

**Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення**

**КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»**

**РОЗРОБКА ПЛАТФОРМИ СКЛАДСЬКОГО ТА БУХГАЛТЕРСЬКОГО
ОБЛІКУ ВИКОРИСТОВУЮЧИ .NET CORE ТА REST API**

**DEVELOPMENT OF A WAREHOUSE AND ACCOUNTING PLATFORM
USING .NET CORE AND REST API**

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-41
Хомич Іван Миколайович

Керівник:
Падалко Анатолій Михайлович

Кваліфікаційну роботу
допущено до захисту
«___» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних та інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Галузь знань: 12 «Інформаційні технології»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

«__» _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Хомичу Івану Миколайовичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Розробка платформи складського та бухгалтерського обліку використовуючи .NET Core та REST API»

Керівник роботи: к. ф.-м., н., Падалко А. М.

затверджені наказом закладу вищої освіти від «31» грудня 2025 р. № 541/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра «29» травня 2026 р.

3. Вихідні дані до роботи: предметна область складського та бухгалтерського обліку, технології розробки програмного забезпечення (C#, .NET Core, ASP.NET Core Web API, Blazor), засоби роботи з базами даних (SQL Server, Entity Framework Core).

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): аналіз предметної області складського та бухгалтерського обліку; постановка завдання; формування вимог до інформаційної системи; обґрунтування вибору технологій та архітектури; проєктування та реалізація програмного забезпечення; розробка бази даних; забезпечення захисту системи; тестування та налагодження програмного продукту.

5. Перелік графічного матеріалу: 13 рисунків, 2 таблиці.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	<i>Падалко А. М.</i>		
Специфікація вимог до розробленої системи	<i>Падалко А. М.</i>		
Розробка об'єкта проектування	<i>Падалко А. М.</i>		
Нормоконтроль	<i>Суринович О. М.</i>		
Гарант ОП	<i>Ліщина Н. М.</i>		
Показник запозичень тексту	<u> </u> %		
Академічна доброчесність	<i>Падалко А. М.</i>		

7. Дата видачі завдання «3» лютого 2026 р.

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 14.02.2026 р.	
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 03.03.2026 р.	
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 14.03.2026 р.	
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 14.04.2026 р.	
5	Практична реалізація об'єкта проектування та розробка бази даних	до 20.04.2026 р.	
6	Тестування та налагодження об'єкта проектування	до 04.05.2026 р.	
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедрі	до 29.05.2026 р.	

Здобувач вищої освіти _____

Іван ХОМИЧ

Керівник кваліфікаційної роботи _____

Анатолій ПАДАЛКО

АНОТАЦІЯ

Хомич І. М. Розробка платформи складського та бухгалтерського обліку використовуючи .NET Core та REST API. Рукопис.

Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел.

У першому розділі проведено аналіз предметної області складського та бухгалтерського обліку, розглянуто існуючі програмні рішення, визначено їхні переваги та недоліки, а також обґрунтовано доцільність розробки власної інформаційної системи. У другому розділі сформувано вимоги до розроблюваного програмного забезпечення, обґрунтовано вибір архітектури та технологій розробки, виконано моделювання системи за допомогою UML та визначено структуру бази даних. У третьому розділі описано процес практичної реалізації інформаційної системи, зокрема розробку серверної та клієнтської частин, реалізацію основних функціональних модулів, механізмів безпеки, бази даних, а також проведено тестування та налагодження програмного забезпечення. У висновках підсумовано результати виконаної роботи, підтверджено досягнення поставленої мети та виконання всіх завдань кваліфікаційної роботи.

Ключові слова: інформаційна система, складський облік, бухгалтерський облік, веб-застосунок, .NET Core, Blazor, REST API, база даних, автоматизація, облік товарів.

ABSTRACT

Khomych I. M. Development of a warehouse and accounting platform using .NET Core AND REST API. Manuscript.

Qualification work for bachelor's degree in Software Engineering, specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2026.

The bachelor's qualification work consists of an introduction, three chapters, conclusions, a list of references.

The first section analyzes the subject area of warehouse and accounting, considers existing software solutions, identifies their advantages and disadvantages, and justifies the feasibility of developing your own information system. The second section forms the requirements for the software being developed, justifies the choice of architecture and development technologies, models the system using UML, and defines the database structure. The third section describes the process of practical implementation of the information system, in particular, the development of the server and client parts, the implementation of the main functional modules, security mechanisms, and the database, as well as testing and debugging the software. The conclusions summarize the results of the work performed, confirm the achievement of the set goal and the fulfillment of all tasks of the qualification work.

Keywords: information system, warehouse accounting, accounting, web application, .NET Core, Blazor, REST API, database, automation, accounting of goods.

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1 АНАЛІЗ МЕТОДІВ РОЗРОБКИ ПЛАТФОРМ СКЛАДСЬКОГО ОБЛІКУ ТА ПОСТАНОВКА ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ	10
1.1 Аналіз сучасного стану проблеми.....	10
1.2 Постановка завдання на кваліфікаційну роботу бакалавра	12
РОЗДІЛ 2 СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ...	14
2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення.....	14
2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання....	20
РОЗДІЛ 3 РОЗРОБКА ПЛАТФОРМИ СКЛАДСЬКОГО ТА БУХГАЛТЕРСЬКОГО ОБЛІКУ	23
3.1 Практична реалізація об'єкта проєктування	23
3.2 Розробка бази даних	31
3.3 Захист інформаційно-комп'ютерної системи.....	35
3.4 Тестування та налагодження інформаційно-комп'ютерної системи	37
ВИСНОВКИ.....	41
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	43

ВСТУП

У сучасних умовах розвитку інформаційних технологій та цифровізації бізнес-процесів особливого значення набуває автоматизація облікових систем підприємств. Системи складського та бухгалтерського обліку є невід'ємною складовою ефективного функціонування організацій, оскільки забезпечують контроль руху товарів, фінансових операцій, залишків ресурсів та управління господарською діяльністю. Використання застарілих або неавтоматизованих методів обліку призводить до зниження продуктивності, виникнення помилок, втрати даних та ускладнення прийняття управлінських рішень.

Актуальність теми зумовлена стрімким розвитком інформаційних технологій та цифровою трансформацією бізнесу, що вимагає впровадження сучасних автоматизованих систем управління ресурсами підприємства. У сучасних умовах ефективного ведення складського та бухгалтерського обліку є критично важливим для забезпечення стабільної діяльності організацій, контролю за рухом матеріальних цінностей, фінансовими потоками та прийняття обґрунтованих управлінських рішень [1].

Традиційні підходи до ведення обліку, зокрема використання паперових носіїв або розрізнених програмних засобів, не забезпечують необхідного рівня оперативності, точності та надійності обробки даних. Це призводить до дублювання інформації, виникнення помилок, складності відстеження змін та обмежених можливостей аналізу діяльності підприємства. Крім того, відсутність централізованих систем ускладнює контроль доступу до даних та підвищує ризики їх втрати або несанкціонованого використання.

Особливої актуальності набуває використання веборієнтованих систем, які забезпечують доступ до даних у режимі реального часу, підтримують багатокористувацьку взаємодію та дозволяють інтегрувати різні бізнес-процеси в єдину інформаційну платформу. Застосування сучасних архітектурних підходів, зокрема клієнт-серверної моделі та REST API, дозволяє створювати

масштабовані, гнучкі та розширювані програмні рішення, здатні адаптуватися до змін вимог користувачів і зростання обсягів даних.

Важливим аспектом є також забезпечення інформаційної безпеки систем обліку, що передбачає реалізацію механізмів автентифікації, авторизації та розмежування прав доступу користувачів. У сучасних умовах кіберзагроз це є необхідною складовою будь-якої інформаційної системи.

Таким чином, розробка сучасної веборієнтованої платформи складського та бухгалтерського обліку з використанням технологій .NET Core та REST API є актуальним завданням, що має практичне значення для підвищення ефективності управління підприємством, автоматизації облікових процесів та забезпечення надійності і безпеки даних.

Об'єктом дослідження є процеси складського та бухгалтерського обліку підприємства. Предметом дослідження є методи, моделі та програмні засоби реалізації інформаційної системи обліку з використанням сучасних веб-технологій.

Метою кваліфікаційної роботи є розробка веборієнтованої платформи складського та бухгалтерського обліку з використанням технологій .NET Core та REST API, яка забезпечує ефективне управління товарами, складами, фінансовими операціями та користувачами.

Для досягнення поставленої мети у роботі необхідно вирішити такі завдання:

- провести аналіз сучасного стану проблеми автоматизації складського та бухгалтерського обліку;
- дослідити існуючі програмні рішення та визначити їх переваги і недоліки;
- сформулювати вимоги до розроблюваної інформаційної системи;
- обґрунтувати вибір технологій та засобів реалізації програмного забезпечення;
- розробити архітектуру системи та структуру бази даних;
- реалізувати програмний продукт у вигляді клієнт-серверної веб-системи;

- забезпечити механізми авторизації, автентифікації та розмежування доступу користувачів;
- реалізувати функціонал управління товарами, складами, операціями та платежами;
- провести тестування та налагодження системи.

Практичне значення отриманих результатів полягає у можливості використання розробленої системи для автоматизації облікових процесів підприємств, підвищення ефективності управління ресурсами, зменшення кількості помилок та забезпечення прозорості бізнес-процесів.

У кваліфікаційній роботі використано сучасні технології та інструменти розробки програмного забезпечення, зокрема мову програмування C#, платформу .NET Core, технологію ASP.NET Core Web API, фреймворк Blazor для побудови користувацького інтерфейсу, систему управління базами даних SQL Server, а також механізми JWT-автентифікації для забезпечення безпеки доступу до системи.

РОЗДІЛ 1

АНАЛІЗ МЕТОДІВ РОЗРОБКИ ПЛАТФОРМ СКЛАДСЬКОГО ОБЛІКУ ТА ПОСТАНОВКА ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

У сучасних умовах цифрової трансформації економіки ефективне управління ресурсами підприємства неможливе без використання автоматизованих інформаційних систем. Одним із ключових напрямів такої автоматизації є впровадження систем складського та бухгалтерського обліку, які забезпечують контроль руху товарно-матеріальних цінностей, облік фінансових операцій, формування звітності та підтримку управлінських рішень. Взаємозв'язок складського та бухгалтерського обліку обумовлює необхідність їх інтеграції в єдину інформаційну систему, що дозволяє підвищити точність, оперативність та прозорість обробки даних [1].

Традиційні методи ведення обліку, зокрема використання паперових носіїв або ізольованих програмних засобів, не відповідають сучасним вимогам до швидкості обробки інформації та її достовірності. Вони супроводжуються значними витратами часу, підвищеним ризиком виникнення помилок, складністю контролю змін та обмеженими можливостями аналізу. Крім того, відсутність централізованого зберігання даних ускладнює доступ до інформації та її узгодженість між різними підрозділами підприємства.

На сьогодні існує значна кількість програмних рішень для автоматизації облікових процесів. Серед них можна виділити комплексні ERP-системи, такі як SAP ERP, Microsoft Dynamics 365, Oracle NetSuite, а також гнучкі модульні рішення, зокрема Odoo. Окрему групу складають українські програмні продукти, адаптовані до вимог національного законодавства, зокрема MASTER:Бухгалтерія. Дані системи забезпечують широкий функціонал, що включає управління складськими запасами, фінансовий облік, аналітику, формування звітності та інтеграцію з іншими інформаційними системами [2].

Незважаючи на значний рівень розвитку існуючих програмних рішень, вони мають ряд суттєвих недоліків. По-перше, комплексні ERP-системи характеризуються високою вартістю впровадження, обслуговування та ліцензування, що робить їх недоступними для малих і середніх підприємств. По-друге, такі системи часто є надмірно складними у використанні, потребують тривалого налаштування та спеціалізованого навчання персоналу. По-третє, закритість архітектури багатьох комерційних продуктів обмежує можливості їх адаптації до специфічних потреб конкретного підприємства.

Окрім комплексних рішень, існують також вузькоспеціалізовані системи, орієнтовані окремо на складський або бухгалтерський облік. Однак їх використання часто призводить до фрагментації інформаційного середовища підприємства, оскільки відсутня єдина база даних і централізований контроль бізнес-процесів. Це ускладнює обмін даними між підсистемами, підвищує ризик виникнення невідповідностей та знижує ефективність управління.

Сучасні тенденції розвитку програмного забезпечення свідчать про активне впровадження веборієнтованих інформаційних систем, які забезпечують доступ до даних через мережу Інтернет, підтримують багатокористувацький режим роботи та дозволяють інтегрувати різні функціональні модулі в єдину платформу. Використання клієнт-серверної архітектури та REST API сприяє розділенню відповідальностей між компонентами системи, підвищенню її масштабованості, гнучкості та зручності супроводу.

Важливим аспектом сучасних систем обліку є забезпечення інформаційної безпеки, що включає реалізацію механізмів автентифікації, авторизації, розмежування прав доступу та ведення журналу дій користувачів. Це дозволяє не лише захистити дані від несанкціонованого доступу, але й забезпечити контроль і прозорість роботи системи.

Таким чином, аналіз сучасного стану проблеми показує, що існуючі програмні рішення не повністю задовольняють потреби підприємств у гнучких, доступних, масштабованих та безпечних системах складського та бухгалтерського обліку. Це обумовлює доцільність розробки нової

веборієнтованої платформи, яка поєднує функціональність, зручність використання, можливість адаптації та сучасні підходи до побудови програмного забезпечення.

Розробка такої системи з використанням технологій .NET Core та REST API дозволяє реалізувати ефективний інструмент автоматизації облікових процесів, який відповідає сучасним вимогам до програмних систем та може бути використаний у практичній діяльності підприємств різного масштабу.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

На основі проведеного аналізу сучасного стану проблеми автоматизації складського та бухгалтерського обліку встановлено, що існуючі програмні рішення не повною мірою задовольняють вимоги підприємств щодо гнучкості, доступності, масштабованості та інтеграції бізнес-процесів. Це обумовлює необхідність розробки сучасної інформаційної системи, яка забезпечує централізоване управління даними, ефективний контроль облікових процесів та зручну взаємодію користувачів із системою.

Метою кваліфікаційної роботи є розробка веборієнтованої платформи складського та бухгалтерського обліку з використанням технологій .NET Core та REST API, яка забезпечує автоматизацію процесів управління товарами, складами, фінансовими операціями та користувачами системи.

Для досягнення поставленої мети у кваліфікаційній роботі необхідно вирішити такі завдання:

- провести аналіз сучасного стану проблеми автоматизації складського та бухгалтерського обліку;
- дослідити існуючі програмні рішення та визначити їх переваги і недоліки;
- сформулювати вимоги до розроблюваної інформаційної системи;
- обґрунтувати вибір технологій та засобів реалізації програмного забезпечення;

- розробити архітектуру системи та структуру бази даних;
- реалізувати програмний продукт у вигляді клієнт-серверної веб-системи;
- забезпечити механізми авторизації, автентифікації та розмежування доступу користувачів;
- реалізувати функціонал управління товарами, складами, операціями та платежами;
- провести тестування та налагодження системи.

Результатом виконання кваліфікаційної роботи є програмний продукт у вигляді веборієнтованої інформаційної системи, що забезпечує автоматизацію облікових процесів та може бути використаний у практичній діяльності підприємств.

РОЗДІЛ 2

СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення

Для розробки сучасної інформаційної системи складського та бухгалтерського обліку доцільно використовувати підхід, що базується на клієнт-серверній архітектурі з використанням REST API. Такий підхід забезпечує розділення логіки представлення, бізнес-логіки та доступу до даних, що підвищує масштабованість, гнучкість і зручність супроводу програмного забезпечення. В якості моделі побудови системи обрано багаторівневу архітектуру, яка включає клієнтський рівень (інтерфейс користувача), серверний рівень (обробка запитів і бізнес-логіка) та рівень зберігання даних (база даних).

У процесі проєктування програмного забезпечення доцільно використати UML-нотацію, оскільки вона дає змогу формалізовано описати структуру та поведінку системи. Для розроблюваної платформи складського та бухгалтерського обліку UML-моделі використовуються для відображення основних ролей користувачів, функціональних можливостей системи, внутрішньої структури програмних компонентів і послідовності виконання окремих операцій [3, 4].

Для розроблюваної системи доцільно побудувати діаграму прецедентів, що відображає взаємодію користувачів із системою, діаграму класів, яка описує основні сутності предметної області та зв'язки між ними, а також діаграму послідовності, що демонструє логіку виконання ключових бізнес-процесів.

Діаграма прецедентів відображає функціональні можливості системи та взаємодію з нею основних акторів. У розроблюваній системі виділено дві основні ролі користувачів: адміністратор і звичайний користувач. Адміністратор має повний доступ до керування даними, зокрема може додавати, редагувати та видаляти записи, управляти користувачами, переглядати журнал дій і виконувати адміністративні операції. Звичайний користувач має обмежений

доступ і може переглядати інформацію, працювати з окремими модулями системи відповідно до наданих прав, а також виконувати операції, що не впливають на налаштування безпеки та адміністрування. До основних прецедентів системи належать авторизація користувача, перегляд інформації про товари, склади, залишки, платежі та товарні операції, додавання і редагування записів, пошук і фільтрація даних, а також аудит дій користувачів. Така модель дозволяє наочно представити межі функціонування системи та розподіл доступу між ролями (рис. 2.1).

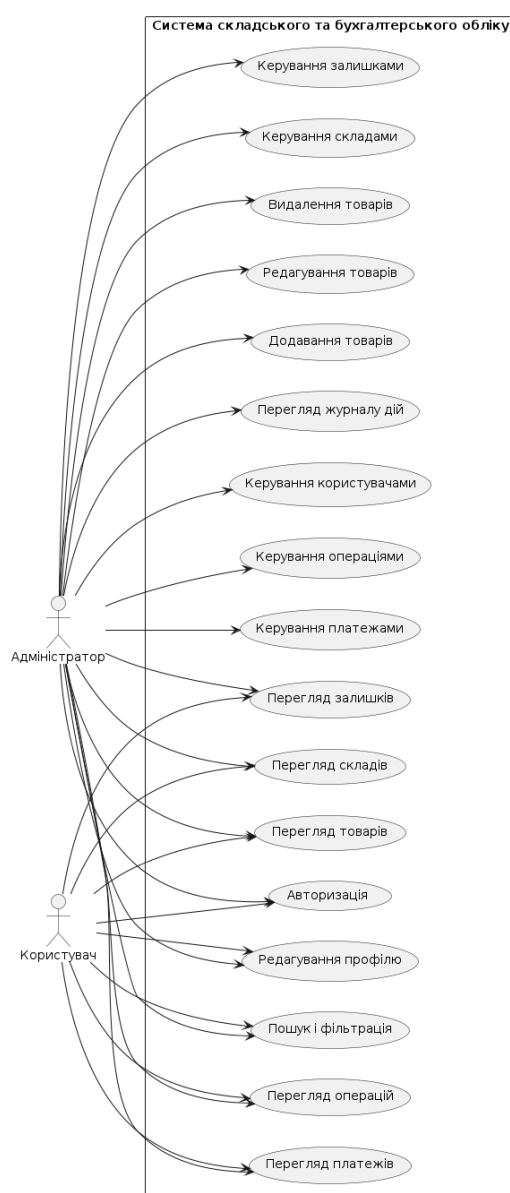


Рисунок 2.1 – UML-діаграма прецедентів системи складського та бухгалтерського обліку

Діаграма класів відображає статичну структуру системи та основні сутності предметної області (рис. 2.2). У розроблюваній системі ключовими класами є User, Product, Warehouse, InventoryItem, Payment, ProductTransaction та AuditLog. Клас User містить відомості про користувачів системи, включаючи логін, пароль у вигляді хешу та роль. Клас Product призначений для зберігання інформації про товари. Клас Warehouse описує склади, в яких зберігаються товари. Клас InventoryItem реалізує зв'язок між товарами та складами та зберігає інформацію про кількість певного товару на конкретному складі.

Клас ProductTransaction використовується для фіксації операцій закупівлі та продажу товарів. Клас Payment зберігає відомості про фінансові операції. Клас AuditLog реалізує журналювання дій користувачів і забезпечує контроль змін у системі. Зв'язки між класами демонструють, що один користувач може виконувати багато дій, один склад може містити багато позицій залишків, а один товар може бути присутнім на кількох складах.

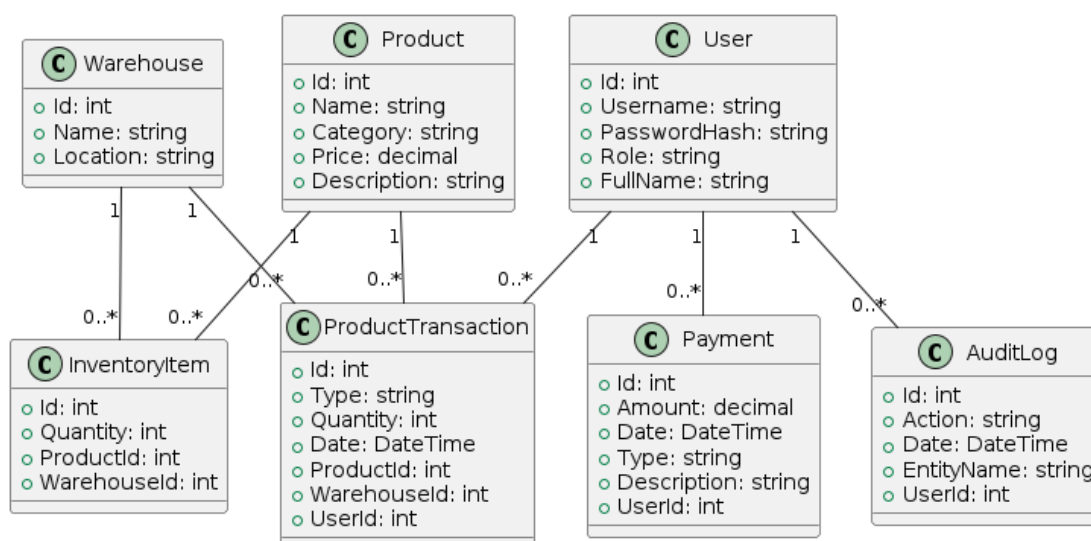


Рисунок 2.2 – UML-діаграма класів системи складського та бухгалтерського обліку

Діаграма послідовності авторизації відображає порядок взаємодії між користувачем, клієнтським інтерфейсом, серверною частиною та базою даних під час входу до системи (рис. 2.3). Користувач вводить логін і пароль у

вебінтерфейсі, після чого клієнтський застосунок передає запит на сервер через REST API. Сервер перевіряє наявність користувача в базі даних, виконує перевірку хешу пароля, а у випадку успішної автентифікації формує JWT-токен і повертає його клієнту. Отриманий токен використовується для подальшої авторизації запитів до системи.

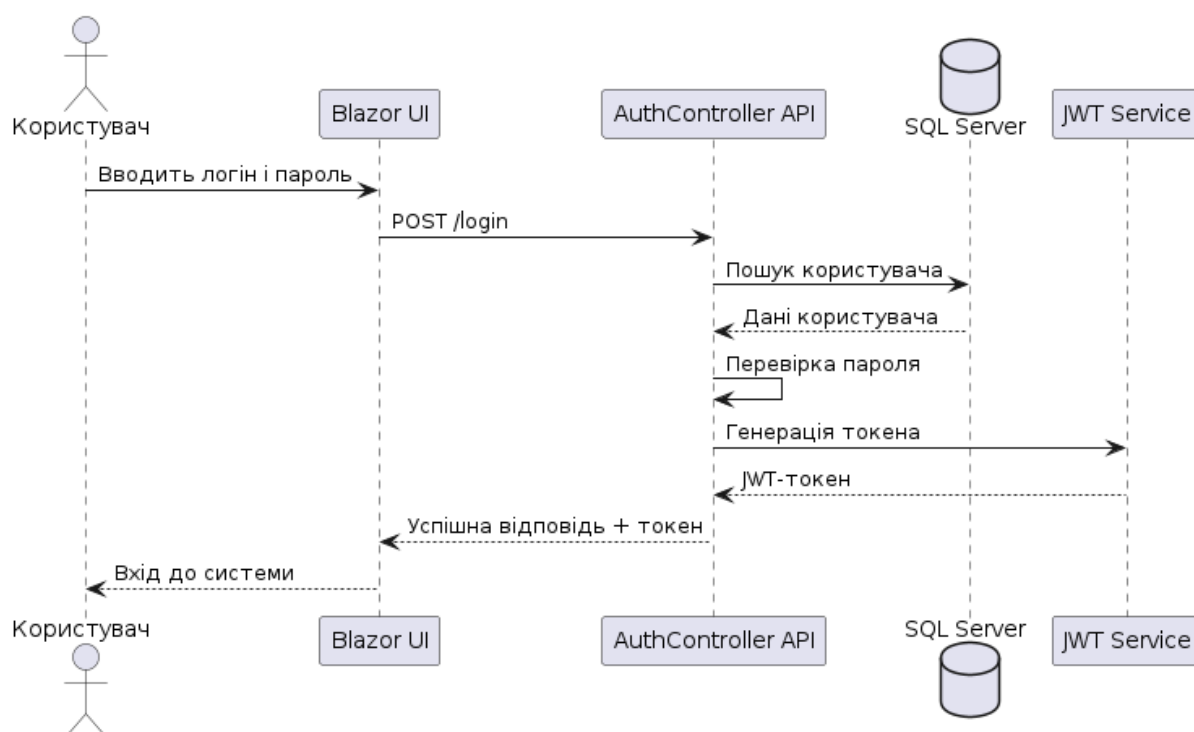


Рисунок 2.3 – UML-діаграма послідовності авторизації користувача в системі

Одним із ключових бізнес-процесів системи є виконання товарної операції. Користувач формує запит на створення операції закупівлі або продажу товару. Клієнтська частина передає відповідні дані до серверного API, яке перевіряє коректність параметрів, зберігає операцію в базі даних та оновлює залишки на складі. Після успішного завершення операції система формує запис у журналі дій, що забезпечує аудит змін. На рисунку 2.4 зображено UML-діаграма послідовності виконання товарної операції.

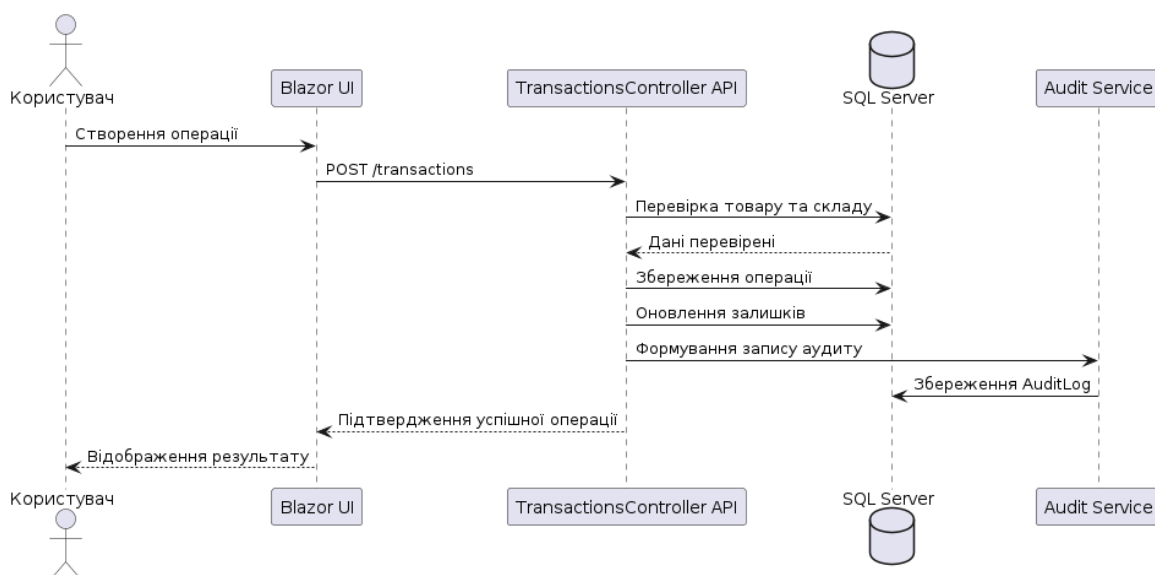


Рисунок 2.4 – UML-діаграма послідовності виконання товарної операції

Побудовані UML-діаграми дозволяють формалізовано описати функціональну та структурну організацію розроблюваної системи. Діаграма прецедентів відображає основні сценарії використання програмного продукту та розподіл прав доступу між користувачами. Діаграма класів характеризує структуру даних і зв'язки між сутностями предметної області. Діаграми послідовності демонструють логіку виконання ключових процесів системи.

Виявлення та аналіз вимог до системи здійснюється на основі аналізу предметної області, функціональних потреб користувачів та особливостей бізнес-процесів, пов'язаних із веденням складського та бухгалтерського обліку. Основними користувачами системи є адміністратор та звичайний користувач, які мають різні рівні доступу до функціональних можливостей системи.

Основною метою розроблюваної системи є створення зручного, надійного та масштабованого програмного засобу для автоматизації процесів обліку товарів, складів, фінансових операцій та дій користувачів.

До основних задач системи належать забезпечення ведення номенклатури товарів, облік складів і залишків, фіксація операцій закупівлі та продажу, облік платежів, управління користувачами та контроль їх дій у системі.

Функціональні вимоги до системи передбачають можливість авторизації користувачів, перегляду та редагування даних, виконання операцій додавання,

зміни та видалення записів, пошуку та фільтрації інформації, ведення журналу дій користувачів, а також відображення узагальненої інформації у вигляді статистичних показників. Система повинна забезпечувати роботу з такими сутностями, як товари, склади, залишки, транзакції та платежі.

Нефункціональні вимоги до системи включають забезпечення безпеки доступу до даних шляхом реалізації механізмів автентифікації та авторизації на основі JWT-токенів, забезпечення цілісності даних у базі даних, підтримку багатокористувацького режиму роботи, достатній рівень продуктивності при обробці запитів, а також можливість масштабування системи при збільшенні обсягів даних і кількості користувачів.

З точки зору UX/UI-дизайну система повинна мати зрозумілий і зручний інтерфейс користувача, що забезпечує швидкий доступ до основних функцій. Головна сторінка повинна містити узагальнену інформацію про стан системи у вигляді статистичних карток. Інтерфейс повинен підтримувати навігацію між модулями, такими як товари, склади, операції, платежі та користувачі, а також забезпечувати інтуїтивно зрозумілі форми введення та редагування даних.

Інформаційні потоки в системі організовані таким чином, що клієнтська частина формує HTTP-запити до серверної частини через REST API, сервер обробляє ці запити, взаємодіє з базою даних та повертає результат у вигляді відповідей, які відображаються користувачу. Такий підхід забезпечує централізовану обробку даних, зменшує навантаження на клієнт і дозволяє реалізувати гнучку архітектуру системи.

Таким чином, проведений аналіз вимог та обґрунтування архітектурних рішень дозволяє сформулювати основу для подальшої реалізації програмного забезпечення, що відповідає сучасним вимогам до інформаційних систем складського та бухгалтерського обліку.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Для реалізації розроблюваної інформаційної системи складського та бухгалтерського обліку необхідно обрати сучасні інструментальні засоби, які забезпечують ефективність розробки, надійність функціонування та можливість подальшого масштабування програмного продукту. Вибір технологій здійснюється на основі аналізу існуючих підходів до розробки веборієнтованих інформаційних систем, їх функціональних можливостей, переваг та недоліків.

Серед сучасних платформ для розробки веб-застосунків можна виділити такі технології, як Java Spring Boot, Node.js, Python Django та платформа .NET. Кожна з них має власні переваги, проте платформа .NET Core була обрана як основна через високу продуктивність, кросплатформеність, підтримку сучасних архітектурних підходів та тісну інтеграцію з мовою програмування C#. Крім того, .NET забезпечує широкий набір інструментів для створення масштабованих серверних додатків та має розвинену екосистему [5].

Для реалізації серверної частини системи обрано технологію ASP.NET Core Web API, яка дозволяє створювати RESTful сервіси для взаємодії між клієнтською частиною та сервером. Використання REST API забезпечує уніфікований доступ до ресурсів системи, спрощує інтеграцію з іншими сервісами та дозволяє розділити логіку обробки даних і представлення [6, 7].

Клієнтська частина реалізується з використанням технології Blazor, що дозволяє створювати інтерактивні веб-інтерфейси мовою C# без необхідності використання JavaScript. Це спрощує процес розробки, забезпечує повторне використання коду та дозволяє створювати єдине середовище розробки як для серверної, так і для клієнтської частини системи.

Для роботи з базою даних обрано систему управління базами даних SQL Server, яка забезпечує надійне зберігання даних, підтримку транзакцій та високу продуктивність. Взаємодія з базою даних реалізується за допомогою технології Entity Framework Core, яка є об'єктно-реляційним відображенням (ORM) і

дозволяє працювати з даними на рівні об'єктів, що значно спрощує розробку та підтримку коду [8].

З метою забезпечення безпеки системи використовується механізм автентифікації та авторизації на основі JWT (JSON Web Token). Даний підхід дозволяє реалізувати безпечний доступ до ресурсів системи, забезпечує перевірку прав користувача та підтримує масштабовану архітектуру без збереження стану сесії на сервері.

Для реалізації алгоритмів обробки даних у системі застосовуються стандартні підходи до роботи з реляційними базами даних, зокрема алгоритми додавання, оновлення та видалення записів (CRUD-операції), а також алгоритми пошуку і фільтрації даних. Пошук реалізується на основі SQL-запитів із використанням умов вибірки, що дозволяє ефективно знаходити необхідну інформацію за заданими параметрами.

Важливим аспектом є обробка товарних операцій, яка включає алгоритм оновлення залишків на складі. При виконанні операції закупівлі або продажу система автоматично змінює значення кількості товару в таблиці залишків відповідно до типу операції. Це забезпечує актуальність даних та виключає необхідність ручного коригування інформації.

Для забезпечення контролю дій користувачів використовується алгоритм журналювання (аудиту), який передбачає фіксацію кожної значущої операції в системі. При виконанні будь-якої дії створюється запис у журналі, що містить інформацію про користувача, тип дії, дату та об'єкт змін. Це дозволяє відслідковувати історію змін та підвищує рівень безпеки системи.

Для обґрунтування вибору інструментальних засобів розробки доцільно провести порівняльний аналіз сучасних платформ для створення веборієнтованих інформаційних систем.

Таблиця 2.1 – Порівняльна характеристика технологій розробки веб-застосунків

Критерій	.NET Core	Java Spring Boot	Node.js	Python Django
Продуктивність	Висока	Висока	Середня–висока	Середня
Мова програмування	C#	Java	JavaScript	Python
Кросплатформеність	Так	Так	Так	Так
Підтримка REST API	Повна	Повна	Повна	Повна
Інтеграція з БД	Висока (EF Core)	Висока (Hibernate)	Середня	Висока
Простота розробки	Висока	Середня	Висока	Висока
Масштабованість	Висока	Висока	Висока	Середня
Безпека	Висока	Висока	Середня	Висока
Підтримка UI (frontend)	Blazor	Потребує окремо	Потребує окремо	Потребує окремо

Аналіз наведених технологій показує, що всі розглянуті платформи дозволяють створювати сучасні веборієнтовані системи, проте кожна з них має свої особливості. Платформа .NET Core забезпечує високу продуктивність, зручність розробки та тісну інтеграцію між компонентами системи, що дозволяє використовувати єдину мову програмування як для серверної, так і для клієнтської частини [9].

На відміну від інших технологій, використання Blazor у складі платформи .NET дозволяє реалізувати користувацький інтерфейс без залучення додаткових фреймворків, що спрощує архітектуру системи та підвищує узгодженість програмного коду [10].

Таким чином, за результатами проведеного порівняльного аналізу обрано платформу .NET Core як основний інструмент розробки інформаційної системи складського та бухгалтерського обліку, оскільки вона найбільш повно відповідає вимогам до продуктивності, масштабованості, безпеки та зручності реалізації.

РОЗДІЛ 3

РОЗРОБКА ПЛАТФОРМИ СКЛАДСЬКОГО ТА БУХГАЛТЕРСЬКОГО ОБЛІКУ

3.1 Практична реалізація об'єкта проєктування

Практична реалізація інформаційної системи складського та бухгалтерського обліку виконана з використанням сучасних технологій платформи .NET. У якості середовища розробки використано Visual Studio, що забезпечує зручні інструменти для створення, налагодження та тестування програмного забезпечення.

Архітектура програмного продукту побудована за клієнт-серверною моделлю. Серверна частина реалізована у вигляді веб-сервісу на основі ASP.NET Core Web API, який відповідає за обробку запитів, реалізацію бізнес-логіки та взаємодію з базою даних. Клієнтська частина реалізована за допомогою технології Blazor, що дозволяє створити інтерактивний веб-інтерфейс для користувача.

Серверна частина системи організована у вигляді контролерів, які обробляють HTTP-запити. Наприклад, для реалізації функціоналу авторизації використовується відповідний контролер, який приймає дані користувача, перевіряє їх та повертає результат (ліст. 3.1).

Лістинг 3.1 – Приклад реалізації методу авторизації користувача

```
[HttpPost("login")]
public async Task<IActionResult> Login(LoginModel model)
{
    var user = await _context.Users
        .FirstOrDefaultAsync(u => u.Username == model.Username);
    if (user == null || !BCrypt.Net.BCrypt.Verify(model.Password,
        user.PasswordHash))
    {
        return Unauthorized("Невірний логін або пароль");
    }
    var token = _jwtService.GenerateToken(user);
    return Ok(new { token });
}
```

кінець лістингу 3.1

У наведеному фрагменті коду реалізовано перевірку облікових даних користувача. Спочатку виконується пошук користувача в базі даних, після чого здійснюється перевірка пароля за допомогою алгоритму хешування BCrypt. У разі успішної автентифікації генерується JWT-токен, який використовується для подальшої авторизації запитів.

Для роботи з базою даних використовується Entity Framework Core, який дозволяє виконувати операції над даними у вигляді об'єктів. Наприклад, додавання нового товару здійснюється наступним чином (ліст. 3.2).

Лістинг 3.2 – Додавання нового товару до бази даних

```
[HttpPost]
public async Task<IActionResult> CreateProduct(Product product)
{
    _context.Products.Add(product);
    await _context.SaveChangesAsync();

    return Ok(product);
}
```

кінець лістингу 3.2

Даний метод реалізує операцію додавання нового запису до таблиці товарів. Після додавання об'єкта до контексту викликається метод збереження змін, який формує відповідний SQL-запит до бази даних.

Однією з ключових функцій системи є обробка товарних операцій, що включає автоматичне оновлення залишків на складі. Для цього реалізовано відповідний алгоритм, який змінює кількість товару залежно від типу операції (ліст. 3.3).

Лістинг 3.3 – Алгоритм оновлення залишків товару

```
public async Task UpdateInventory(int productId, int warehouseId, int
quantity, string type)
{
    var item = await _context.InventoryItems
        .FirstOrDefaultAsync(i => i.ProductId == productId &&
i.WarehouseId == warehouseId);
```

```

if (item == null)
{
    item = new InventoryItem
    {
        ProductId = productId,
        WarehouseId = warehouseId,
        Quantity = 0
    };
    _context.InventoryItems.Add(item);
}

if (type == "IN")
    item.Quantity += quantity;
else if (type == "OUT")
    item.Quantity -= quantity;

await _context.SaveChangesAsync();
}

```

кінець лістингу 3.3

У наведеному алгоритмі здійснюється перевірка наявності запису про залишки товару на складі. Якщо запис відсутній, він створюється. Далі, залежно від типу операції (надходження або списання), змінюється кількість товару. Це забезпечує автоматичне підтримання актуального стану складських залишків.

Для забезпечення контролю дій користувачів у системі реалізовано механізм аудиту. При виконанні будь-якої операції створюється запис у журналі, що дозволяє відслідковувати історію змін (ліст. 3.4).

Лістинг 3.4 – Запис дії користувача в журнал

```

public async Task LogAction(string action, int userId)
{
    var log = new AuditLog
    {
        Action = action,
        UserId = userId,
        Date = DateTime.UtcNow
    };

    _context.AuditLogs.Add(log);
    await _context.SaveChangesAsync();
}

```

кінець лістингу 3.4

На рисунку 3.1 представлено головну сторінку інформаційної системи складського та бухгалтерського обліку, яка реалізує функціонал інформаційної панелі (Dashboard). Даний інтерфейс призначений для швидкого отримання узагальненої інформації про стан системи.

У верхній частині відображається інформація про поточного користувача та його роль, що підтверджує реалізацію механізму автентифікації та розмежування доступу. Основна частина сторінки представлена у вигляді інформаційних блоків, які містять ключові показники діяльності системи, зокрема кількість товарів, складів, залишків, платежів, операцій, а також фінансові показники – суми закупівель, продажів та загальне сальдо.

Кожен блок містить кнопку переходу до відповідного модуля системи, що забезпечує зручну навігацію та швидкий доступ до функціональних можливостей. Окремо представлені модулі профілю користувача, управління доступом, користувачами та журналом дій.

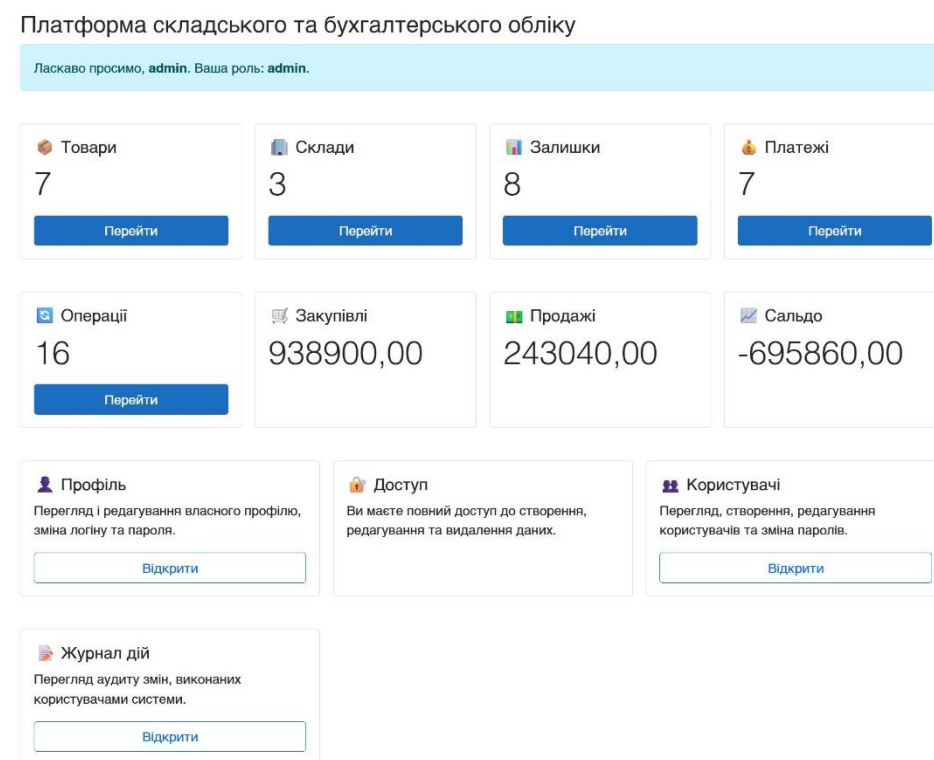


Рисунок 3.1 – Головна сторінка системи (інформаційна панель Dashboard)

На рисунку 3.2 представлено інтерфейс документації REST API, реалізований за допомогою інструменту Swagger. Даний інтерфейс забезпечує зручний доступ до опису всіх доступних HTTP-методів системи, а також дозволяє тестувати запити безпосередньо через веб-браузер.

У Swagger відображено основні контролери системи, зокрема модулі аудиту (Audit), авторизації (Auth), інформаційної панелі (Dashboard) та обліку залишків (Inventory). Для кожного ендпоінта визначено тип HTTP-запиту (GET, POST, PUT, DELETE), маршрут доступу та можливість перегляду параметрів запиту і відповіді.

Використання Swagger значно спрощує процес розробки та тестування програмного забезпечення, оскільки дозволяє швидко перевіряти коректність роботи API, аналізувати структуру запитів і відповідей, а також забезпечує автоматичну документацію системи. Це особливо важливо при побудові клієнт-серверних застосунків, де взаємодія між компонентами здійснюється через REST API.

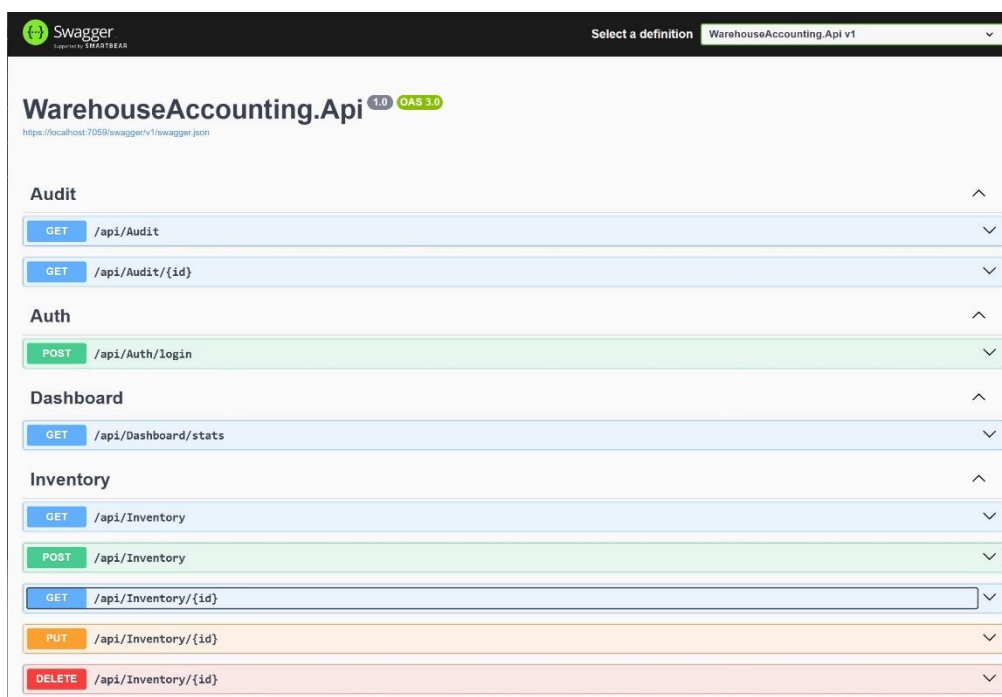


Рисунок 3.2 – Документація REST API системи (Swagger UI)

На рисунку 3.3 представлено основний інтерфейс управління товарами в інформаційній системі обліку складу. Даний модуль забезпечує перегляд, пошук, додавання, редагування та видалення товарних позицій.

У верхній частині інтерфейсу реалізовано механізми пошуку товарів за назвою та кодом, а також можливість фільтрації товарів за наявністю. Це дозволяє швидко знаходити необхідні записи у великому обсязі даних.

Основна частина сторінки представлена у вигляді таблиці, яка містить інформацію про товари, зокрема ідентифікатор, назву, код, ціну та кількість. Для кожного запису передбачено дії редагування та видалення, що забезпечує повний контроль над даними.

Зліва розташоване навігаційне меню, яке дозволяє переходити між основними модулями системи, такими як профіль користувача, склади, залишки, операції, платежі, користувачі та журнал дій. Також відображається інформація про поточного користувача та його роль, що підтверджує реалізацію механізму розмежування доступу.

The screenshot displays the 'Warehouse' management interface. On the left is a dark sidebar menu with options: Головна, Профіль, Товари, Склади, Залишки, Товарні операції, Платежі, Користувачі, and Журнал дій. The main area is titled 'Товари' and includes search filters for 'Пошук за назвою' and 'Пошук за кодом', along with a checkbox for 'Лише з наявністю'. A green 'Додати товар' button is present. Below is a table of goods with columns for ID, Назва, Код, Ціна, Кількість, and Дії.

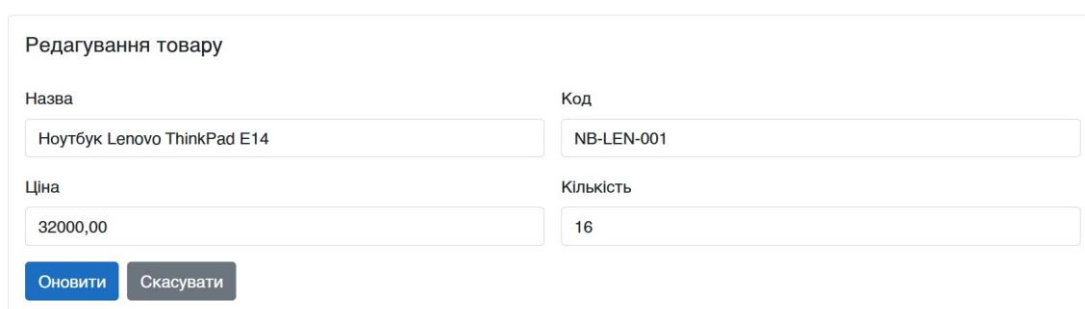
ID	Назва	Код	Ціна	Кількість	Дії
2	Macbook Air M3	M3	55000,00	1	Редагувати Видалити
3	Ноутбук Lenovo ThinkPad E14	NB-LEN-001	32000,00	16	Редагувати Видалити
4	Монітор Dell 24"	MON-DEL-024	7800,00	21	Редагувати Видалити
5	Клавіатура Logitech K120	KEY-LOG-120	850,00	30	Редагувати Видалити
6	Миша Logitech M185	MOU-LOG-185	620,00	58	Редагувати Видалити
7	SSD Samsung 1TB	SSD-SAM-1TB	3400,00	17	Редагувати Видалити
8	Принтер HP LaserJet	PRN-HP-001	12400,00	6	Редагувати Видалити

Рисунок 3.3 – Інтерфейс перегляду та управління товарами

На рисунку 3.4 представлено інтерфейс редагування товару, який забезпечує можливість внесення змін до існуючих записів у системі. Даний

модуль реалізує функціонал оновлення даних про товар та є складовою CRUD-операцій.

Інтерфейс містить основні поля для введення та редагування інформації, зокрема назву товару, унікальний код, ціну та кількість. Дані вводяться у відповідні текстові поля, що забезпечує зручність взаємодії користувача із системою. Користувач має можливість змінювати значення полів і зберігати оновлену інформацію за допомогою кнопки «Оновити» або скасувати зміни.



Редагування товару	
Назва	Код
Ноутбук Lenovo ThinkPad E14	NB-LEN-001
Ціна	Кількість
32000,00	16
Оновити Скасувати	

Рисунок 3.4 – Інтерфейс редагування товару

На рисунку 3.5 представлено інтерфейс управління товарними операціями, який забезпечує облік руху товарів у системі. Даний модуль дозволяє фіксувати операції закупівлі та продажу, а також аналізувати їх у зручному вигляді.

У верхній частині інтерфейсу відображаються узагальнені показники, зокрема кількість виконаних операцій, загальна сума закупівель, продажів та фінансове сальдо. Це дозволяє користувачу оперативно оцінити фінансовий стан діяльності.

Передбачено функціонал пошуку та фільтрації операцій за товаром, складом, типом операції та мінімальною сумою, що значно спрощує роботу з великими обсягами даних.

Основна частина представлена у вигляді таблиці, яка містить інформацію про кожну операцію, зокрема дату, товар, склад, тип операції, кількість, ціну, суму та коментар. Це забезпечує повну прозорість руху товарів і дозволяє контролювати всі зміни.

Товарні операції

Кількість операцій	Закупівлі	Продажі	Сальдо
16	938900,00	243040,00	-695860,00

Пошук за товаром	Пошук за складом	Тип операції	Мінімальна сума
		Усі ▼	

Додати операцію

ID	Дата	Товар	Склад	Тип	Кількість	Ціна	Сума	Коментар
16	25.03.2026	Миша Logitech M185	Основний склад	Purchase	20	440,00	8800,00	Додаткова закупівля мишок
15	24.03.2026	SSD Samsung 1TB	Резервний склад	Sale	3	3500,00	10500,00	Повторний продаж SSD
14	23.03.2026	Монітор Dell 24"	Основний склад	Purchase	7	6600,00	46200,00	Додаткова закупівля моніторів
13	22.03.2026	Ноутбук Lenovo ThinkPad E14	Резервний склад	Purchase	5	28500,00	142500,00	Додаткова закупівля ноутбуків
12	21.03.2026	Принтер HP LaserJet	Склад електроніки	Sale	2	12400,00	24800,00	Продаж принтерів
11	20.03.2026	SSD Samsung 1TB	Резервний склад	Sale	5	3400,00	17000,00	Продаж SSD
10	19.03.2026	Миша Logitech M185	Основний склад	Sale	12	620,00	7440,00	Продаж мишок
9	18.03.2026	Клавіатура Logitech K120	Основний склад	Sale	10	850,00	8500,00	Продаж клавіатур
8	17.03.2026	Монітор Dell 24"	Склад електроніки	Sale	6	7800,00	46800,00	Продаж моніторів

Рисунок 3.5 – Інтерфейс управління товарними операціями

На рисунку 3.6 представлено інтерфейс управління платежами в інформаційній системі складського та бухгалтерського обліку. Даний модуль забезпечує облік фінансових операцій підприємства та дозволяє контролювати витрати.

У верхній частині інтерфейсу реалізовано функціонал пошуку платежів за номером, описом та мінімальною сумою, що дозволяє ефективно працювати з фінансовими даними та швидко знаходити необхідні записи.

Основна частина сторінки представлена у вигляді таблиці, яка містить інформацію про платежі, зокрема ідентифікатор, номер платежу, суму, дату та опис. Для кожного запису передбачено можливість редагування та видалення, що забезпечує повний контроль над фінансовими операціями.

Також передбачено кнопку «Додати платіж», яка дозволяє створювати нові фінансові записи. Реалізація даного модуля дозволяє вести облік витрат підприємства, аналізувати фінансові потоки та забезпечує підтримку прийняття управлінських рішень.

Платежі

Пошук за номером

Пошук за описом

Мінімальна сума

Додати платіж

ID	Номер	Сума	Дата	Опис	Дії
1	12	122,00	27.03.2026	32e12	Редагувати Видалити
7	PAY-006	12700,00	26.03.2026	Оплата господарських витрат	Редагувати Видалити
6	PAY-005	31800,00	24.03.2026	Оплата рекламної кампанії	Редагувати Видалити
5	PAY-004	25000,00	22.03.2026	Оплата сервісного обслуговування обладнання	Редагувати Видалити
4	PAY-003	9600,00	19.03.2026	Оплата інтернету та зв'язку	Редагувати Видалити
3	PAY-002	18500,00	17.03.2026	Оплата за транспортні послуги	Редагувати Видалити
2	PAY-001	42000,00	15.03.2026	Оплата оренди складського приміщення	Редагувати Видалити

Рисунок 3.6 – Інтерфейс управління платежами

3.2 Розробка бази даних

У процесі розробки інформаційної системи складського та бухгалтерського обліку одним із ключових етапів стало проектування бази даних, оскільки саме вона забезпечує централізоване зберігання інформації, її цілісність, узгодженість та можливість оперативної обробки. База даних є основою функціонування всієї системи, оскільки через неї реалізується збереження відомостей про товари, склади, залишки, фінансові операції, користувачів та журнал дій.

Для реалізації сховища даних було обрано реляційну модель, яка є доцільною для систем такого типу, оскільки дозволяє структурувати інформацію у вигляді взаємопов'язаних таблиць і забезпечує підтримку зв'язків між основними сутностями предметної області. У якості системи керування базами даних використано SQL Server, а взаємодія прикладного програмного забезпечення з базою даних реалізована за допомогою Entity Framework Core. Такий підхід дозволив поєднати переваги реляційного зберігання даних із

можливостями об'єктно-орієнтованого програмування та спростив процес роботи з таблицями, запитам та зв'язками між сутностями.

У структурі розробленої бази даних виділено кілька основних сутностей, що відповідають функціональним модулям системи. Однією з центральних є таблиця користувачів, у якій зберігаються дані для автентифікації та авторизації в системі. Вона містить логін користувача, хеш пароля та роль, що дозволяє реалізувати механізм розмежування доступу. Наявність цієї таблиці забезпечує підтримку багатокористувацького режиму роботи та контроль прав доступу до окремих функцій системи.

Для ведення товарного обліку створено таблицю товарів, яка містить основні характеристики кожної товарної позиції, зокрема назву, код, ціну та кількість. Окремо реалізовано таблицю складів, у якій зберігаються відомості про назву складу та його адресу. Таке розділення дозволяє логічно структурувати інформацію і забезпечує гнучкість при роботі з даними, особливо в умовах використання кількох складських приміщень.

Важливою частиною бази даних є таблиця товарних операцій, яка використовується для фіксації руху товарів. У ній зберігається інформація про дату виконання операції, товар, склад, тип операції, кількість, ціну, загальну суму та коментар. Саме ця таблиця забезпечує реалізацію обліку закупівель і продажів, а також дозволяє формувати історію руху матеріальних цінностей. На її основі система виконує оновлення залишків і розрахунок узагальнених показників.

Для збереження інформації про фінансові витрати в системі передбачено таблицю платежів. У ній зберігаються номер платежу, сума, дата та опис. Наявність окремої таблиці платежів дозволяє відокремити товарні операції від фінансових записів та забезпечити більш точний облік господарської діяльності.

З метою контролю змін у системі реалізовано таблицю журналу дій, яка призначена для фіксації подій, пов'язаних із роботою користувачів. У ній зберігаються дата і час виконання дії, користувач, тип операції, сутність, до якої вона застосована, і текстовий опис змін. Це дає можливість здійснювати аудит

подій, відслідковувати історію змін та підвищує загальний рівень безпеки системи.

Між таблицями бази даних реалізовано логічні зв'язки, які забезпечують цілісність інформації та коректність виконання операцій. Зокрема, таблиця товарних операцій пов'язана з таблицями товарів і складів через відповідні зовнішні ключі. Це дозволяє однозначно визначити, який саме товар і на якому складі брав участь у певній операції. Таблиця журналу дій, у свою чергу, пов'язана з користувачами, що дає змогу встановити, ким саме була виконана конкретна дія.

Під час проєктування бази даних особлива увага приділялася забезпеченню цілісності даних. Для цього в таблицях визначено первинні ключі, зовнішні ключі, а також обмеження на обов'язковість заповнення окремих полів. Використання таких механізмів дозволяє уникнути дублювання інформації, зменшити ймовірність помилок при внесенні даних та забезпечити узгодженість між усіма компонентами системи.

Отже, розроблена база даних є логічно структурованою, цілісною та придатною для реалізації всіх функціональних можливостей інформаційної системи складського та бухгалтерського обліку. Вона забезпечує ефективне збереження та опрацювання даних, підтримує зв'язки між ключовими сутностями предметної області та створює надійну основу для подальшого функціонування програмного продукту.

На рисунку 3.7 представлено структуру бази даних із відображенням основних сутностей та зв'язків між ними. Центральне місце займає таблиця товарних операцій, яка поєднує інформацію про товари, склади та користувачів і забезпечує реалізацію обліку руху товарів. Таблиці користувачів і журналу дій дозволяють реалізувати механізми безпеки та аудиту, тоді як таблиці товарів і складів забезпечують зберігання базових даних предметної області.

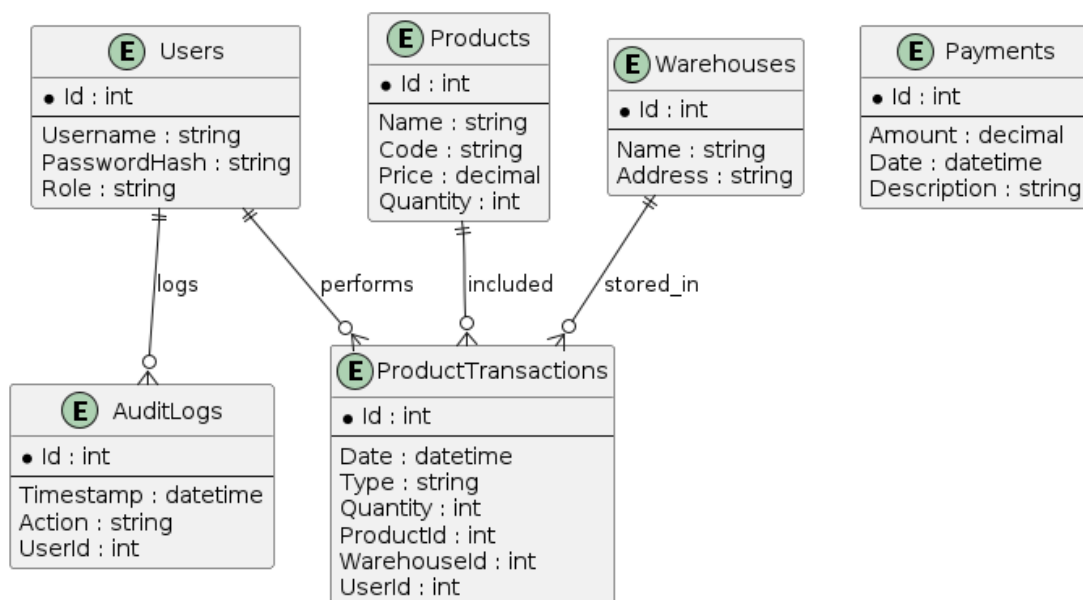


Рисунок 3.7 – Діаграма структури бази даних інформаційної системи складського та бухгалтерського обліку

Практична реалізація структури бази даних у програмному коді здійснюється за допомогою моделей сутностей. Приклад опису однієї з ключових сутностей наведено в лістингу 3.5.

Лістинг 3.5 – Приклад моделі сутності товарної операції

```

public class ProductTransaction
{
    public int Id { get; set; }
    public DateTime Date { get; set; }
    public string Type { get; set; } = string.Empty;
    public int Quantity { get; set; }

    public int ProductId { get; set; }
    public int WarehouseId { get; set; }
    public int UserId { get; set; }

    public Product Product { get; set; } = null!;
    public Warehouse Warehouse { get; set; } = null!;
    public User User { get; set; } = null!;
}
  
```

кінець лістингу 3.5

У наведеному фрагменті коду описано модель сутності товарної операції, яка використовується для збереження інформації про рух товарів у системі. Клас

містить як основні атрибути операції, так і зовнішні ключі, що забезпечують зв'язок із відповідними сутностями. Використання навігаційних властивостей дозволяє реалізувати зручну роботу з пов'язаними даними засобами Entity Framework Core.

3.3 Захист інформаційно-комп'ютерної системи

У сучасних інформаційних системах питання захисту даних є одним із ключових, оскільки від цього залежить надійність функціонування програмного забезпечення, збереження конфіденційної інформації та запобігання несанкціонованому доступу. У розроблюваній системі складського та бухгалтерського обліку реалізовано комплекс заходів, спрямованих на забезпечення безпеки даних та контролю доступу користувачів.

Одним із основних механізмів захисту є автентифікація користувачів, яка передбачає перевірку їх облікових даних під час входу до системи. Для цього використовується перевірка логіну та пароля, при цьому паролі зберігаються у базі даних у вигляді хешів із застосуванням сучасних криптографічних алгоритмів. Такий підхід дозволяє уникнути збереження паролів у відкритому вигляді та підвищує рівень захисту системи.

Після успішної автентифікації користувачу надається доступ до системи відповідно до його ролі. У системі реалізовано механізм авторизації, який базується на розмежуванні прав доступу. Користувачі поділяються на ролі, зокрема адміністратор та звичайний користувач. Адміністратор має розширені права, що дозволяють керувати всіма модулями системи, включаючи управління користувачами, тоді як звичайний користувач має обмежений доступ до функціональних можливостей.

Реалізація механізму управління користувачами представлена на рисунку 3.8.

Користувачі

Пошук за логіном

Фільтр за роллю

Усі ролі

Створити користувача

ID	Логін	Роль	Дії
1	admin	admin	Редагувати Змінити пароль
2	user	user	Редагувати Змінити пароль Видалити

Рисунок 3.8 – Інтерфейс управління користувачами

На даному інтерфейсі відображається список користувачів системи із зазначенням їх логіну та ролі. Передбачено можливість пошуку, редагування та видалення користувачів, а також зміни їх ролей і паролів. Це дозволяє адміністратору ефективно керувати доступом до системи та забезпечувати контроль за користувачами.

На рисунку 3.9 представлено інтерфейс журналу дій користувачів, який реалізує функціонал аудиту в інформаційній системі. Даний модуль призначений для фіксації та контролю всіх значущих подій, що відбуваються під час роботи користувачів із системою.

У журналі відображається інформація про виконані дії, зокрема ідентифікатор запису, дата та час виконання операції, користувач, тип дії, сутність, до якої була застосована дія, а також опис змін. Це дозволяє відстежувати історію роботи системи, аналізувати поведінку користувачів та забезпечувати контроль за змінами даних. Інтерфейс також містить засоби пошуку та фільтрації інформації за користувачем, сутністю та типом дії, що значно спрощує навігацію по журналу та дозволяє швидко знаходити необхідні записи.

Журнал дій

Пошук за користувачем
Пошук за сутністю
Пошук за дією

ID	Дата/Час (UTC)	Користувач	Дія	Сутність	EntityId	Опис
7	2026-03-28 11:24:38	anonymous	LoginSuccess	Auth	1	Успішний вхід користувача 'admin' з пароллю 'admin'.
6	2026-03-27 23:38:37	admin	Update	Warehouse	1	Оновлено склад '0'.
5	2026-03-27 23:37:41	anonymous	LoginSuccess	Auth	1	Успішний вхід користувача 'admin' з пароллю 'admin'.
4	2026-03-27 23:29:13	anonymous	LoginSuccess	Auth	1	Успішний вхід користувача 'admin' з пароллю 'admin'.
3	2026-03-27 23:29:09	anonymous	LoginFailed	Auth	1	Невдала спроба входу для користувача 'admin': невірний пароль.
2	2026-03-27 23:24:00	anonymous	LoginSuccess	Auth	1	Успішний вхід користувача 'admin' з пароллю 'admin'.
1	2026-03-27 22:53:12	anonymous	LoginSuccess	Auth	1	Успішний вхід користувача 'admin' з пароллю 'admin'.

Рисунок 3.9 – Інтерфейс журналу дій користувачів (Audit Log)

Журнал містить інформацію про дату і час виконання дії, користувача, тип операції, сутність та опис змін. Це дозволяє забезпечити прозорість роботи системи та здійснювати контроль за всіма виконаними операціями.

Додатковим рівнем захисту є перевірка прав доступу при виконанні запитів до серверної частини системи. Кожен запит обробляється з урахуванням ролі користувача, що унеможливорює виконання операцій без відповідних прав. Такий підхід забезпечує захист від несанкціонованого доступу до даних і функціональних можливостей системи.

Таким чином, у розроблюваній інформаційній системі реалізовано комплексний підхід до забезпечення безпеки, який включає автентифікацію, авторизацію, шифрування паролів, розмежування прав доступу та аудит дій користувачів. Це дозволяє забезпечити надійний захист інформації та стабільну роботу програмного забезпечення.

3.4 Тестування та налагодження інформаційно-комп'ютерної системи

Після реалізації інформаційної системи складського та бухгалтерського обліку було проведено її тестування та налагодження з метою перевірки коректності функціонування, відповідності поставленим вимогам та виявлення можливих помилок у роботі програмного забезпечення [11].

Тестування виконувалося у середовищі розробки Visual Studio із використанням вбудованих засобів налагодження, а також шляхом взаємодії з REST API через Swagger. Основна увага приділялася перевірці функціональних можливостей системи, зокрема операцій створення, редагування, видалення та перегляду даних.

У процесі тестування було перевірено роботу модуля управління товарами. Зокрема, виконувалося додавання нового товару із заповненням усіх обов'язкових полів, після чого перевірялося його відображення у списку. Також здійснювалося редагування існуючих записів і перевірка коректності оновлення даних у базі. При видаленні товару перевірялася його відсутність у таблиці після виконання операції. Усі зазначені дії виконувалися без помилок, що підтверджує правильність реалізації CRUD-операцій.

Аналогічно було протестовано модуль управління складами, де перевірялася можливість створення нових складів, редагування їх характеристик та видалення записів. Результати тестування показали стабільну роботу функціоналу та коректну взаємодію з базою даних.

Особливу увагу приділено тестуванню модуля товарних операцій, оскільки він реалізує основну бізнес-логіку системи. Було перевірено сценарії додавання операцій закупівлі та продажу товарів. При виконанні операції закупівлі система коректно збільшувала кількість товару на складі, тоді як при продажу — зменшувала її. Також перевірялася правильність розрахунку загальної суми операції. У результаті тестування встановлено, що алгоритм оновлення залишків працює коректно.

Тестування модуля платежів включало перевірку додавання, редагування та видалення фінансових записів. Було підтверджено, що система правильно обробляє введені дані та відображає їх у таблиці, що дозволяє вести облік витрат.

Окремо проведено тестування механізмів автентифікації та авторизації. Перевірялося введення коректних і некоректних облікових даних, а також обмеження доступу до функціоналу системи відповідно до ролі користувача. Було встановлено, що при введенні неправильного пароля доступ до системи не

надається, а при спробі виконання дій без відповідних прав система коректно обмежує доступ.

Також перевірено роботу журналу дій користувачів. У ході тестування встановлено, що всі ключові операції (додавання, редагування, видалення даних) фіксуються в системі із зазначенням користувача, дати та типу дії. Це підтверджує правильність реалізації механізму аудиту.

Налагодження програмного забезпечення здійснювалося шляхом покрокового виконання коду із використанням точок зупину (breakpoints), що дозволило перевірити логіку роботи окремих методів і своєчасно виявити та усунути помилки. Зокрема, було усунуто неточності у роботі алгоритму оновлення залишків та обробці запитів до бази даних.

Крім того, було протестовано взаємодію клієнтської та серверної частин системи. Перевірено коректність обміну даними через REST API, правильність обробки HTTP-запитів та відповідей, а також стабільність роботи системи при виконанні послідовних операцій.

Таким чином, результати проведеного тестування та налагодження підтверджують працездатність розробленої інформаційної системи, коректність реалізації її функціональних можливостей та відповідність поставленим вимогам. Система демонструє стабільну роботу та готова до практичного використання.

Для більш наочного представлення результатів перевірки функціональних можливостей системи доцільно подати кілька тест-кейсів, що відображають основні сценарії її роботи.

Таблиця 3.1 – Основні тест-кейси інформаційної системи

№	Назва тесту	Вхідні дані / дії користувача	Очікуваний результат	Результат
1	Авторизація користувача	Введення коректного логіну та пароля	Успішний вхід у систему, відкриття головної сторінки	Успішно
2	Відмова в доступі при неправильному паролі	Введення правильного логіну та неправильного пароля	Система не надає доступ, відображається повідомлення про помилку	Успішно

Продовження таблиці 3.1

№	Назва тесту	Вхідні дані / дії користувача	Очікуваний результат	Результат
3	Додавання нового товару	Заповнення форми створення товару та збереження запису	Новий товар відображається у списку та зберігається в базі даних	Успішно
4	Виконання операції продажу	Створення операції продажу для вибраного товару	Кількість товару на складі зменшується, операція відображається в журналі	Успішно
5	Реєстрація дії в журналі аудиту	Редагування або видалення запису в системі	У журналі дій створюється новий запис із датою, користувачем і типом дії	Успішно

Подані тест-кейси підтверджують коректність реалізації основних функціональних можливостей системи. Результати перевірки свідчать про те, що програмне забезпечення стабільно виконує базові операції, забезпечує контроль доступу, коректно обробляє зміни даних та підтримує механізм аудиту дій користувачів.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досягнуто поставлену мету – розроблено веборієнтовану інформаційну систему складського та бухгалтерського обліку. У процесі роботи виконано всі поставлені завдання.

Проведено аналіз сучасного стану проблеми автоматизації складського та бухгалтерського обліку, у ході якого встановлено основні недоліки існуючих підходів, зокрема складність використання, обмежену гнучкість та надлишковий функціонал окремих систем.

Досліджено існуючі програмні рішення, проаналізовано їх переваги та недоліки, що дозволило визначити вимоги до власної інформаційної системи та обґрунтувати необхідність її розробки.

Сформовано функціональні та нефункціональні вимоги до розроблюваної системи, зокрема визначено необхідність реалізації модулів управління товарами, складами, операціями, платежами та користувачами, а також вимоги до безпеки та продуктивності.

Обґрунтовано вибір технологій та засобів реалізації програмного забезпечення. Для розробки було обрано платформу .NET Core, ASP.NET Core Web API для серверної частини, Blazor для клієнтської частини, а також SQL Server і Entity Framework Core для роботи з базою даних.

Розроблено архітектуру системи на основі клієнт-серверного підходу та спроектовано структуру бази даних, що включає таблиці користувачів, товарів, складів, операцій, платежів і журналу дій, а також визначено зв'язки між ними.

Реалізовано програмний продукт у вигляді клієнт-серверної веб-системи, що забезпечує взаємодію користувача з системою через веб-інтерфейс та обробку даних на сервері за допомогою REST API.

Забезпечено механізми автентифікації, авторизації та розмежування доступу користувачів шляхом реалізації перевірки облікових даних, використання ролей та обмеження доступу до функціональних можливостей системи.

Реалізовано функціональні модулі системи, зокрема управління товарами, складами, товарними операціями та платежами, що забезпечує автоматизацію обліку та контроль за рухом матеріальних і фінансових ресурсів.

Проведено тестування та налагодження системи, у ході якого перевірено коректність роботи функціональних модулів, обробку даних, реалізацію бізнес-логіки та механізмів безпеки, що підтвердило стабільність роботи програмного забезпечення.

Таким чином, у ході виконання кваліфікаційної роботи було розроблено повноцінну інформаційну систему складського та бухгалтерського обліку, яка забезпечує автоматизацію основних процесів, має зручний інтерфейс та відповідає сучасним вимогам до програмного забезпечення.

Практичне значення отриманих результатів полягає у можливості використання розробленої інформаційної системи для автоматизації процесів складського та бухгалтерського обліку на підприємствах малого та середнього бізнесу.

Подальший розвиток системи може бути спрямований на розширення функціональних можливостей, зокрема інтеграцію з іншими інформаційними системами, впровадження аналітичних модулів, використання технологій штучного інтелекту для прогнозування та оптимізації облікових процесів, а також підвищення рівня масштабованості та безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. World Bank. Digital Technologies in Education and Business. URL: <https://www.worldbank.org/en/topic/digitaldevelopment> (дата звернення: 26.04.2026).
2. European Commission. Digital Economy and Society Index (DESI). URL: <https://digital-strategy.ec.europa.eu/en/policies/desi> (дата звернення: 26.04.2026).
3. Fowler M. Patterns of Enterprise Application Architecture. URL: <https://martinfowler.com/eaCatalog/> (дата звернення: 26.04.2026).
4. OMG. Unified Modeling Language (UML) Specification Version 2.5.1. URL: <https://www.omg.org/spec/UML/2.5.1/> (дата звернення: 26.04.2026).
5. Microsoft. .NET documentation. URL: <https://learn.microsoft.com/dotnet> (дата звернення: 26.04.2026).
6. Microsoft. ASP.NET Core documentation. URL: <https://learn.microsoft.com/aspnet/core> (дата звернення: 26.04.2026).
7. Swagger. OpenAPI Specification. URL: <https://swagger.io/specification/> (дата звернення: 26.04.2026).
8. Microsoft. SQL Server documentation. URL: <https://learn.microsoft.com/sql> (дата звернення: 26.04.2026).
9. Microsoft. Entity Framework Core documentation. URL: <https://learn.microsoft.com/ef/core> (дата звернення: 26.04.2026).
10. Microsoft. Blazor documentation. URL: <https://learn.microsoft.com/aspnet/core/blazor> (дата звернення: 26.04.2026).
11. Microsoft. Web API documentation. URL: <https://learn.microsoft.com/aspnet/core/web-api> (дата звернення: 26.04.2026).