Отже, визначені вимоги — інтуїтивний інтерфейс, автоматичний збір інформації, типовий набір реквізитів, контроль достовірності, миттєва аналітика, гнучкий доступ, захищене зберігання, масштабованість, інтеграція та широкі опції кастомізації — у комплексі забезпечують, що програмна система буде ефективною, продуктивною й максимально відповідатиме унікальним потребам бізнес-середовища.

**3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ**

3.1 UML проєктування ПЗ

Уніфікована мова моделювання (UML) є широко вживаним засобом конструювання програмного забезпечення, що пропонує графічну нотацію для опису, візуалізації та фіксації характеристик програмних рішень. UML застосовують для відтворення систем на різних щаблях абстракції — від оглядових концептуальних схем до детально прописаних моделей реалізації [4].

Діаграма варіантів використання подає функції програмної платформи очима кінцевого користувача [5]. На ній відображено сукупність сценаріїв, які підтримує система, а також акторів, що взаємодіють із нею (див. рис. 3.1).

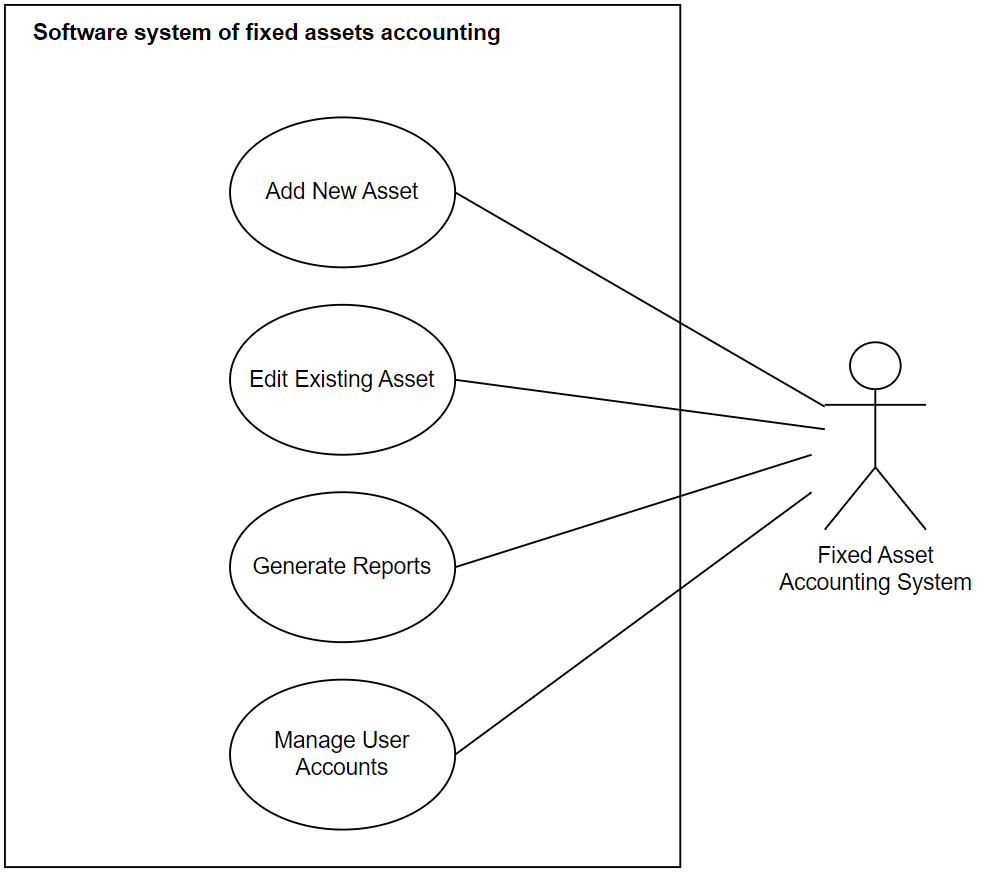


Рисунок 3.1 – Діаграма варіантів використання

У центрі наведеної діаграми розташовано систему обліку основних засобів. Виділено чотири головні сценарії: додавання нового активу, редагування вже наявного, формування звітів та адміністрування облікових записів. Сценарій «Додати новий актив» відображає процедуру внесення в систему свіжої одиниці майна, тоді як «Редагувати існуючий актив» ілюструє оновлення відомостей щодо вже зареєстрованого об’єкта. Сценарій «Створити звіти» охоплює формування вихідних документів (наприклад, зведення по амортизації чи інвентаризаційні відомості). Нарешті, «Керувати обліковими записами» демонструє операції з користувачами — підвищення/зниження прав доступу, створення чи видалення профілів. Отже, ця діаграма дає цілісне уявлення про функціонал системи та її використання зацікавленими сторонами.

Діаграма класів описує логічну структуру даних програмного продукту [6]. Вона відтворює класи, їх атрибути й методи, що формують основу рішення (див. рис. 3.2).

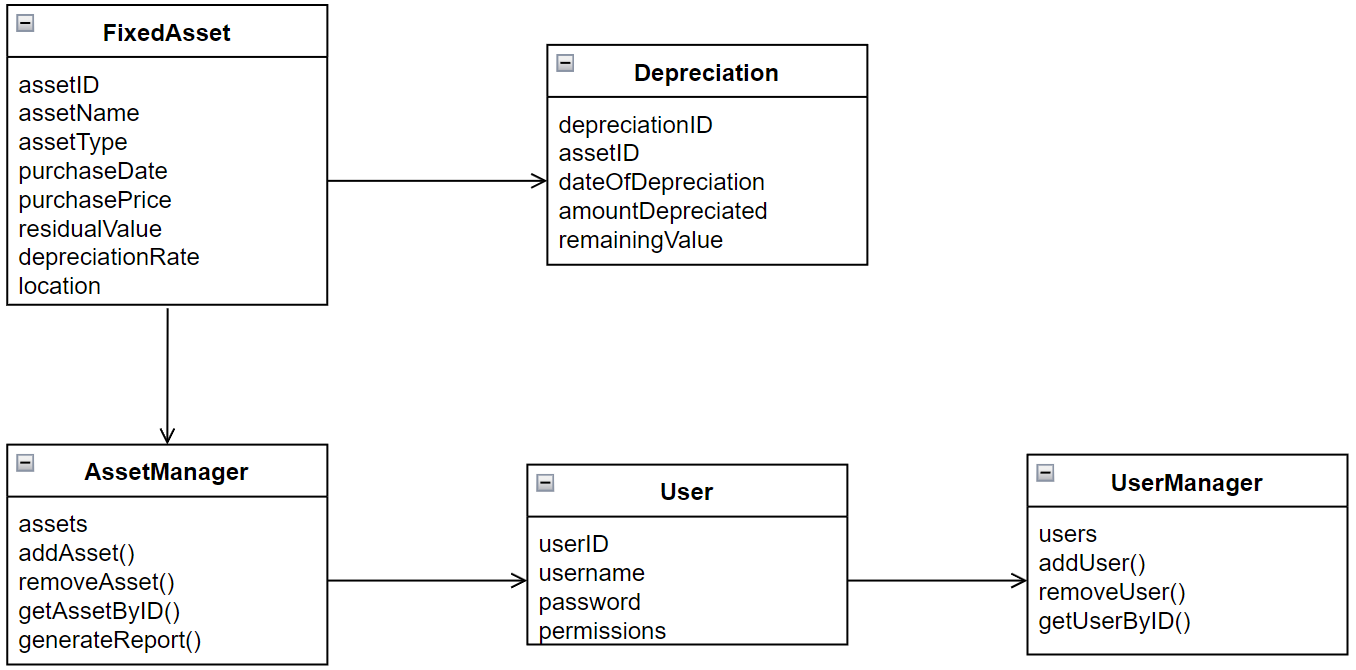


Рисунок 3.2 – Діаграма класів

У представленій схемі клас FixedAsset символізує матеріальні активи, що перебувають під обліком. До його складу входять поля assetID, assetName, assetType, purchaseDate, purchasePrice, residualValue, depreciationRate і location. Клас «Амортизація» пов’язаний із FixedAsset співвідношенням «один до багатьох», пояснюючи, що кожен ресурс може мати декілька записів про нараховану амортизацію. Depreciation зберігає такі атрибути, як depreciationID, assetID, dateOfDepreciation, amountDepreciated та remainingValue. Клас AssetManager реалізує бізнес-логіку керування активами, утримуючи колекцію assets та методи для додавання, вилучення, пошуку об’єктів і створення звітів. Клас User відображає облікові записи персоналу й містить userID, userName, password та permissions. Клас UserManager відповідає за операції адміністрування користувачів — додавання, видалення, пошук. У цілому ця діаграма глибоко розкриває взаємозв’язки між сутностями системи обліку основних засобів.

Діаграма послідовності демонструє, як різні об’єкти чи модулі взаємодіють усередині системи [7]. Вона відображає хронологію повідомлень, що передаються між компонентами під час конкретного сценарію (див. рис. 3.3).

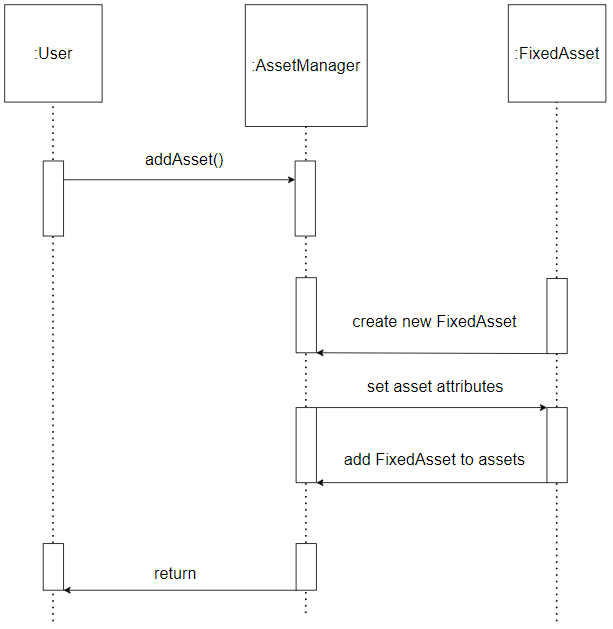


Рисунок 3.3 – Діаграма послідовності

На ній проілюстровано ланцюг викликів між учасниками під час додавання чергового активу. Діаграма деталізує комунікацію між користувачем, AssetManager та екземплярами FixedAsset, слугуючи засобом для глибшого розуміння і вдосконалення цієї функції системи.

Діаграма станів фіксує можливі стани об’єкта протягом його життєвого циклу, а також події, що спонукають переходи між ними (див. рис. 3.4).

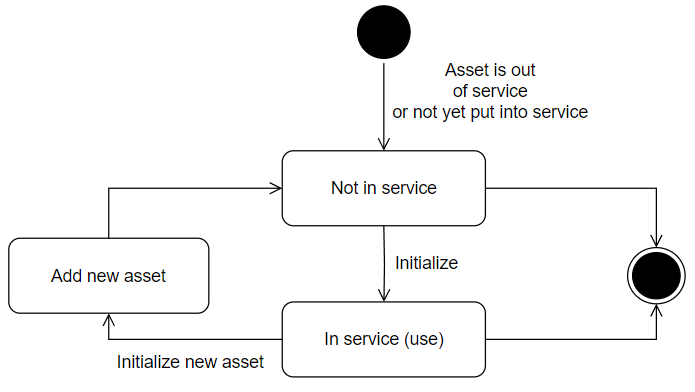


Рисунок 3.4 – Діаграма станів для об'єкта FixedAsset

У наведеній моделі елемент FixedAsset може перебувати у двох ключових станах: «Не в експлуатації» та «В експлуатації (використанні)». Після створення актив стартує у статусі «Не в експлуатації». Коли ж ресурс починають задіювати, він переходить до стану «В експлуатації». За будь-якої потреби актив ініціалізують заново або додають новий, унаслідок чого стан повертається до «Не в експлуатації». Дана схема дає узагальнений огляд потенційних станів об’єкта та шляхів переходу між ними.

Ця діаграма станів надає високорівневий огляд можливих станів, в яких може перебувати об'єкт FixedAsset, а також переходів між цими станами.

Діаграма діяльності відтворює робочий процес окремого сценарію в межах системи: вона описує послідовність кроків і взаємний порядок їх виконання (див. рис. 3.5).

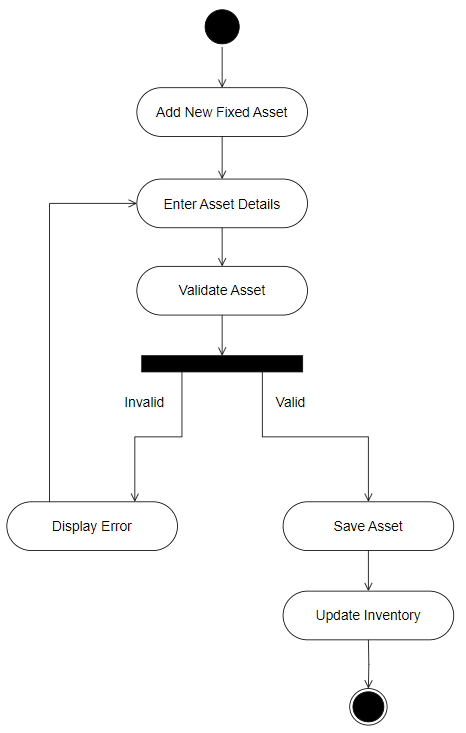


Рисунок 3.5 – Діаграма діяльності для процесу додавання нового основного засобу

Схема розбиває процес внесення нового активу на низку етапів. Користувач спочатку вводить назву, опис і дату закупівлі. Далі система перевіряє коректність і повноту відомостей. Якщо знайдено помилки, користувач отримує повідомлення та повертається на попередній крок. У разі валідних даних актив зберігається в базі, а реєстр основних засобів оновлюється.

Ця діаграма діяльності дає наочне уявлення про кроки і потік контролю під час додавання ресурсу до системи.

Отже, UML є потужним засобом для конструювання систем обліку основних засобів. Вона надає спільну мову й набори нотацій, що допомагають розробникам, дизайнерам і зацікавленим сторонам ефективно спілкуватися та співпрацювати. Застосування UML сприяє тому, аби програмне рішення повністю відповідало поставленим вимогам і технічним специфікаціям.

3.2 Проєктування архітектури ПЗ

Побудова архітектури інформаційної системи для обліку основних фондів і бухгалтерії приватного підприємства із задіянням платформи .NET розглядається як сучасний та дієвий спосіб створення такого програмного комплексу. Сама платформа .NET являє собою фреймворк, придатний для розробки застосунків під браузерну технологію Blazor і базу даних MS SQL, що робить її бездоганним варіантом для реалізації системи контролю основних фондів і фінансів приватного бізнесу [8].

Стартовим етапом у побудові архітектури системи обліку основних засобів і бухгалтерії компанії на основі .NET виступає фіксація вимог. Тут визначаються необхідні функції, зокрема можливість адміністрування та моніторингу активів, генерація звітів і взаємодія з сусідніми фінансовими модулями. Перелік вимог також охоплює продуктивність рішення, його масштабованість та заходи безпеки.  
Коли вимоги закріплено, наступним кроком стає підбір релевантних технологій і інструментарію. Платформа .NET вважається відмінною основою завдяки надійним можливостям, високій швидкодії та потенціалу масштабування. Blazor – актуальний веб-фреймворк, який дозволяє інженерам будувати інтерактивні клієнтські програми мовою C# та .NET [9]; саме тому він підходить для створення зручного інтерфейсу користувача. MS SQL – потужна стабільна система керування базами, що пропонує увесь набір функцій для збереження та опрацювання даних [10].

Загальна архітектура проєктується як тришарова. Презентаційний шар відповідає за користувацький інтерфейс і відтворення даних. Бізнес-шар займається логічною обробкою та виконанням службових операцій. Шар доступу до даних забезпечує запис і вибірку інформації з бази [11].

Застосовуючи Blazor, клієнтську частину реалізують у C# та .NET. Інтерфейс формується таким, щоб він був дружнім, інтуїтивним і адаптивним. Логічний шар також пишеться мовою C#, у ньому сконцентровано алгоритми керування та інвентаризації активів. Доступ до даних реалізується через MS SQL, що гарантує ефективне та безпечне збереження і вибірку записів.

Платформа .NET пропонує низку плюсів, серед яких захищене середовище розробки, висока продуктивність, гнучке масштабування й спрощена інтеграція з зовнішніми рішеннями, зокрема фінансовими та обліковими системами, через API чи веб-сервіси.

Окрім загального проєктного вигляду, рішення на .NET отримує детальне модульне опрацювання. Сюди належить схема окремих компонентів: представницький модуль, бізнес-логіка й структура бази.

Додатково планується впровадити механізм багаторівневого логування, що відстежує як системні події, так і бізнес-операції, забезпечуючи прозорість аудиту. Для підвищення безвідмовності буде задіяно резервне копіювання за розкладом і реплікацію бази даних на окремий сервер. Передбачено підтримку ролей користувачів з деталізованими дозволами, а також інтеграцію з поштовим сервісом для автоматичних повідомлень про критичні події. Веб-інтерфейс отримає можливість перемикання мов, що спростить масове впровадження. Крім того, за допомогою SignalR буде реалізовано миттєве оновлення даних на клієнтських сторінках, що дозволить персоналу бачити зміни у режимі реального часу. Це робить систему по-справжньому гнучкою та дійсно клієнтоорієнтованою.

3.3 Проєктування структури зберігання даних

У програмній системі обліковування базових фондів та фінансового обліку приватної компанії надзвичайно суттєвим є проєктування структури сховування відомостей. Microsoft SQL – це масово застосовувана система керування базами, яка забезпечує значну продуктивність, захищеність, масштабованість і надійність.

Проєктування структури сховування відомостей у програмній системі забезпечує результативне, достовірне та надійне утримування відомостей. Першим кроком у проєктуванні структури сховування відомостей є визначення об'єктів відомостей, які необхідно утримувати. У випадку обліковування базових фондів та фінансового обліку приватної компанії об'єктами відомостей є активи, операції, співробітники, замовники, постачальники та грошова звітність. Кожна сутність відомостей ретельно проаналізована для того, щоб визначити атрибути, які необхідно утримувати. База відомостей для програмної системи складається з декількох таблиць, в яких зберігаються відомості, пов’язані з активами, транзакціями, співробітниками, замовниками, постачальниками та грошовою звітністю. Кожна таблиця має стовпець первинного ключа для однозначної ідентифікації кожного рядка відомостей. Стовпець первинного ключа, як правило, є цілим числом з автоінкрементом.

Схема бази відомостей включає наступні таблиці:

* + таблиця активів: ця таблиця містить відомості, пов’язані з базовими фондами, такі як ідентифікатор активу, назва активу, дата придбання, вартість, термін корисного використання, метод амортизації та ліквідаційна вартість;
  + таблиця операцій: ця таблиця містить відомості, пов’язані з транзакціями, такі як ідентифікатор транзакції, дата транзакції, тип транзакції, ідентифікатор активу та сума транзакції;
  + таблиця співробітників: ця таблиця містить відомості, пов’язані зі співробітниками, такі як ідентифікатор співробітника, ім'я співробітника, контактна інформація та назва посади;
  + таблиця замовників: ця таблиця містить відомості, пов’язані з замовниками, такі як ідентифікатор замовника, ім'я замовника та контактна інформація;
  + таблиця постачальників: ця таблиця містить відомості про постачальників, такі як ідентифікатор постачальника, ім'я постачальника та контактна інформація;
  + таблиця грошового звіту: ця таблиця містить відомості, пов’язані з грошовими звітами, такі як ідентифікатор звіту, дата звіту та тип звіту.

Зовнішні ключі використовуються для встановлення зв'язків між таблицями. Наприклад, стовпчик ідентифікатора активу в таблиці транзакцій є зовнішнім ключем, який посилається на стовпчик ідентифікатора активу в таблиці активів. Такий зв’язок гарантує, що кожна транзакція буде пов’язана з певним активом. Індекси створені для покращення продуктивності бази відомостей. Індекси використовуються для прискорення запитів, дозволяючи базі відомостей швидко знаходити відомості, які запитуються. Цього можна досягти за допомогою первинних ключів, зовнішніх ключів та обмежень. Первинні ключі використовуються для унікальної ідентифікації кожного рядка в таблиці, тоді як зовнішні ключі використовуються для встановлення зв'язків між таблицями.

Обмеження використовуються для застосування правил до відомостей, щоб забезпечити їхню достовірність та узгодженість. На додаток до схеми бази відомостей були створені індекси для підвищення продуктивності програмної системи. Індекси використовуються для прискорення запитів, дозволяючи базі відомостей швидко знаходити відомості, які запитуються. Однак створення занадто великої кількості індексів може сповільнити продуктивність бази відомостей, тому їх слід ретельно розробляти та оптимізувати. При проєктуванні структури сховування відомостей слід також враховувати вимоги до безпеки та резервного копіювання програмної системи.

Слід запровадити механізми контролю доступу, щоб гарантувати, що тільки авторизовані користувачі можуть мати доступ до відомостей і змінювати їх. Процедури резервного копіювання та відновлення повинні бути встановлені для забезпечення захисту відомостей від втрати або пошкодження.

Отже, проєктування структури сховування відомостей є невід’ємною частиною програмної системи обліковування базових фондів та фінансового обліку приватної компанії. Використання Microsoft SQL як системи керування базами відомостей забезпечує надійну і масштабовану платформу для зберігання і оброблення відомостей. Завдяки ретельному проєктуванню схеми бази, індексів та механізмів захисту, програмна система може гарантувати точність, узгодженість та захищеність відомостей.

3.4 Створення UI / UX дизайну системи

UI/UX оформлення програмного комплексу для реєстру основних фондів і бухгалтерії приватної фірми є ключовим чинником, який слід враховувати, аби гарантувати безперебійну взаємодію користувачів. Грамотно спроєктований інтерфейс здатен значно збільшити ефективність, скоротити помилки і поліпшити сукупний досвід аудиторії. Інтерфейсний дизайн програми залишається простим, інтуїтивним, а також комфортним у навігації для категорій. Дизайн UI сплановано з огляду на запити кінцевих користувачів, часто саме технічно недосвідчених. Якісний користувацький інтерфейс полегшує людям швидкий і результативний доступ саме до потрібних даних системи. У застосунку реалізовано ясну, стислу ієрархію меню, добре марковані іконки й кнопки, а також уніфікований макет по всьому середовищу.

Окрім оформлення UI, не менш суттєвим компонентом вважається проєктування користувацького досвіду, тобто саме UX. UX-проєктування охоплює повний спектр вражень людини, у тому числі спосіб її безпосередньо взаємодії з програмою, реакцію програми на дії оператора, а також емоційне відчуття користувача під час експлуатації рішення.

Мета UX-дизайну — сформувати приємні враження клієнта, що зрештою підвищує рівень залучення, задоволення і продуктивність кінцевої аудиторії. Щоб гарантувати ефективність UI/UX, розробники активно здійснили юзабіліті-тестування та постійно збирали фідбек саме користувачів упродовж усього періоду проєктування. Такий підхід дозволив виявити проблемні точки й області покращень, що потім закладалися у дизайн задля формування нижчого користувацького дискомфорту. Щодо візуальних елементів, застосунок використовує охайний сучасний стиль із стриманою палітрою кольорів, а також цілком зрозумілою типографікою. Біле поле потрібно оптимізувати, аби запобігти перевантаженню та допомогти саме користувачам фокусуватися на ключових частинах інтерфейсу. Крім того, програма володіє адаптивністю, отже підлаштовується під різні габарити дисплеїв та пристроїв. Це особливо критично у нинішньому середовищі, де домінують портативні девайси, і користувачі часто заходять у систему зі своїх смартфонів чи саме планшетів. Готовий макет UI/UX системи продемонстровано саме у Додатку А.

Таким чином, UI/UX оформлення програми обліку активів залишається простим, інтуїтивно зрозумілим і комфортним для переміщення, роблячи особливий наголос на формуванні позитивного досвіду користувачів у процесі щоденної взаємодії системою. Тестування разом із фідбеком від користувачів необхідно було включати у процес розробки, а дизайн залишено гнучким для коригування під саме різні екрани та гаджети.

**4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ**

Програмний комплекс для обліку основних фондів і бухгалтерії приватної компанії реалізовано на базі платформи .NET, з інтерфейсною частиною на Blazor і серверною частиною на MS SQL.

.NET — потужне середовище, що дає змогу формувати надійні й масштабовані прикладні рішення під різні ОС: Windows, Linux і macOS. Воно забезпечує великий набір інструментів і фреймворків задля створення високопродуктивних, захищених і стабільних застосунків.

Blazor — веб-фреймворк, який дозволяє реалізувати інтерактивні та динамічні браузерні програми мовою C# у межах .NET. Завдяки йому веб-додатки функціонують без окремих JavaScript-бібліотек чи плагінів.

MS SQL — реляційна СУБД, що дає змогу ефективно зберігати й опрацьовувати великі обсяги структурованих записів. Вона забезпечує багатий набір засобів для інтеграції, аналітики й формування звітності.

Застосувавши згадані технології, розробники створили систему, котра надає зручний інтерактивний інтерфейс для керування бізнес-процесами приватного підприємства, зокрема для адміністрування основних фондів.

Графічний інтерфейс побудовано за допомогою Blazor, що відкриває можливість формувати динамічні веб-сторінки мовою C#. UI спроєктовано за принципами responsive-дизайну, аби гарантувати доступність і легкість роботи на десктопах, ноутбуках, планшетах і смартфонах.

Серверний модуль базується на MS SQL, що забезпечує стійку й масштабовану платформу для збереження й обробки значних масивів даних. Схема бази спроєктована так, щоб підтримувати нормалізацію та логічне структурування записів для оперативного й точного виконання запитів.

Задіяно резервні копії, журнали транзакцій і реплікацію для мінімізації ризику втрат. З'єднання шифруються TLS, доступ контролюють ролі AD чи локального провайдера. Додатково система через API е-звітності формує й подає регламентовані документи просто з інтерфейсу у кілька кліків користувача лише.

**public RegisterModel(**

**UserManager userManager,**

**IUserStore<User> userStore,**

**SignInManager signInManager,**

**ILogger<RegisterModel> logger){**

**\_userManager = userManager;**

**\_userStore = userStore;**

**\_emailStore = GetEmailStore();**

**\_signInManager = signInManager;**

**\_logger = logger;}**

**[BindProperty]**

**public InputModel Input { get; set; }**

**public string ReturnUrl { get; set; }**

**public IList<AuthenticationScheme> ExternalLogins { get; set; } public class InputModel{**

**[Required] [EmailAddress][Display(Name = "Email")]**

**public string Email { get; set; } [Required]**

**[StringLength(100, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.", MinimumLength = 6)]**

**[DataType(DataType.Password)]**

**[Display(Name = "Password")]**

**public string Password { get; set; }**

**[DataType(DataType.Password)]**

**[Display(Name = "Password")]**

**public string Password { get; set; }**

**[DataType(DataType.Password)]**

**[Display(Name = "Confirm password")]**

**[Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]**

**public string ConfirmPassword { get; set; }}**

Код розпочинається з підключення потрібних просторів імен, зокрема Managers та Entities із проєктів Accounting.BLL і Accounting.DAL. Далі оголошується клас RegisterModel, що наслідує PageModel. Усередині класу визначено приватні поля SignInManager, UserManager, IUserStore, IUserEmailStore та ILogger; ці об’єкти відповідають за автентифікацію, авторизацію й роботу з репозиторієм облікових записів. Вкладений клас InputModel описує поля реєстраційної форми (адреса електронної пошти, пароль і підтвердження пароля), а також містить атрибути перевірки Required, EmailAddress і StringLength.

Метод OnGetAsync спрацьовує, коли користувач відкриває сторінку реєстрації через запит GET: він задає ReturnUrl та отримує перелік зовнішніх схем входу, підтримуваних застосунком. Метод OnPostAsync активується під час надсилання форми POST. Спершу перевіряється ModelState, потім створюється новий екземпляр User і заповнюється введеними даними. Далі UserManager.CreateAsync додає користувача, а IUserStore.SetUserNameAsync та IUserEmailStore.SetEmailAsync зберігають логін і адресу. Якщо створення успішне, UserManager.AddToRoleAsync долучає користувача до ролі «Користувач», SignInManager.SignInAsync виконує вхід, і клієнт перенаправляється на головну сторінку; помилки фіксуються у ModelState.

Функція CreateUser створює новий об’єкт User за допомогою Activator.CreateInstance, а якщо об’єкт не формується, генерується виняток. Метод GetEmailStore повертає реалізацію IUserEmailStore, що керує полем e-mail у сховищі. У сукупності цей код забезпечує повний цикл реєстрації в ASP.NET Core і гарантує безпечне збереження облікових даних.

Нижче подано опис Razor-Page, яка обробляє автентифікацію користувачів з використанням механізмів ASP.NET Core Identity.

Далі визначаю модель Razor Page для обробки автентифікації користувачів за допомогою ASP.NET Core Identity:

**public class LoginModel : PageModel{**

**private readonly SignInManager \_signInManager;**

**private readonly ILogger<LoginModel> \_logger;**

**public class LoginModel : PageModel{**

**ILogger<LoginModel> logger){**

**\_signInManager = signInManager;**

**\_logger = logger;}**

**[BindProperty]**

**public InputModel Input { get; set; }**

**public IList<AuthenticationScheme> ExternalLogins { get; set; }**

**public string ReturnUrl { get; set; }**

Продовження коду:

**[TempData]**

**public string ErrorMessage { get; set; }**

**public class InputModel{**

**[Required]**

**[EmailAddress]**

**public string Email { get; set; }**

**[Required]**

**[DataType(DataType.Password)]**

**public string Password { get; set; }**

**[Display(Name = "Запам'ятати мене?")]**

**public bool RememberMe { get; set; } }**

**public async Task OnGetAsync(string returnUrl = null){**

**if (!string.IsNullOrEmpty(ErrorMessage)){**

**ModelState.AddModelError(string.Empty, ErrorMessage);}**

**returnUrl ??= Url.Content("~/");**

**await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);**

**ExternalLogins = (await \_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();**

**ReturnUrl = returnUrl;}**

**public async Task<IActionResult> OnPostAsync(string returnUrl = null){**

**returnUrl ??= Url.Content("~/");**

**ExternalLogins = (await \_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();**

**if (ModelState.IsValid){**

**var result = await \_signInManager.PasswordSignInAsync(Input.Email, Input.Password, Input.RememberMe, lockoutOnFailure: false);**

**if (result.Succeeded){**

**\_logger.LogInformation("User logged in.");**

**return LocalRedirect(returnUrl);}**

**if (result.RequiresTwoFactor){**

**return RedirectToPage("./LoginWith2fa", new { ReturnUrl = returnUrl, RememberMe = Input.RememberMe });}**

**if (result.IsLockedOut){**

**\_logger.LogWarning("User account locked out.");**

Продовження коду:

**return RedirectToPage("./Lockout");}**

**else{**

**ModelState.AddModelError(string.Empty, "Недійсна спроба входу.");**

**return Page();}}**

**return Page();}}**

У фрагменті коду задіяні об’єкти SignInManager і ILogger з просторів Microsoft.AspNetCore.Identity та Microsoft.Extensions.Logging. Клас LoginModel успадковує PageModel, що слугує основою для Razor-сторінок в ASP.NET Core.

Конструктор LoginModel приймає примірники SignInManager й ILogger: перший керує сесіями входу, другий занотовує події у логи.

Компонент містить декілька публічних властивостей:

* + BindProperty, зв’язану з полями, які заповнює користувач (e-mail, пароль, прапорець «запам’ятати мене») – Input;
  + збірку зовнішніх механізмів автентифікації – ExternalLogins;  
    − URL, на який система переадресує клієнта після успішної авторизації – ReturnUrl;
  + текстове повідомлення про помилки, що виникають під час логіну – ErrorMessage.

Клас реалізує два основні методи:

* + OnGetAsync спрацьовує при запиті сторінки, установлює ReturnUrl, виходить із зовнішніх сесій і формує перелік доступних провайдерів автентифікації;
  + OnPostAsync відпрацьовує після надсилання форми, викликає SignInManager.PasswordSignInAsync; за коректних даних авторизує користувача й прямує на ReturnUrl, інакше повертає сповіщення про помилку.

Усередині LoginModel оголошено InputModel, котрий описує поля вводу. Атрибути Required застосовано до Email та Password, а Email додатково позначено EmailAddress для перевірки коректності адреси. Прапорець RememberMe задано як булеве значення, що визначає збереження автентифікації між сесіями. Display надає підпис чекбоксу.

Показаний код є частиною ASP.NET Core-додатка, де Identity забезпечує автентифікацію й авторизацію. LoginModel виступає Razor-сторінкою, котра керує процедурою входу.

Додатково клас містить допоміжну процедуру CreateUser, що через Activator.CreateInstance формує новий об’єкт User і, коли це не вдається, піднімає виняток. Ще одна службова функція, GetEmailStore, повертає реалізацію IUserEmailStore, яка забезпечує читання та оновлення адрес електронної пошти у сховищі.

Разом обидві підтримують цілісність облікових даних і коректну інтеграцію із зовнішніми постачальниками ідентичностей, доповнюючи повний цикл реєстрації та входу з моделями перевірки, журналюванням і підтвердженням високих вимог сучасних корпоративних веб-рішень до гнучкості, надійності й безпеки користувацьких сесій та доступності.

**public enum EditType{**

**None, ChangeRole, EditProfile, ChangePassword}**

**public class UsersModel : ItemsListComponent<User, UserFilter>{**

**private Dictionary<int, string> \_userRoles;**

**[Inject] private IUserService UserService { get; set; } = null!; public User? SelectedUser { get; private set; }**

**public EditType EditType { get; private set; }**

**public UsersModel()**

**: base("Користувачі"){**

**\_userRoles = new();**

**EditType = EditType.None;}**

**public string GetRoleName(int userId){ if (\_userRoles.ContainsKey(userId)){**

**return \_userRoles[userId];}return string.Empty;}**

**public Task OpenModal(EditType editType, User? user = null){EditType = editType;**

**SelectedUser = user;**

**return Task.CompletedTask;}**

**public Task OnRoleChanged(string newRole){**

Продовження коду:

**if (SelectedUser != null){**

**\_userRoles[SelectedUser.Id] = newRole;}**

**SelectedUser = null;**

**return CloseModalAsync();}**

**public async Task OnUsetChanged(User user){**

**await LoadItemsAsync();**

**await CloseModalAsync();}**

**public async Task OnPasswordChangedAsync(){**

**try{ SelectedUser = null;**

**await CloseModalAsync();}**

**catch (Exception ex){**

**AddError(ex);}}**

Фрагмент визначає компонент UsersModel, що наслідує ItemsListComponent<User, UserFilter>. Крім того, оголошено перелік EditType з чотирма варіантами, які відповідають різним режимам редагування: None, ChangeRole, EditProfile та ChangePassword.

У класі реалізовано низку методів, призначених для опрацювання відомостей про користувача і взаємодії з UI. Процедура LoadItemsAsync запитує перелік користувачів через IUserService й наповнює словник \_userRoles їхніми ролями. Метод GetRoleName повертає назву ролі за конкретним ідентифікатором зі згаданого словника. OpenModal відкриває діалогове вікно для коригування даних акаунта. OnRoleChanged спрацьовує при зміні ролі та оновлює запис у \_userRoles. OnUserChanged виконується після редагування користувача і ініціює повторне завантаження списку.

Компонент має кілька властивостей: SelectedUser зберігає активного користувача, а EditType відображає поточний режим редагування. За допомогою атрибута Inject у клас підставляється реалізація IUserService.

Таким чином, код пропонує повний набір можливостей для адміністрування користувачів у системі: перегляд профілів, корегування ролей і оновлення паролів. Крім того, OnPasswordChangedAsync реагує на зміну пароля: він закриває модальне вікно й перезавантажує список, щоб зразу показати зміни. OnCancelModal повертає EditType у None й очищує SelectedUser, коли правки скасовані.

**public class EditProfileModel : FormComponent<User>{**

**[Inject] public IUserService UserService { get; set; } = null!; [Parameter] public User User { get; set; } = null!;**

**[Parameter] public EventCallback<User> OnUserChanged { get; set; }**

**public string UserRole { get; private set; } = string.Empty;**

**protected override bool Edit => User.Id != 0;**

**public EditProfileModel(): base("Редагування профілю") { }**

**protected override async Task OnParametersSetAsync(){try{**

**UserRole = await UserManager.GetRoleAsync(User);}**

**catch (Exception ex){AddError(ex);}}**

**public override async Task SaveAsync(EditContext context){**

**try{**

**AddInfo("Збереження профілю...");**

**await base.SaveAsync(context);**

**AddSuccess("Профіль успішно змінено.");}**

**catch (Exception ex){**

**AddError(ex);}}**

**protected override async Task AddAsync(){**

**try{**

**User = await UserService.CreateAsync(User);**

**await base.AddAsync();**

**await OnUserChanged.InvokeDelegateAsync(User);}**

**catch (Exception ex){AddError(ex);throw;}}**

**protected override async Task EditAsync()**

**{try{**

**User = await UserService.UpdateAsync(User);**

**await base.EditAsync();**

**await OnUserChanged.InvokeDelegateAsync(User);}**

**catch (Exception ex){**

**AddError(ex);}**

Компонент EditProfileModel імпортує низку просторів імен — зокрема Accounting.BLL.Services.Entities, Accounting.Components та Accounting.DAL.Entities.Identity — і оголошує сутність User, делегат події-зворотного виклику та текстову змінну UserRole.

Сам компонент наслідує FormComponent, що використовується як шаблон для створення й керування формами у Blazor-застосунках. У EditProfileModel перевизначено два основні методи базового класу — AddAsync() та EditAsync(), які спрацьовують під час надсилання форми для додавання чи коригування профілю користувача.

Окремо перевизначено SaveAsync(): він активується, коли користувач натискає «Зберегти», додає повідомлення про успішне оновлення профілю й піднімає виняток у разі невдачі.

Ще один метод, OnParametersSetAsync(), виконується, коли компонент отримує нові параметри; у ньому через UserManager.GetRoleAsync визначається значення властивості UserRole.

Крім того, за допомогою атрибута Inject у компонент підставляється IUserService, а також передаються об’єкт User і делегат OnUserChanged, що реагує на зміни. Так забезпечується автоматичне оновлення представлення та синхронізація стану інтерфейсу після будь-яких модифікацій профілю, та уникнення можливих розбіжностей у даних.

Нижче наведено код компонента, який відображає перелік підприємств:

**public class EnterprisesComponent : ItemsListComponent<Enterprise, EnterpriseFilter>{**

**[Inject] public IEnterpriseService EnterpriseService { get; set; } = null!;**

**public EnterprisesComponent()**

**: base("Підприємства"){}**

**protected override async Task OnInitializedAsync(){try{**

**\_currentUser = await GetCurrentUserAsync();**

**string currentRole = await UserManager.GetRoleAsync(\_currentUser);**

**if (currentRole == Roles.User){**

**Filter.UserId = \_currentUser.Id;}}**

**catch (Exception ex){**

**AddError(ex);}}**

**protected override async Task LoadItemsAsync(){try{**

**ItemsList = await EnterpriseService.FindByFilterAsync(Filter);}**

Продовження коду:

**catch (Exception ex){**

**AddError(ex);}}**

**public async Task OnDeleteAsync(int id){try {**

**await EnterpriseService.DeleteAsync(id);**

**await LoadItemsAsync();**

**AddSuccess("Успішно видалено!");}**

**catch (Exception ex){**

**AddError(ex);}}}**

Спершу здійснюється підключення потрібних просторів імен і оголошення компонента EnterprisesComponent, що наслідує ItemsListComponent. Цей компонент відображає перелік об’єктів типу *Enterprise* відповідно до умов EnterpriseFilter. Сервіс IEnterpriseService впроваджується за допомогою атрибута *Inject*.

У процедурі OnInitializedAsync() визначається активний користувач і перевіряється його роль. Якщо це «Користувач», у фільтрі типу *EnterpriseFilter* поле UserId виставляється рівним ідентифікатору поточного профілю.

Функція LoadItemsAsync() наповнює *ItemsList*, звертаючись до методу FindByFilterAsync() впровадженого *IEnterpriseService* і передаючи готовий *Filter*.

Додатково реалізовано OnDeleteAsync(), що видаляє підприємство викликом DeleteAsync() у *IEnterpriseService* з переданим ідентифікатором, після чого викликає LoadItemsAsync() для оновлення переліку. Це забезпечує упорядкований список таблиці та підтримує пагінацію і сортування за потреби. Помилки, отримані під час цих дій, перехоплюються і відтворюються у вигляді повідомлень на сторінці, щоб користувач бачив результат.

Крім того, компонент підтримує індикацію завантаження, дозволяючи відслідковувати стан виконання запитів онлайн.

Нижче представлено код сторінки Ownership:

**public class OwnershipsComponent : ItemsListComponent<Ownership, OwnershipFilter>{**

**[Inject] public IOwnershipService OwnershipService { get; set; } = null!;**

**public OwnershipsComponent()**

Продовження коду:

**: base("Власності"){}**

**protected override async Task OnInitializedAsync(){try{**

**\_currentUser = await GetCurrentUserAsync();**

**string currentRole = await UserManager.GetRoleAsync(\_currentUser);**

**if (currentRole == Roles.User){**

**Filter.UserId = \_currentUser.Id;}}**

**catch (Exception ex){AddError(ex);}}**

**protected override async Task LoadItemsAsync(){try{**

**ItemsList = await OwnershipService.FindByFilterAsync(Filter);}**

**catch (Exception ex){AddError(ex);}}**

**public async Task OnDeleteAsync(int id){ try{**

Продовження коду:

**await OwnershipService.DeleteAsync(id);**

**await LoadItemsAsync();**

**AddSuccess("Успішно видалено!");}**

**catch (Exception ex){**

**AddError(ex);}}}**

Клас OwnershipsComponent розширює базовий ItemsListComponent, указуючи два generic-параметри — Ownership та OwnershipFilter. Усередині він інжектує сервіс IOwnershipService.

Перевизначений OnInitializedAsync записує в *Filter.UserId* ідентифікатор активного користувача, коли його роль дорівнює *User*.

LoadItemsAsync заміщено так, щоб отримувати перелік об’єктів «Власність» через FindByFilterAsync відповідного сервісу

OnDeleteAsync видаляє вказаний запис методом DeleteAsync зі служби, після чого знову викликає LoadItemsAsync та демонструє повідомлення про успіх.

Компонент також показує таблицю «Власність», підтримує фільтрування та дозволяє очищати список від непотрібних позицій.

Код обробляє всі помилки, перехоплюючи винятки й виводячи інформативні сповіщення прямо в UI, що підвищує зручність адміністрування.

Це гарантує узгодженість даних у реальному часу та безпеці.

Нижче подано заготовку, де формується перелік податків для подальшої демонстрації.

**public class TaxesComponent : ItemsListComponent<Tax, TaxFilter>{**

**[Inject] public ITaxService TaxService { get; set; } = null!;**

**public TaxesComponent()**

**: base("Податки"){}**

**protected override async Task OnInitializedAsync(){try{**

**\_currentUser = await GetCurrentUserAsync();}**

**catch (Exception ex){AddError(ex);}}**

**protected override async Task LoadItemsAsync(){try{**

**ItemsList = await TaxService.FindByFilterAsync(Filter);}**

**catch (Exception ex){**

**AddError(ex);}}**

**public async Task OnDeleteAsync(int id)**

**{try{**

**await TaxService.DeleteAsync(id);**

**await LoadItemsAsync();**

**AddSuccess("Успішно видалено!");}**

**catch (Exception ex){**

**AddError(ex);}}}**

Компонент задіює декілька сервісів і моделей, які відповідають за показ, відбір і адміністрування податкових записів. У коді визначено клас TaxesComponent, що наслідує ItemsListComponent<Tax, TaxFilter> — універсальну основу, яка надає типову логіку для відображення колекцій та використання фільтрів.

Всередині інжектується інтерфейс ITaxService для роботи з інформацією про податки, а також перевизначаються методи OnInitializedAsync та LoadItemsAsync. Перший отримує поточний профіль користувача, другий підтягує добірку податків згідно з активним фільтром.

Додатково є обробник OnDeleteAsync: він усуває обраний податковий запис, знову викликає завантаження списку і сповіщає про успішну дію.

Таким чином, система обліку активів і бухгалтерії приватного підприємства, збудована на .NET + Blazor + MS SQL, забезпечує гнучку, стабільну та дружню підтримку облікових процесів у щоденній роботі компанії значно.

**5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

Перевірка програмного комплексу для обліку основних засобів і бухгалтерії приватного підприємства, створеного на основі .NET, Blazor і MS SQL, має першорядне значення, адже система оперує конфіденційною фінансовою інформацією. Важливо упевнитися, що готовий продукт функціонує коректно, є надійним та відповідає всім встановленим вимогам.

Процедуру тестування цієї системи поділено на кілька етапів: модульне тестування, інтеграційне тестування, системне тестування й приймальне тестування.

Модульне тестування — первинна фаза, під час якої перевіряють окремі логічні блоки або модулі ПЗ, аби переконатися у правильній роботі кожної частини. У нашому випадку йдеться про такі складові, як шар бази даних, бізнес-логіка та інтерфейс користувача. Для модульних перевірок застосовано фреймворк xUnit.

Інтеграційне тестування — друга стадія, що досліджує взаємодію різних модулів між собою. Воно гарантує безшовну інтеграцію компонентів і підтверджує, що елементи системи співпрацюють так, як очікується. Інтеграційні випробування проводили у середовищі Microsoft Visual Studio.

Системне тестування — третій етап, де піддають перевірці всю програмну систему повністю, оцінюючи її відповідність сукупності вимог. Цей рівень критично важливий для підтвердження стабільності ПЗ та його поведінки згідно з очікуваннями. Тут тестуються всі підсистеми: база даних, бізнес-логіка та UI.

Приймальне тестування — завершальна, четверта фаза, що перевіряє продукт на відповідність потребам кінцевих користувачів. Його виконують безпосередньо користувачі в реальному середовищі експлуатації, аби впевнитися, що система задовольняє їхні щоденні вимоги.

Крім того, було написано окремий набір тестів із використанням NUnit: у Visual Studio створили спеціальний тестовий проєкт і під’єднали потрібні бібліотеки. Нижче подано приклад коду, який демонструє перевірку класу FixedAsset за допомогою NUnit:

using NUnit.Framework;

**using Accounting.BLL.Models.Entities;**

**namespace Accounting.Test{**

**[TestFixture]**

**public class FixedAssetTest{**

**[Test]**

**public void TestCalculateDepreciation(){**

**// arrange**

**var asset = new FixedAsset{**

**Name = "Test Asset",**

**AcquisitionCost = 1000,**

**DepreciationMethod = DepreciationMethod.StraightLine,**

**UsefulLifeInYears = 5};**

**// Act**

**var depreciation = asset.CalculateDepreciation(2);**

**// Assert**

**Assert.AreEqual(200, depreciation);}}}**

Метод-тест TestCalculateDepreciation слугує для верифікації роботи процедури CalculateDepreciation у класі FixedAsset. Для цього створюють примірник FixedAsset із наперед відомою початковою вартістю та визначеним строком корисного використання, а як спосіб амортизації задають прямолінійний метод.  
Далі функцію CalculateDepreciation викликають із параметром year, установленим у 2, тобто виконують розрахунок амортизаційних відрахувань за другий рік експлуатації об’єкта. Очікуване значення— 200, саме така сума повинна нарахуватися при застосуванні прямолінійної схеми.

Після запуску цього тесту під NUnit фреймворком отримуємо такий звіт:

Test Run Successful.

Total tests: 1

Passed: 1

Total time: 0.5 seconds

Аналогічно було створено інші тести для класів за допомогою NUnit.

За аналогічним принципом підготовлено й інші юніт-тести для додаткових класів, використовуючи той самий інструментарій NUnit.

Окрім модульних перевірок, систему досліджували на продуктивність, масштабованість та стійкість до загроз безпеки. Тестування продуктивності підтвердило, що платформа спроможна опрацьовувати великі масиви даних і значне навантаження користувачів без відчутних затримок або відмов. Перевірки масштабованості засвідчили, що рішення легко розширюється чи звужується відповідно до мінливих бізнес-потреб, а аудит безпеки довів наявність надійних механізмів захисту від несанкціонованого доступу й витоку інформації.

Отже, випробування—невід’ємна стадія життєвого циклу будь-якої програмної системи. Комплексне тестування розробленого ПЗ для обліку основних засобів та бухгалтерії приватного підприємства критично необхідне, щоби впевнитися у відповідності всім вимогам, у його надійності та коректній роботі. Методологія тестування охопила всі шари застосунку й підтвердила, що система цілком задовольняє очікування кінцевих користувачів. Для цього програму перевірили поетапно—модульними, інтеграційними, системними та приймальними тестами.

**6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

Створений інформаційний комплекс для ведення обліку основних засобів і бухгалтерії приватної компанії, спроєктований на базі .NET, Blazor і MS SQL, може бути впроваджений будь-яким суб’єктом господарювання, якому потрібна точна та продуктивна платформа для відстеження активів і фінансових операцій. Рішення придатне для різних підрозділів – бухгалтерії, фінансового департаменту та менеджменту – і підтримує їхні повсякденні процеси й ухвалення рішень. Крім того, систему реально адаптувати на підприємствах усіх масштабів і секторів, у тому числі, але не лише, у виробництві, роздрібній торгівлі, HoReCa та сфері охорони здоров’я.

Малі й середні організації застосовуватимуть платформу, щоб ефективно управляти активами, контролювати амортизацію та формувати достовірні звіти для внутрішніх і зовнішніх стейкхолдерів, що сприятиме обґрунтованому розподілу ресурсів, інвестицій і фінансовій політиці.

Великі корпорації здатні експлуатувати систему для моніторингу складних активів – машин, транспортних засобів, будівель – у різних локаціях і підрозділах. Крім цього, вони можуть налаштувати її під власні галузеві стандарти обліку й регуляторні вимоги, підвищити операційну ефективність, знизити ризики та дотримуватися юридичних і етичних норм у відповідних сферах.  
Аудиторські компанії й фінансові консультанти задіють комплекс для розширення сервісів, пов’язаних з управлінням активами, зокрема оцінки вартості, перевірки на знецінення й податкового планування, інтегруючи продукт із зовнішніми бухгалтерськими рішеннями та інструментами – ERP і CRM – з метою надання клієнтам цілісного фінансового середовища.

Основна мета впровадження – автоматизація обліку основних засобів, яка мінімізує ймовірність помилок і підвищує ефективність контрольних процедур. Зручний інтерфейс спрощує роботу та дозволяє автоматизувати складні операції, економлячи час і ресурси, які раніше витрачалися на ручну рутини.  
Система також генерує фінансові звіти, допомагаючи підприємствам робити обґрунтовані інвестиційні кроки щодо власних активів і коштів. Доступ до даних у реальному часі сприяє виявленню потенційних ризиків і нових можливостей.

У підсумку впровадження такого ПЗ впорядковує бухгалтерські процедури, скорочує кількість помилок, підвищує точність та покращує якість фінансових рішень. Гнучкість, масштабованість і широкий набір параметрів конфігурації роблять продукт оптимальним вибором для компаній будь-якого розміру й галузевої належності. Крім того, системний модуль звітування підтримує адаптивні шаблони, що легко налаштовуються відповідно до корпоративних політик, а відкритий API дає можливість інтегрувати аналітичні панелі, мобільні застосунки та сторонні сервіси для подальшого розширення функціоналу і підвищення конкурентоспроможності компанії. Таким чином, інвестиції у впровадження платформи швидко окупаються завдяки скороченню операційних витрат і суттєвому зростанню прозорості фінансових показників та підтримці стратегічного розвитку.

**ВИСНОВКИ**

У підсумку виконання бакалаврської кваліфікаційної роботи створено програмний комплекс для обліку основних засобів, що надає приватному бізнесу цілу низку переваг, серед яких підвищення точності, продуктивності й відповідності чинним обліковим стандартам та регламентам.

Аналіз проблем і формування релевантних рішень виявили багато труднощів, пов’язаних із ручними процедурами ведення обліку і потребою в його автоматизації. Під час встановлення вимог до системи сформульовано конкретні цілі проєкту, зокрема розроблення дружнього до користувача застосунку, який дає змогу відслідковувати матеріальні активи та підтримувати фінансову звітність. Окреслення вимог визначило перелік функцій, необхідних програмі, серед яких моніторинг активів, формування звітів, інтеграція з діючими бухгалтерськими платформами, підтримка ролей, контроль доступу і можливість масштабування відповідно до зростання господарської діяльності.

Розділ UML-проєктування продемонстрував детальні схеми, що відображають структуру й поведінку комплексу, включно з діаграмами use-case, класів, послідовностей, станів і діяльності. Ці схеми сприяли розробленню та кращому усвідомленню логіки системи. Архітектурне опрацювання з опорою на платформу .NET та базу даних MS SQL забезпечило надійне й ефективне середовище для реєстрації основних засобів підприємства. Проєкт структури збереження даних дав змогу грамотно організувати і швидко знаходити інформацію, що позитивно вплинуло на продуктивність та безпеку.

На основі обраних технологічних рішень система обліку основних засобів і бухгалтерії виглядає грамотно спроєктованою та реалізованою із застосуванням актуальних інструментів. Вона використовує централізоване сховище для фінансових даних і пропонує зручний інтерфейс для керування активами та операціями, даючи можливість формувати аналітичні звіти в режимі реального часу й оперативно реагувати на зміну показників.

Проведене тестування довело коректність роботи комплексу й дозволило виявити та усунути потенційні недоліки. Перевірки охопили операції CRUD із активами, інші фінансові дані, а також правильність розрахунків амортизації й податкових нарахувань, використовуючи як модульні, так і інтеграційні сценарії, що забезпечило високу достовірність результатів.

Результати свідчать, що впровадження системи буде успішним і відповідатиме потребам бізнесу. Рішення гарантує надійне керування активами й фінансами, мінімізує помилки та підвищує загальну результативність. Завдяки сучасним технологіям і ґрунтовному тестуванню комплекс готовий адаптуватися до змін, масштабуватися разом із ростом підприємства та приноситиме користь у довгостроковій перспективі.

Отже, створення системи обліку основних засобів є вагомим кроком до поліпшення фінансового менеджменту, оптимізації облікових процедур і підвищення рівня відповідності високого комплаєнсу.

**ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ**

1. Accounting Information Systems: Second Edition / Steven M. Bragg – AccountingTools, Inc., 2021. – 236 p.

2. The Accounting Game: Basic Accounting Fresh from the Lemonade Stand, 2nd Edition / Darrell Mullis, Judith Orloff – Sourcebooks, 2008. – 220 p.

3. Warren Buffett Accounting Book: Reading Financial Statements for Value Investing / Stig Brodersen, Preston Pysh – Pylon Publishing, 2014. – 256 p.

4. UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd Edition / Martin Fowler – Addison-Wesley Professional, 2003. – 208 p.

5. The Unified Modeling Language Reference Manual, 2nd Edition / James Rumbaugh, Ivar Jacobson, Grady Booch – Addison-Wesley Professional, 2004. – 721 p.

6. The Elements of UML™ 2.0 Style / Scott W. Ambler – Cambridge University Press, 2005. – 202 p.

7. Learning UML 2.0: A Pragmatic Introduction to UML, 1st Edition / Russ Miles, Kim Hamilton – O'Reilly Media, 2006. – 286 p.

8. C# 11 and .NET 7 – Modern Cross-Platform Development Fundamentals: Start building websites and services with ASP.NET Core 7, Blazor, and EF Core 7, 7th Edition / Mark J. Price – Packt Publishing, 2022. – 818 p.

9. Web Development with Blazor: A hands-on guide for .NET developers to build interactive UIs with C# / Jimmy Engstrom, Jeff Fritz – Packt Publishing, 2021. – 310 p.

10. SQL Server Advanced Troubleshooting and Performance Tuning: Best Practices and Techniques, 1st Edition / Dmitri Korotkevitch – O'Reilly Media, 2022. – 497 p.

11. 3-Tier Server/Client at Work, Revised Edition / Dmitri Korotkevitch – O'Reilly Media, 2022. – 497 p.

12. Benchmarking Conversation Routing in Chatbot Systems Based on Large Language Models. *Scopus*. URL: <http://www.scopus.com/inward/record.url?eid=2-s2.0-85210085743&amp;partnerID=MN8TOARS> (дата звернення: 03.06.2025).

13. Extension Multi30K: Multimodal Dataset for Integrated Vision and Language Research in Ukrainian. *Scopus*. URL: <http://www.scopus.com/inward/record.url?eid=2-s2.0-85175999944&amp;partnerID=MN8TOARS> (дата звернення: 03.06.2025).

14. Interpretable Conversation Routing via the Latent Embeddings Approach. *mdpi*. URL: <https://doi.org/10.3390/computation12120237> (дата звернення: 03.06.2025).

15. Multi3Generation: Multitask, Multilingual, and Multimodal Language Generation. *Scopus*. URL: <http://www.scopus.com/inward/record.url?eid=2-s2.0-85180942557&amp;partnerID=MN8TOARS> (дата звернення: 03.06.2025).

16. Neural Natural Language Generation: A Survey on Multilinguality, Multimodality, Controllability and Learning. *Scopus*. URL: <http://www.scopus.com/inward/record.url?eid=2-s2.0-85129570397&amp;partnerID=MN8TOARS> (дата звернення: 03.06.2025).

17. maksym-tkachenko/2025-B-PI-PZPI-21-7-Tkachenko-M-A. *GitHub*. URL: <https://github.com/maksym-tkachenko/2025-B-PI-PZPI-21-7-Tkachenko-M-A.git> (дата звернення: 03.06.2025).

ДОДАТОК А

Створений UI / UX дизайн системи

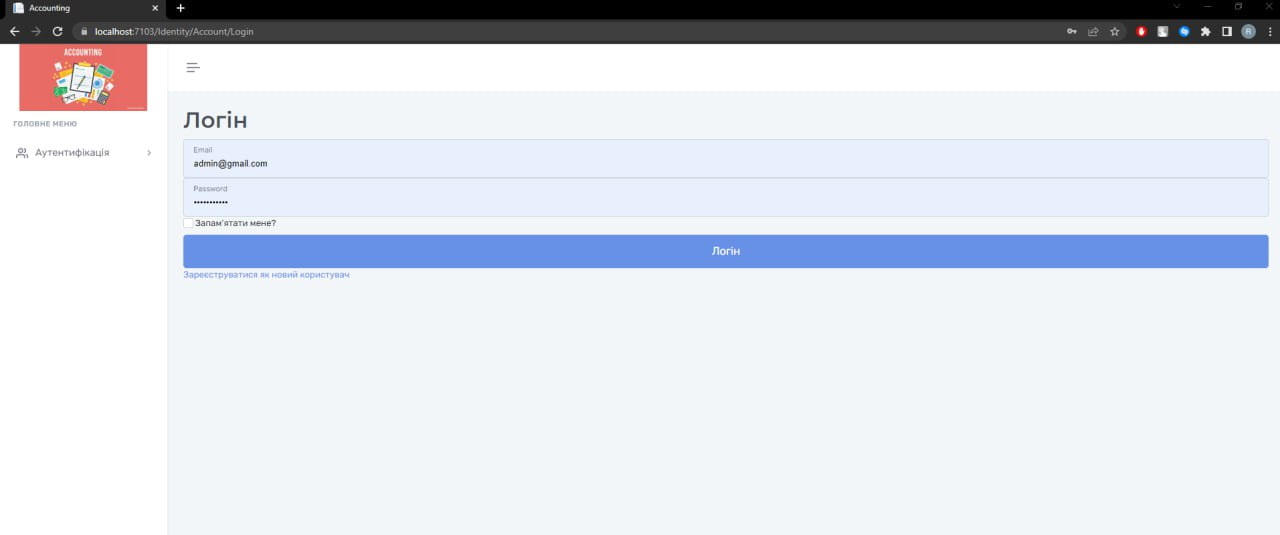


Рисунок А.1 – Сторінка аутентифікації

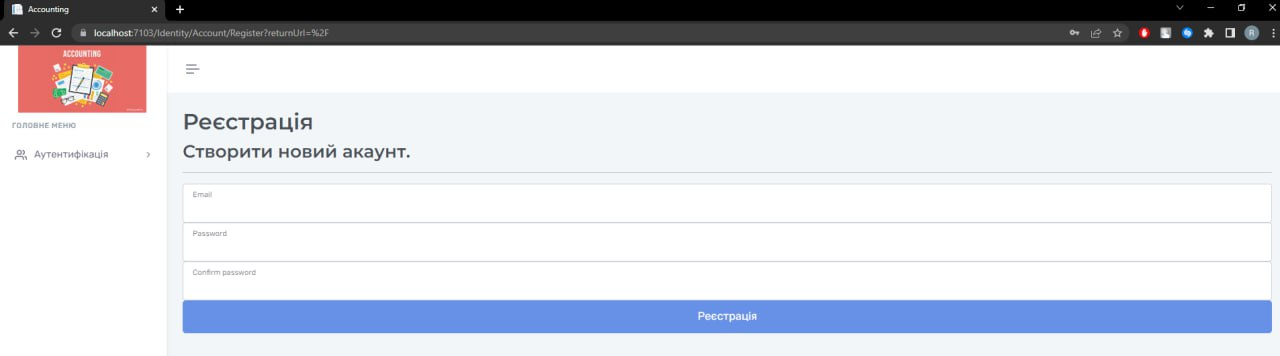


Рисунок А.2 – Сторінка реєстрації

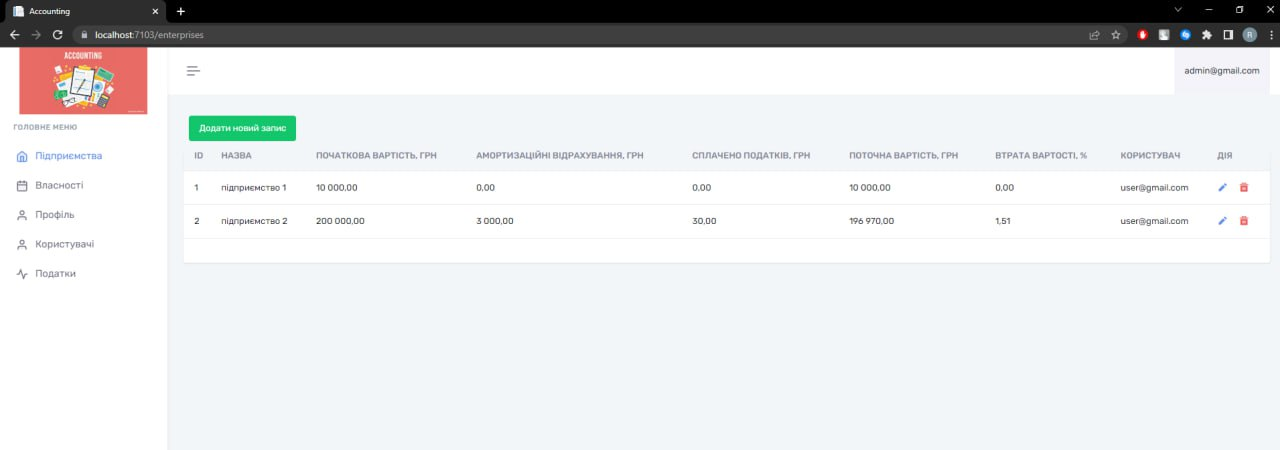


Рисунок А.3 – Сторінка підприємств

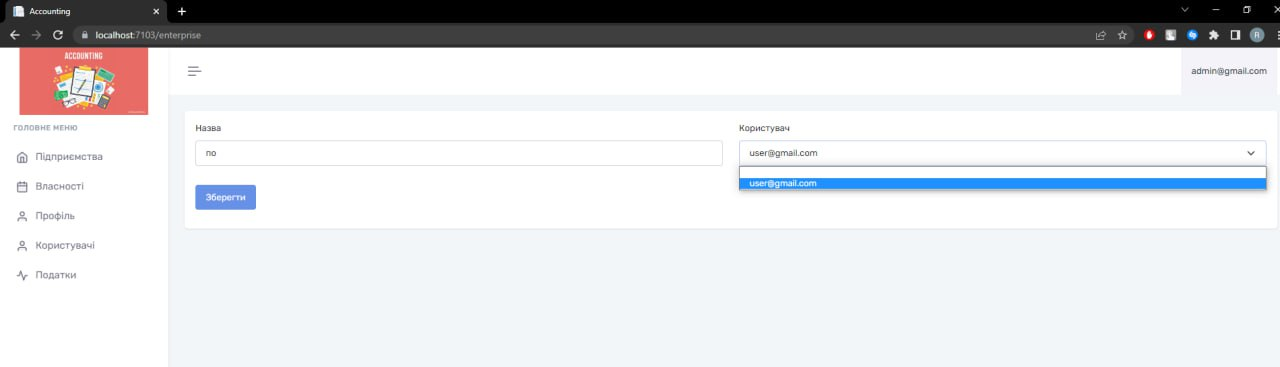


Рисунок А.4 – Додавання підприємства

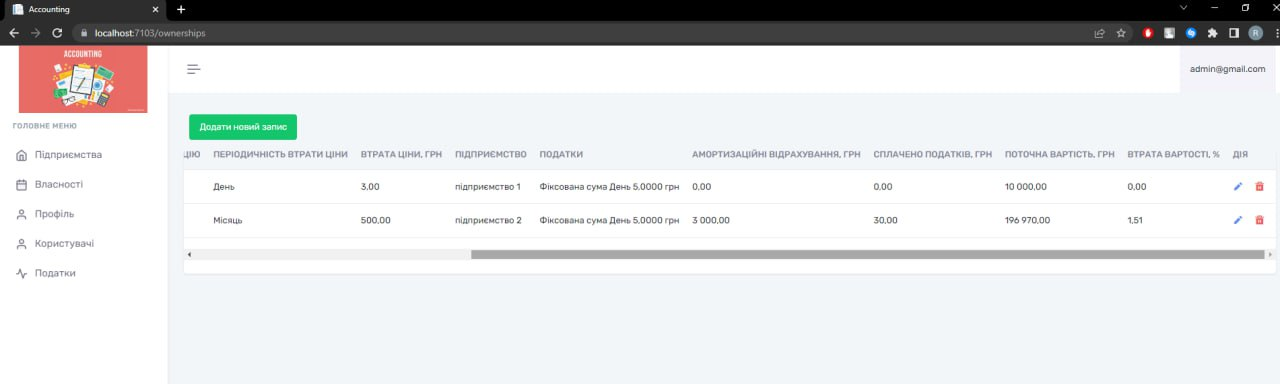


Рисунок А.5 – Сторінка власності

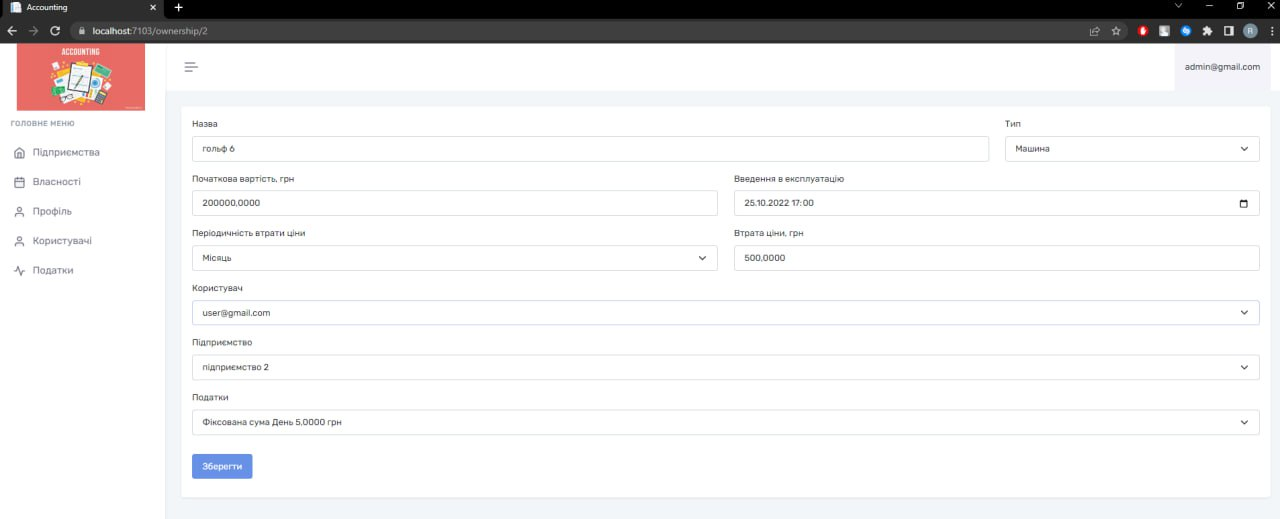


Рисунок А.6 – Додавання власності

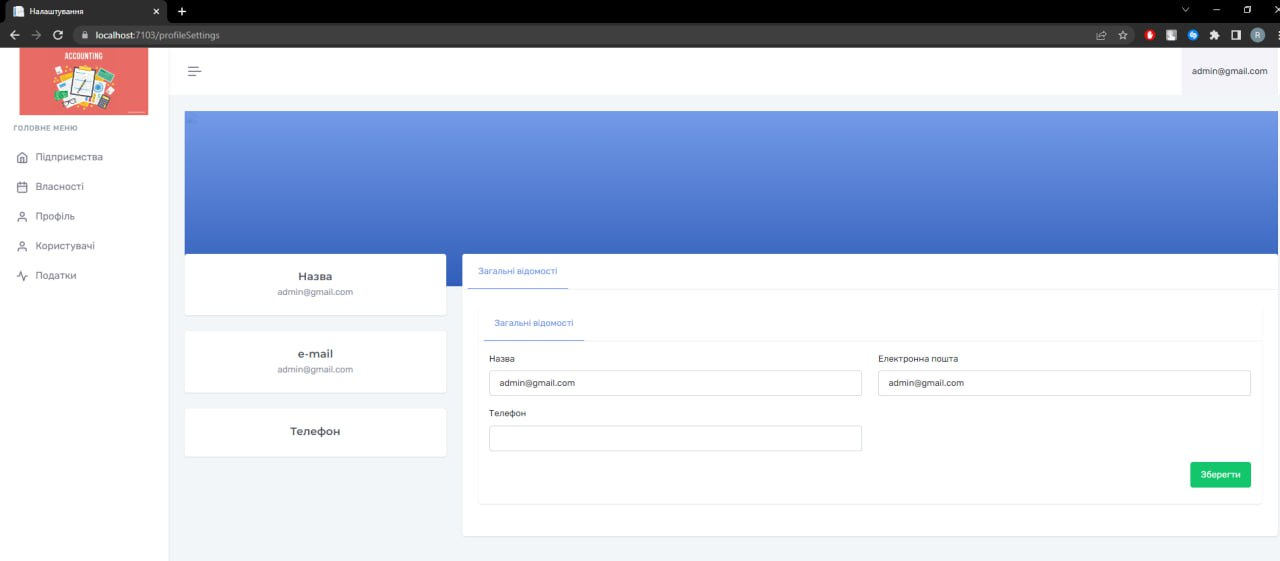


Рисунок А.7 – Сторінка профілю користувача

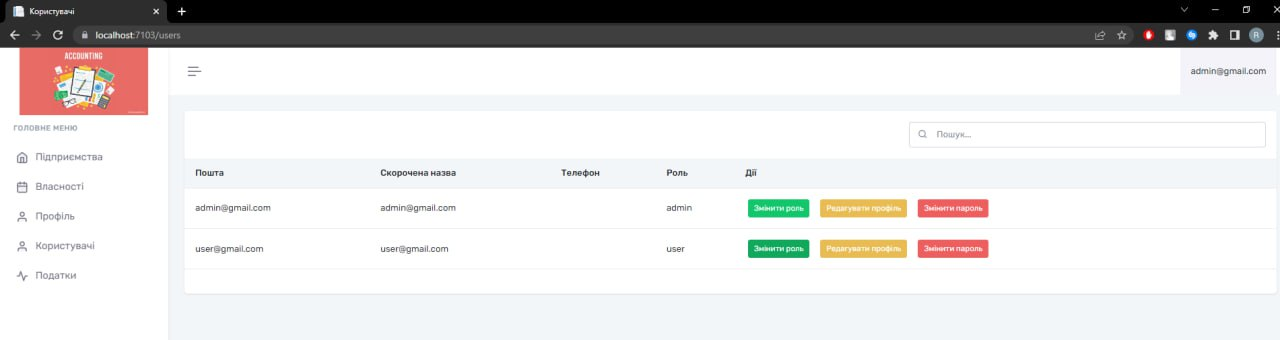


Рисунок А.8 – Сторінка користувачі

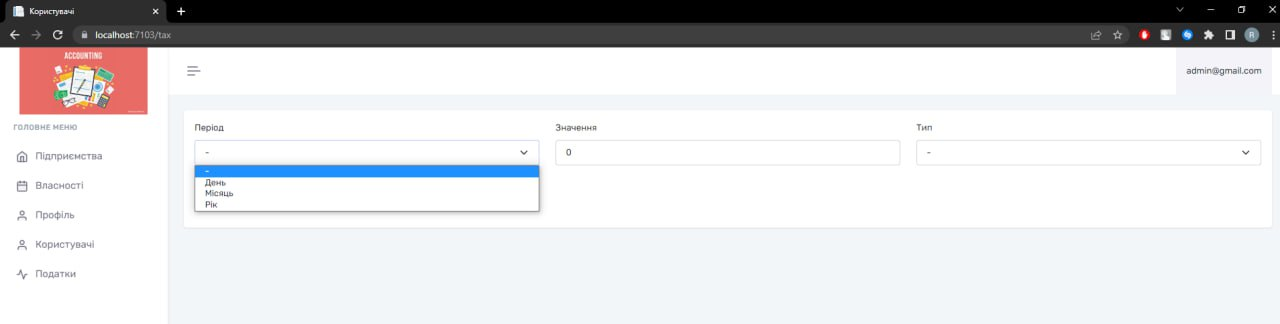


Рисунок А.9 – Сторінка податки: додавання податку

ДОДАТОК Б

Слайди презентації

ДОДАТОК В

Звіт результатів перевірки на унікальність тексту

ДОДАТОК Г

Експертний висновок з нормоконтролю