НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ Ігор БОЛБОТ	**_____ Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення для автоматизованого обліку інвентарю та матеріалів на підприємстві»

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи
к.т.н., с.н.с _____ **Юлія БОЯРІНОВА**
(підпис)

Виконав _____ **Андрій ЮМАНОВ**
(підпис)

КИЇВ – 2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

ЗАВДАННЯ ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Юманову Андрію Дмитровичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи
«Програмне забезпечення для автоматизованого обліку інвентарю та матеріалів на підприємстві»
затверджена наказом від «19» грудня 2025 р. № 3196 «С»

Термін подання завершеної роботи на кафедру «22» червня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи: опис процесів руху товарно-матеріальних цінностей на складах підприємства
Перелік питань, що підлягають дослідженню:
1. Дослідження предметної області та вимог
2. Проектування інформаційної частини
3. Проектування та розробка програмного забезпечення системи
4. Рекомендації щодо впровадження та експлуатації системи
Перелік графічних документів (за необхідності).

Дата видачі завдання « 19 » грудня 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Юлія БОЯРІНОВА**
(підпис)

Завдання взяв до виконання _____ **Андрій ЮМАНОВ**
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

DDL – Data Definition Language (мова визначення даних)

ER – Entity-Relationship (сутність-зв'язок)

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

SQL – Structured Query Language

UML – Unified Modeling Language

ПЗ – програмне забезпечення

СУБД – система управління базами даних

ТМЦ – товарно-матеріальні цінності

ВСТУП

Актуальність. Ефективна організація обліку матеріальних цінностей є однією з базових умов стабільної роботи будь-якого виробничого підприємства. Сировина, комплектувальні, інструмент, готова продукція та інші товарно-матеріальні цінності постійно переміщуються між складами, виробничими дільницями та контрагентами, і кожна така операція має бути своєчасно та достовірно зафіксована. Від точності цих даних безпосередньо залежить планування закупівель, собівартість продукції та фінансовий результат підприємства загалом.

На багатьох підприємствах облік досі ведеться у паперових відомостях або в розрізнених електронних таблицях, які заповнюються вручну. Такий підхід породжує низку системних проблем: дані дублюються між різними носіями, втрачається їхня узгодженість, а помилки оператора виявляються лише під час планової інвентаризації. Розбіжності між фактичними залишками на складі та обліковими записами призводять до простоїв виробництва через нестачу матеріалів або, навпаки, до заморожування коштів у надлишкових запасах.

Окремою складністю є відсутність єдиного джерела актуальної інформації про рух цінностей. Коли кілька працівників одночасно вносять зміни до різних копій облікових таблиць, керівництво не має змоги оперативно отримати достовірну картину наявних залишків. Ручне зведення даних з кількох джерел займає значний час і саме по собі стає джерелом нових помилок через людський фактор.

Наявні на ринку рішення здебільшого є частиною громіздких комплексних систем управління підприємством, упровадження яких потребує суттєвих витрат та тривалого навчання персоналу, або ж навпаки — обмежуються базовими функціями електронних таблиць без контролю цілісності даних. Це обґрунтовує доцільність розробки окремого програмного забезпечення, яке поєднує облік номенклатури, фіксацію операцій надходження та списання, контроль залишків і формування звітності у єдиному застосунку з прийнятним порогом входження.

Мета розробки. Метою розробки є автоматизація процесів обліку інвентарю та матеріалів на підприємстві шляхом створення програмного

забезпечення, що забезпечує ведення єдиної бази товарно-матеріальних цінностей, реєстрацію операцій їхнього руху та оперативний контроль складських залишків. Це дозволить скоротити частку ручної праці, підвищити достовірність облікових даних та надати відповідальним працівникам зручний інструмент для щоденної роботи.

Основна ідея полягає в заміні розрізнених паперових відомостей і неузгоджених електронних таблиць централізованою системою з єдиним сховищем даних. Розроблене програмне забезпечення дозволить зберігати повну номенклатуру матеріалів, документувати кожну операцію надходження, переміщення та списання, автоматично перераховувати поточні залишки й сигналізувати про досягнення мінімального рівня запасу. Завдяки цьому облік стане прозорим і відтворюваним, а ризик розбіжностей між фактичними та обліковими даними суттєво знизиться.

Методи та технології. Методологічною основою роботи є принципи реляційного моделювання даних, клієнт-серверної архітектури та сучасних підходів до побудови вебзастосунків із серверним відтворенням інтерфейсу. Як основну мову програмування серверної частини обрано Go, що забезпечує високу швидкодію, статичну типізацію та зручність розгортання у вигляді єдиного виконуваного файлу.

Для зберігання даних використано реляційну систему управління базами даних PostgreSQL, яка гарантує цілісність облікової інформації за рахунок транзакційності та механізмів обмежень. Логічну модель даних побудовано відповідно до вимог нормалізації, що унеможливлює дублювання та аномалії під час оновлення записів.

Користувацький інтерфейс реалізовано як вебдодаток на основі шаблонів Go Templates із застосуванням бібліотеки HTMX для динамічного оновлення окремих фрагментів сторінки без повного її перезавантаження. Стилізацію інтерфейсу виконано засобами фреймворку Bootstrap, що забезпечує адаптивне відображення та звичний для користувача вигляд елементів керування. Такий підхід дозволяє зосередити бізнес-логіку на стороні сервера та спростити підтримку застосунку порівняно з повноцінними клієнтськими фреймворками.

Структура записки. Пояснювальна записка містить вступ, чотири основні розділи та висновки. До записки також входять перелік використаних джерел і додатки з графічним матеріалом. Загальний обсяг записки — 61 сторінка основного тексту, кількість використаних джерел — 21, кількість додатків — 3.

У першому розділі «Облік інвентарю та матеріалів як об'єкт дослідження» розглянуто особливості складського обліку як предметної області, змодельовано основні процеси за допомогою UML-діаграм, проаналізовано наявні рішення, а також сформульовано завдання, яке покликана вирішити розроблена система.

У другому розділі «Проєктування інформаційного забезпечення» побудовано логічну модель даних у формі ER-діаграми, обґрунтовано вибір системи управління базами даних, спроєктовано структуру таблиць інформаційної бази та її фізичну організацію.

У третьому розділі «Розробка програмного забезпечення» висвітлено етапи побудови прикладної частини системи: обґрунтовано вибір технологічного стеку, описано організаційну та компонентну архітектуру застосунку, окремо розглянуто алгоритмізацію ключових модулів та фрагменти програмної реалізації.

Четвертий розділ «Рекомендації щодо впровадження та експлуатації системи» охоплює функціональне тестування реалізованих модулів, вимоги до апаратного й програмного забезпечення вузлів системи, перелік артефактів інсталяційного пакета та практичні поради щодо розгортання застосунку у робочому середовищі.

1 ОБЛІК ІНВЕНТАРЮ ТА МАТЕРІАЛІВ ЯК ОБ'ЄКТ ДОСЛІДЖЕННЯ

1.1 Аналіз складського обліку як предметної області

Під складським обліком аналізують комплексну підсистему безперервної реєстрації, контролю та моніторингу матеріально-технічних ресурсів на підприємстві. До категорії товарно-матеріальних цінностей (ТМЦ) традиційно відносять сировинні бази, основні й допоміжні матеріали, комплектуючі деталі, інструментарій, паливні ресурси, тару, запасні частини для обладнання, а також кінцеву продукцію, яка зберігається на складах до моменту її реалізації чи відвантаження замовнику. Організація точного та оперативного обліку цих засобів є критично важливою умовою для забезпечення стабільного виробничого циклу, а також виступає інформаційним підґрунтям для менеджменту під час планування закупівель і оптимізації загальних витрат [1].

Об'єктом обліку виступає номенклатура — структурований перелік усіх позицій ТМЦ, кожна з яких має унікальний код, найменування, одиницю виміру та належність до певної облікової групи. Кількісний стан кожної позиції на конкретний момент часу визначається як залишок, що зберігається у розрізі складів та матеріально відповідальних осіб. Саме показник залишку є ключовим результатом обліку, на який спираються всі подальші господарські операції.

Рух матеріальних цінностей на складі описується кількома типовими операціями. Надходження (оприбуткування) фіксує приймання цінностей від постачальника або з власного виробництва та збільшує залишок відповідних позицій. Відпуск (списання) відображає видачу матеріалів у виробництво, реалізацію готової продукції або вибуття цінностей з інших причин і зменшує залишок. Внутрішнє переміщення фіксує передачу цінностей між складами або між матеріально відповідальними особами без зміни загального обсягу запасів підприємства. Окремою періодичною процедурою є інвентаризація — звіряння

фактичної наявності цінностей з обліковими даними з подальшим виявленням і документуванням розбіжностей [2].

Традиційно складський облік на багатьох підприємствах ведеться у паперових картках складського обліку та журналах або у розрізнених електронних таблицях, які заповнюються вручну. Такий підхід має низку істотних недоліків. Дані дублюються між різними носіями, через що втрачається їхня узгодженість, а помилка, припущена під час ручного введення, виявляється лише під час планової інвентаризації. Відсутність єдиного джерела актуальної інформації унеможливлює оперативне отримання достовірної картини залишків: коли кілька працівників одночасно вносять зміни до різних копій облікових документів, звести їх у цілісний стан стає окремою трудомісткою задачею.

Наслідком неточного обліку є прямі господарські втрати. Розбіжності між фактичними та обліковими залишками призводять або до зупинки виробничих процесів через несподівану нестачу матеріалів, або, навпаки, до заморожування обігових коштів у надлишкових запасах. Несвоєчасне виявлення дефіциту критичних позицій змушує здійснювати позапланові термінові закупівлі за завищеними цінами. Усе це робить ручний облік джерелом фінансових ризиків, що зростають разом з обсягами діяльності підприємства.

З огляду на викладене, автоматизація складського обліку є актуальним завданням. Перенесення обліку до спеціалізованої програмної системи із централізованим сховищем даних дозволяє забезпечити узгодженість і цілісність інформації, автоматичний перерахунок залишків після кожної операції, контроль мінімального рівня запасів та оперативне формування звітності. Це знижує вплив людського фактору, скорочує час на рутинні дії та підвищує загальну керованість матеріальних потоків підприємства.

1.2 Моделювання предметної області

Текстовий опис бізнес-процесів не завжди спроможний чітко окреслити всі нюанси та приховані суперечності у вимогах до майбутньої системи, що може викликати непорозуміння між розробниками та замовником. Для мінімізації цих

ризиків використовують методи формального моделювання — представлення архітектури предметної області у вигляді наочних графічних діаграм, які декомпонують систему на рівні користувачів, їхніх обов'язків та алгоритмів взаємодії [3].

Для концептуального опису контуру складського обліку було обрано уніфіковану мову моделювання UML (Unified Modeling Language), яка є загальноприйнятим стандартом у сфері інженерії програмного забезпечення [4]. У межах дослідження розроблено три типи моделей: діаграму прецедентів (визначає функціональний обсяг системи з позиції користувача), діаграму активності (ілюструє логічну послідовність операцій руху ТМЦ) та діаграму класів (описує статичну структуру даних та зв'язки між сутностями). Детальний аналіз кожної моделі наведено нижче.

1.2.1 Діаграма прецедентів

Діаграма прецедентів (use case diagram) фокусується на описі функціональних кордонів системи з точки зору кінцевих користувачів [5]. Вона демонструє не внутрішній алгоритм реалізації функцій, а безпосередньо сам перелік можливостей (сервісів), які система надає кожному учаснику процесу. Завдяки високому рівню абстракції цю діаграму створюють на початкових стадіях проектування для формування єдиного погляду на рамки розробки.

Ключовими конструктами цієї моделі виступають актори (зовнішні сутності) та прецеденти (варіанти використання). Під актором розуміють будь-якого зовнішнього суб'єкта (роль працівника або суміжну систему), який ініціює взаємодію з програмним комплексом. Прецедент є описом завершеної послідовності дій, що приносить корисний результат користувачеві. Взаємозв'язки між цими елементами моделюються лініями асоціації, а внутрішня логіка відносин між прецедентами може уточнюватися за допомогою відношень включення (<<include>>) та розширення (<<extend>>), що дозволяє оптимізувати структуру моделі [5]. На рис. 1 зображено розроблену діаграму прецедентів.

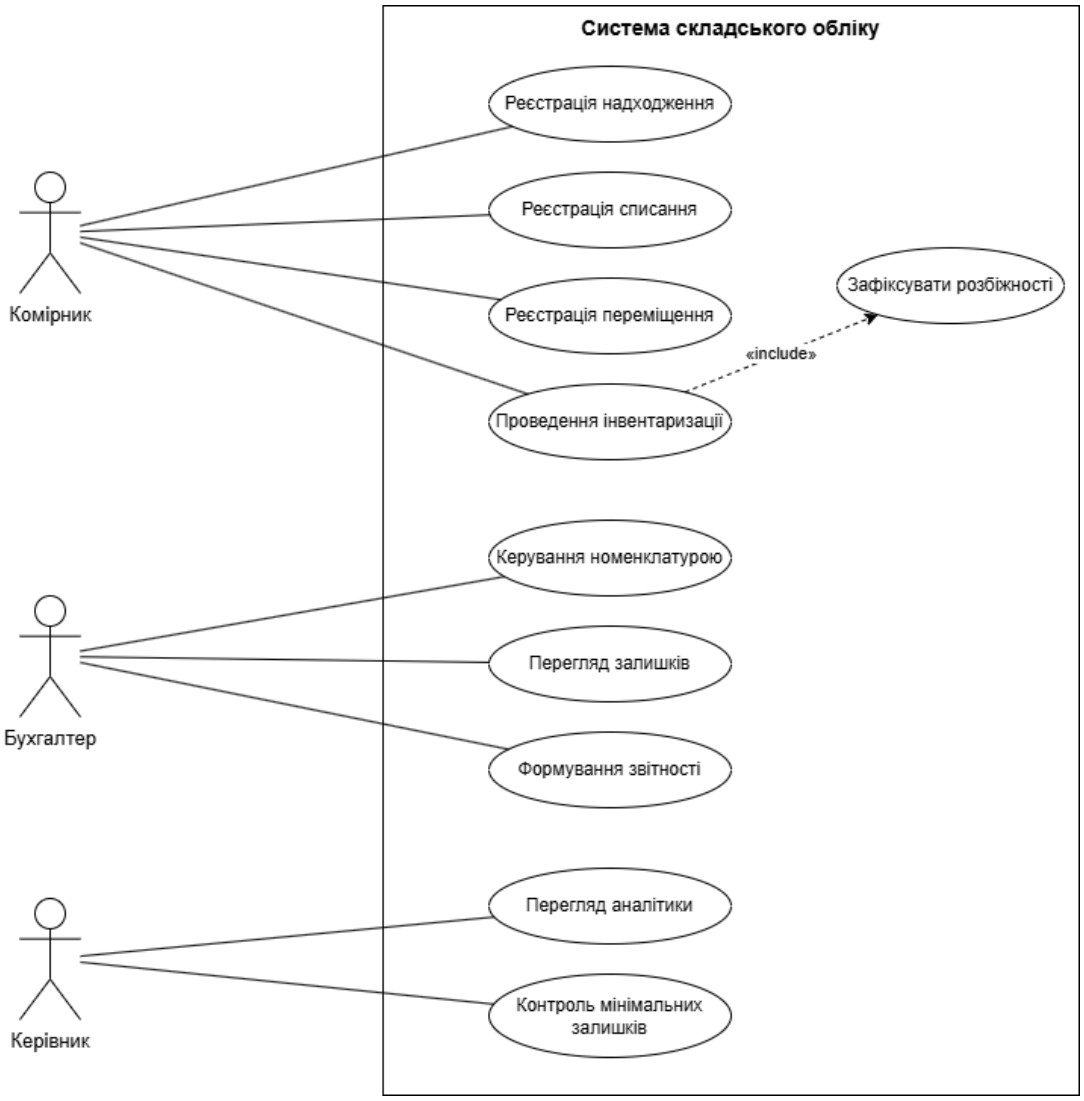


Рис. 1 Діаграма прецедентів системи складського обліку

Специфікація акторів системи

Виокремлено трьох ключових акторів, які взаємодіють із системою:

- **Комірник (комірник складу)** — матеріально відповідальна особа, яка безпосередньо здійснює приймання, відпуск та переміщення цінностей і вносить відповідні операції до системи;
- **Бухгалтер** — працівник, що веде облік номенклатури, контролює коректність проведених операцій та формує облікову звітність;
- **Керівник (керівник складу)** — особа, що приймає управлінські рішення на основі звітних даних про залишки та рух цінностей і має доступ переважно до перегляду й аналітики.

Опис прецедентів

Комірник:

- реєстрація надходження — оприбуткування цінностей, що надійшли, зі збільшенням залишку відповідних позицій;
- реєстрація списання — оформлення відпуску цінностей у виробництво чи реалізацію зі зменшенням залишку;
- реєстрація переміщення — фіксація передачі цінностей між складами або відповідальними особами;
- проведення інвентаризації — внесення фактичних залишків та звіряння їх з обліковими (з включенням прецеденту «зафіксувати розбіжності»).
- Бухгалтер:
- керування номенклатурою — додавання, редагування та групування позицій довідника ТМЦ;
- перегляд залишків — отримання поточного стану запасів у розрізі складів і позицій;
- формування звітності — побудова звітів про рух та залишки цінностей за обраний період.
- Керівник:
- перегляд аналітики — ознайомлення зі зведеними показниками руху та залишків;
- контроль мінімальних залишків — отримання переліку позицій, запас яких опустився нижче встановленого порога.

Окреслена діаграма прецедентів наочно диференціює зони відповідальності в системі та формує базис для проектування алгоритмів і архітектури бази даних

1.2.2 Діаграма активності

Діаграма активності (activity diagram) у стандарті UML слугує для деталізованого представлення динамічних аспектів системи, описуючи потоки керування та послідовність дій у межах конкретного бізнес-процесу [5]. Вона акцентує увагу на логіці переходів від однієї операції до іншої, що є вкрай

важливим для розробників ПЗ. Модель будується за допомогою початкових і фінальних вузлів, блоків дій (прямокутники із закругленими кутами), вузлів розгалуження (ромби прийняття рішень) та орієнтованих стрілок, що задають напрямок виконання процесу.

Як центральний бізнес-процес для моделювання було обрано алгоритм реєстрації руху ТМЦ. Розроблену діаграму наведено на рис. 2. Вона покриває повний життєвий цикл документа — від моменту вибору користувачем типу операції до фінального оновлення записів у базі даних, враховуючи всі етапи перевірок та обробки помилок.

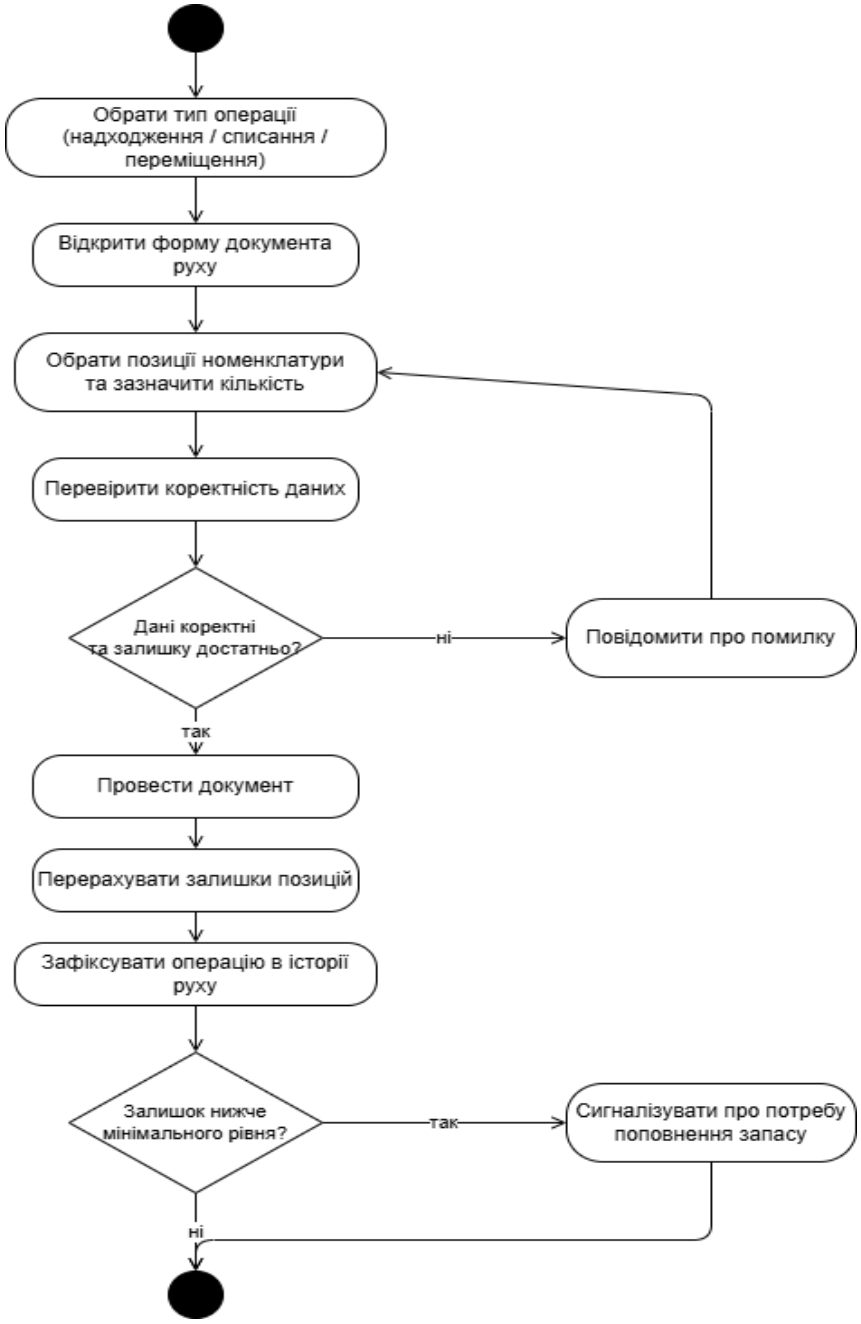


Рис. 2 Діаграма активності процесу реєстрації руху цінностей

Опис діаграми

Процес реєстрації руху цінностей:

- користувач обирає тип операції (надходження, списання або переміщення);
- система відкриває відповідну форму документа руху;
- користувач обирає позиції номенклатури та зазначає їхню кількість;
- система перевіряє коректність даних: для списання та переміщення контролюється достатність поточного залишку;
- якщо залишку недостатньо або дані некоректні — система повідомляє про помилку, і користувач повертається до коригування документа;
- за умови успішної перевірки документ проводиться, а залишки відповідних позицій автоматично перераховуються;
- система фіксує операцію в історії руху та, за потреби, перевіряє досягнення мінімального рівня запасу для списаних позицій.

Явне виокремлення альтернативних шляхів виконання (помилки валідації, нестача залишку, повторне коригування) робить діаграму активності зручним джерелом для формулювання сценаріїв нормального та виняткового перебігу подій у програмній системі. Саме ділянки, на яких можлива помилка ручного введення, є першочерговими кандидатами на автоматизований контроль з боку системи.

1.2.3 Діаграма класів

Діаграма класів (class diagram) є структурною діаграмою UML і призначена для опису статичної структури системи — переліку класів, їхніх атрибутів та зв'язків між ними [4]. На етапі аналізу предметної області діаграма класів використовується для виявлення основних сутностей, якими оперує система, та характеру відношень між ними, що згодом стає підґрунтям для проєктування структури бази даних. Розроблену діаграму класів предметної області наведено на рис. 3.

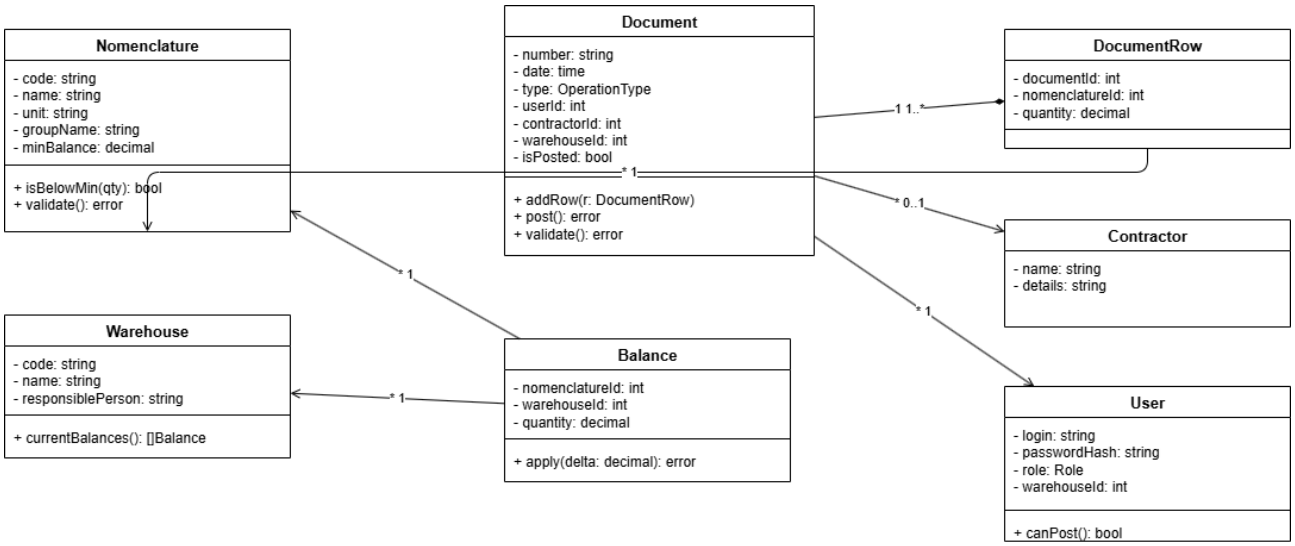


Рис. 3 Діаграма класів предметної області

Опис основних класів

- **Номенклатура** — позиція довідника ТМЦ; атрибути: код, найменування, одиниця виміру, облікова група, мінімальний залишок;
- **Склад** — місце зберігання цінностей; атрибути: код, найменування, відповідальна особа;
- **Залишок** — кількість конкретної позиції номенклатури на конкретному складі; атрибути: позиція, склад, кількість;
- **Документ руху** — зафіксована господарська операція; атрибути: номер, дата, тип (надходження / списання / переміщення), відповідальна особа;
- **Рядок документа** — окрема позиція у складі документа руху; атрибути: документ, позиція номенклатури, кількість;
- **Контрагент** — постачальник або отримувач цінностей; атрибути: найменування, реквізити;
- **Користувач** — обліковий запис особи, що працює в системі; атрибути: логін, роль, відповідальний склад.

Між класами встановлено такі зв'язки: документ руху складається з кількох рядків (відношення композиції); кожен рядок документа посилається на одну позицію номенклатури; залишок пов'язує позицію номенклатури з конкретним складом; документ руху належить одному користувачеві та, для операцій надходження чи реалізації, одному контрагентові. Така структура без

надмірності описує предметну область і безпосередньо відображається на реляційну модель даних, що проєктується у другому розділі.

1.3 Огляд існуючих аналогічних систем

Завдання автоматизації складського обліку не є новим, тому на ринку представлено значну кількість програмних рішень різного масштабу — від комплексних систем управління підприємством до спеціалізованих застосунків для малого та середнього бізнесу. Для аналізу обрано три представницькі системи: BAS Управління торгівлею, Zoho Inventory та inFlow Inventory. Далі по черзі розглянуто кожне рішення з виокремленням переваг і недоліків з погляду задач складського обліку на виробничому підприємстві.

BAS Управління торгівлею

BAS Управління торгівлею — українське програмне рішення для автоматизації оперативного обліку торговельного підприємства, створене на платформі Business Automation Software як заміна рішень сімейства «1С:Підприємство» після відмови від російського програмного забезпечення [6]. Система охоплює управління запасами, складський облік, закупівлі, продажі та аналіз діяльності. Складська підсистема підтримує ордерну та неордерну схеми обліку, облік цінностей у розрізі серій і термінів придатності, розміщення товарів за комірками та контроль залишків [7]. На рис. 4 наведено інтерфейс системи.

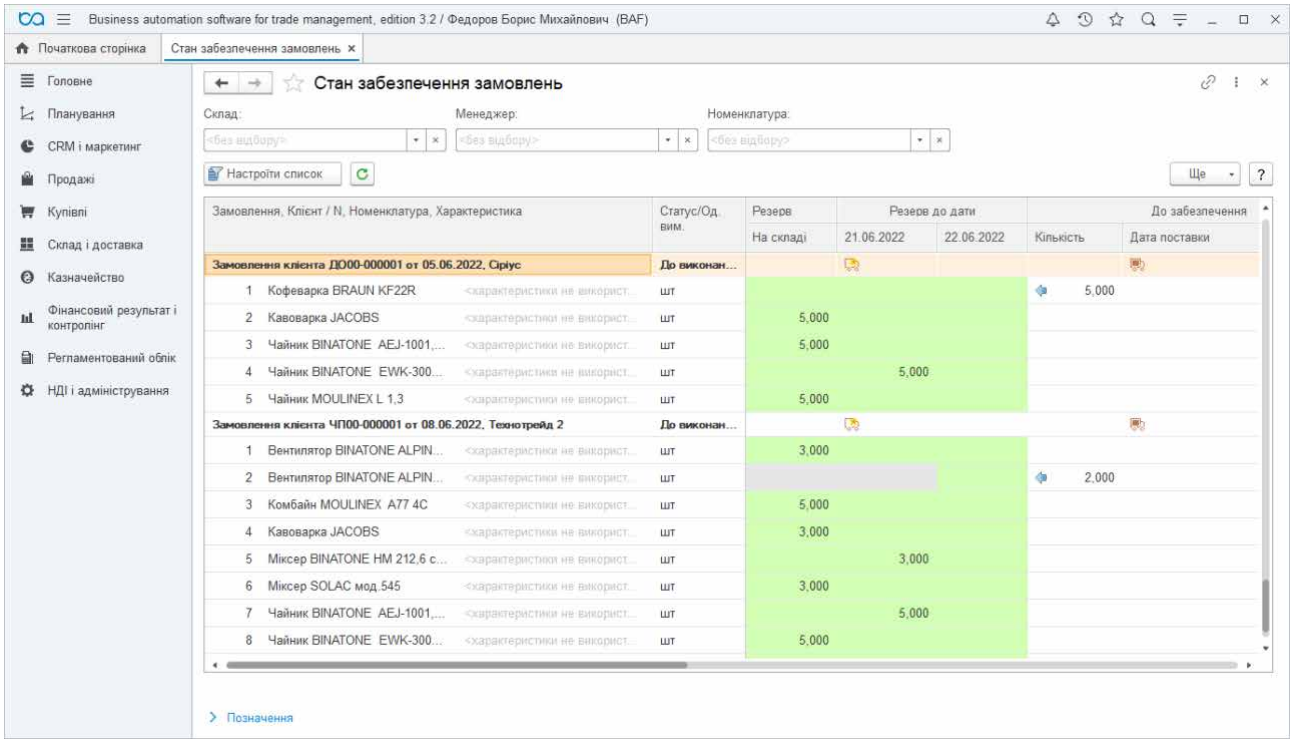


Рис. 4 Інтерфейс системи BAS Управління торгівлею

Переваги*:*

- комплексне охоплення обліку — склад, закупівлі, продажі та фінанси в одній системі;
- відповідність українському законодавству та регламентованому обліку;
- гнучке налаштування схем складського обліку під потреби підприємства;
- підтримка торгового обладнання (сканери штрих-кодів, термінали збору даних).

Недоліки*:*

- значна складність та надлишковість функціоналу для підприємства, якому потрібен лише складський облік;
- висока сукупна вартість володіння — ліцензії, впровадження та супровід;
- потреба у тривалому навчанні персоналу та, як правило, у залученні зовнішніх спеціалістів для налаштування.

Zoho Inventory

Zoho Inventory — хмарна система управління запасами для малого та середнього бізнесу, що входить до екосистеми сервісів Zoho. Система надає облік номенклатури, керування кількома складами з переміщенням цінностей між ними, відстеження залишків у режимі реального часу, сповіщення про досягнення мінімального рівня запасу та формування звітів у розрізі окремих складів [8]. Підтримується серійний та партійний облік, штрих-кодування, а також розмежування доступу користувачів до даних конкретних складів [9]. На рис. 5 наведено інтерфейс системи.

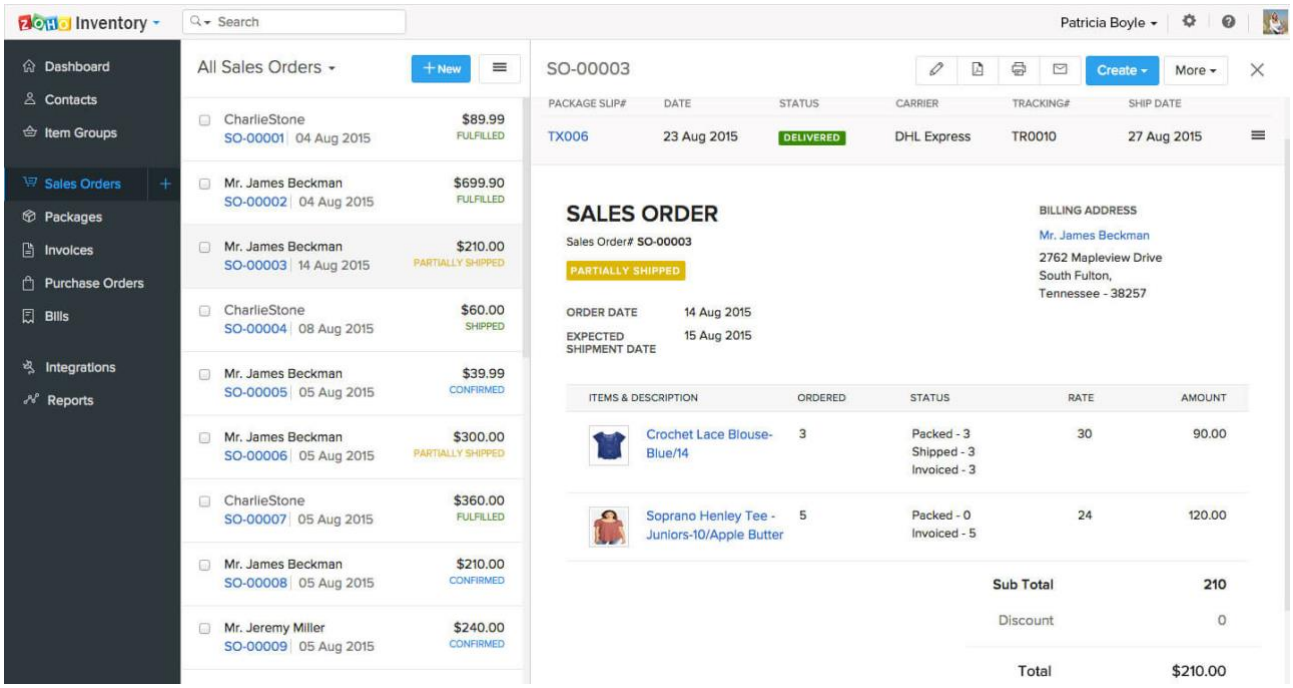


Рис. 5 Інтерфейс системи Zoho Inventory

Переваги:

- хмарний доступ із будь-якого пристрою без встановлення програмного забезпечення;
- керування кількома складами з переміщеннями та окремою звітністю;
- автоматичні сповіщення про низький рівень запасів і автоматичне дозамовлення;
- розмежування прав доступу користувачів до даних окремих складів.

Недоліки:

- орієнтація передусім на торгівлю та електронну комерцію, а не на виробничий облік матеріалів;

- залежність від хмарного провайдера та постійного підключення до мережі;
- обмеження безкоштовного плану (кількість замовлень, користувачів та складів) і прив'язка до платної екосистеми Zoho;
- зберігання даних підприємства на зовнішніх серверах, що не завжди прийнятно з міркувань безпеки.

inFlow Inventory

inFlow Inventory — система управління запасами для малого та середнього бізнесу, що пропонується у хмарному варіанті, а також у вигляді мобільного застосунку (підтримку локальної версії inFlow v3 виробник припинив у 2024 році) [10]. Система забезпечує облік позицій у розрізі кількох складів, штрих-кодування та сканування, відстеження повної історії руху кожної позиції із зазначенням відповідальної особи, серійний та партійний облік, а також гнучку звітність про рух запасів і закупівлі [11]. Передбачено налаштування ролей і прав доступу для членів команди [12]. На рис. 6 наведено інтерфейс системи.

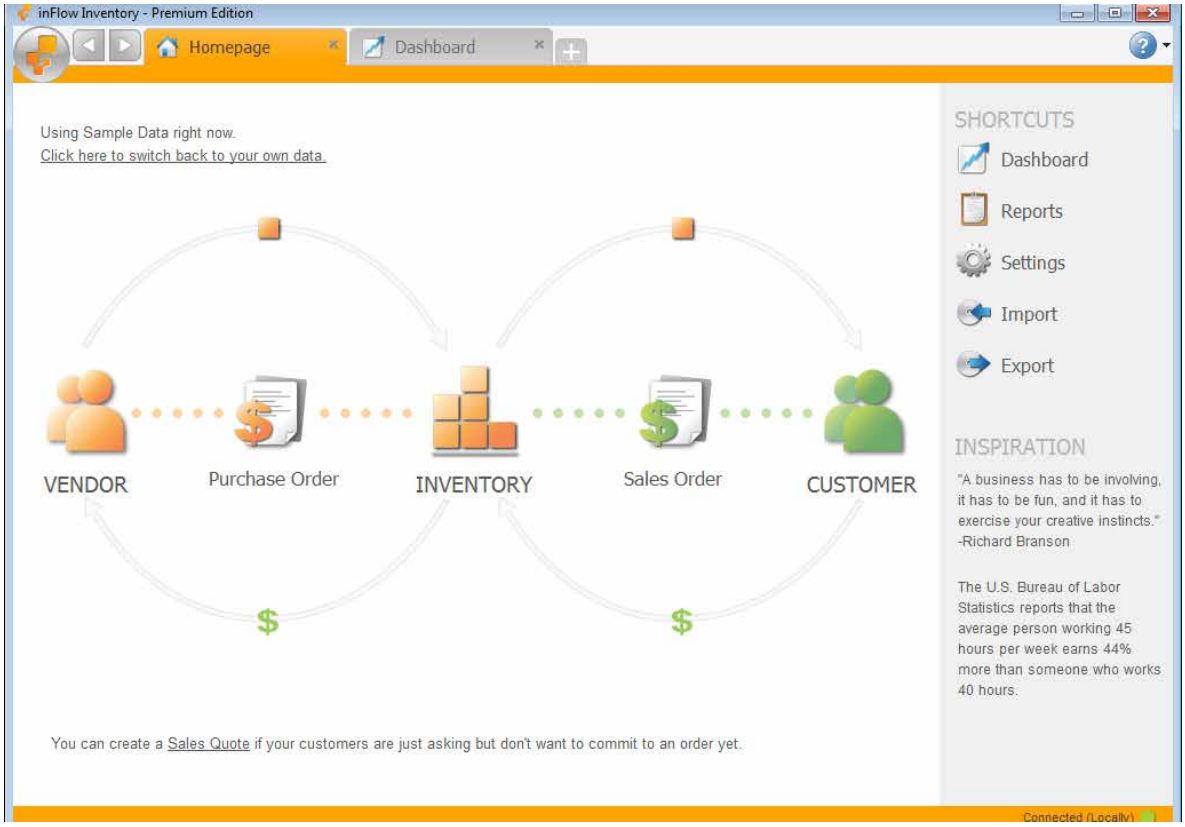


Рис. 6 Інтерфейс системи inFlow Inventory

Переваги:

- простий і зрозумілий інтерфейс із низьким порогом входження;
- повна історія руху кожної позиції із фіксацією відповідальної особи;
- підтримка штрих-кодування та мобільного доступу до складу;
- налаштування ролей і прав доступу користувачів.
- *Недоліки:*
- припинення підтримки локальної версії — фактичний перехід виключно у хмару;
- частина потрібних функцій (зокрема пошук за серійними номерами) доступна лише за додаткову плату;
- обмежені можливості звітності та налаштування під специфіку конкретного виробництва, що відзначають користувачі;
- відсутність повноцінної української локалізації та врахування вимог вітчизняного обліку.

Проведений аналіз показує, що наявні рішення або є надлишково складними та дорогими для підприємства, якому потрібен передусім облік матеріалів (BAS Управління торгівлею), або орієнтовані на торгівлю та електронну комерцію і прив'язані до зовнішньої хмарної екосистеми (Zoho Inventory, inFlow Inventory). Спільними обмеженнями хмарних систем є зберігання облікових даних на зовнішніх серверах та залежність від платних тарифних планів. Це обґрунтовує доцільність розробки власного програмного забезпечення, що зосереджене саме на складському обліку матеріалів, розгортається на власному сервері підприємства та має простий вебінтерфейс із розмежуванням ролей користувачів.

1.4 Постановка завдання

Мета даного дипломного проєкту - це розробка програмного забезпечення для автоматизованого обліку інвентарю та матеріалів на підприємстві. Система має забезпечувати ведення єдиного довідника товарно-матеріальних цінностей, реєстрацію операцій їхнього руху, автоматичний контроль залишків та

формування звітності. Розмежування прав доступу між користувачами з різними ролями має гарантувати коректність і захищеність облікових даних.

Вхідні дані

Система повинна обробляти такі вхідні дані:

- дані довідника номенклатури: код, найменування, одиниця виміру, облікова група, мінімальний рівень залишку;
- дані складів та матеріально відповідальних осіб: найменування складу, відповідальна особа;
- дані контрагентів: найменування постачальника або отримувача, реквізити;
- дані документів руху: тип операції, дата, перелік позицій з кількістю, склад, контрагент;
- дані облікових записів користувачів: логін, роль, закріплений склад;
- параметри інвентаризації: фактичні залишки позицій, зафіксовані під час перерахунку.

Вихідні дані системи

Система повинна формувати такі результати:

- поточні залишки цінностей у розрізі складів і позицій номенклатури;
- історію руху кожної позиції із зазначенням операцій, дат та відповідальних осіб;
- відомість результатів інвентаризації з виявленими розбіжностями між фактичними та обліковими залишками;
- перелік позицій, залишок яких опустився нижче встановленого мінімального рівня;
- зведені звіти про надходження та списання цінностей за обраний період.

Функціональні вимоги до системи

Автентифікація та розмежування доступу. Система повинна підтримувати вхід користувачів за обліковими даними та розмежовувати доступні дії відповідно до ролі (комірник, бухгалтер, керівник). Паролі користувачів мають зберігатися у захищеному (хешованому) вигляді.

Керування номенклатурою. Користувач із відповідними правами повинен мати можливість додавати, редагувати та групувати позиції довідника ТМЦ, а також задавати для них мінімальний рівень залишку.

Реєстрація операцій руху. Система повинна забезпечувати оформлення документів надходження, списання та внутрішнього переміщення цінностей з автоматичним перерахунком залишків після проведення документа.

Контроль залишків. Під час списання та переміщення система повинна перевіряти достатність поточного залишку й не допускати проведення операції за його нестачі.

Інвентаризація. Система повинна підтримувати внесення фактичних залишків та автоматичне порівняння їх з обліковими з формуванням відомості розбіжностей.

Контроль мінімального рівня запасів. Система повинна автоматично виявляти позиції, залишок яких опустився нижче заданого порога, та сигналізувати про потребу їх поповнення.

Звітність. Система повинна формувати звіти про залишки, рух цінностей та результати інвентаризації за обраний період.

Нефункціональні вимоги до системи

Безпека. Доступ до даних має бути обмежений відповідно до ролі користувача. Паролі зберігаються у хешованому вигляді, а обмін даними між клієнтом і сервером відбувається захищеним каналом.

Цілісність даних. Операції, що змінюють залишки, повинні виконуватися в межах транзакцій, що гарантує узгодженість облікових даних навіть у разі збою під час виконання операції.

Зручність використання. Інтерфейс має бути зрозумілим для працівників складу з мінімальним технічним досвідом; основні дії повинні бути доступними з головного меню та виконуватися без перезавантаження сторінки.

Продуктивність. Система повинна забезпечувати прийнятний час відгуку під час роботи з довідником номенклатури та документами руху за типового обсягу даних підприємства.

Надійність. Система повинна коректно обробляти некоректне введення даних та помилки, не допускаючи порушення цілісності облікової інформації.

Розгортання. Серверна частина повинна розгортатися на віддаленому сервері підприємства й бути доступною користувачам через вебоглядач без встановлення додаткового програмного забезпечення на робочих місцях.

Запропонована програмна система орієнтована на автоматизацію типових операцій складського обліку матеріалів на виробничому підприємстві. Завдяки централізованому зберіганню даних, автоматичному контролю залишків та розмежуванню ролей користувачів програмне забезпечення стане ефективним інструментом для підвищення достовірності обліку та зменшення впливу людського фактору на його результати.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

На етапі логічного проєктування формується концептуальна структура бази даних, яка чітко відображає ключові об'єкти предметної області, їхні внутрішні властивості та архітектуру взаємозв'язків. Важливою особливістю такої моделі є її незалежність від фізичного середовища: спроєктована схема може бути успішно імплементована засобами будь-якої реляційної системи управління базами даних (СУБД). Візуальним інструментом для представлення цієї структури виступає ER-діаграма (від англ. Entity-Relationship — «сутність-зв'язок»). У даній графічній нотації інформаційні об'єкти (сутності) подаються як прямокутні блоки з переліком атрибутів, а залежності між ними ілюструються лініями, що визначають характер і кратність зв'язку [10].

Спираючись на результати аналізу предметної області, проведеного у попередньому розділі, для розроблюваної системи обліку матеріалів та інвентарю було визначено **сім ключових сутностей**:

- **Користувачі системи** — облікові записи персоналу, що працює з програмою;
- **Склади — довідник місць зберігання цінностей;**
- **Контрагенти** — зовнішні суб'єкти (постачальники, клієнти);
- **Номенклатура** — генеральний каталог товарно-матеріальних цінностей;
- **Поточні залишки** — регістр актуального кількісного стану запасів;
- **Документи руху** — зафіксовані факти складських операцій;
- **Рядки документів** — деталізовані специфікації до кожної транзакції.

Для гарантування суворої посилальної цілісності інформації на рівні бази даних усі взаємозв'язки між наведеними таблицями проєктуються із застосуванням механізму зовнішніх ключів (Foreign Keys).ER-діаграму наведено на рис. 7.

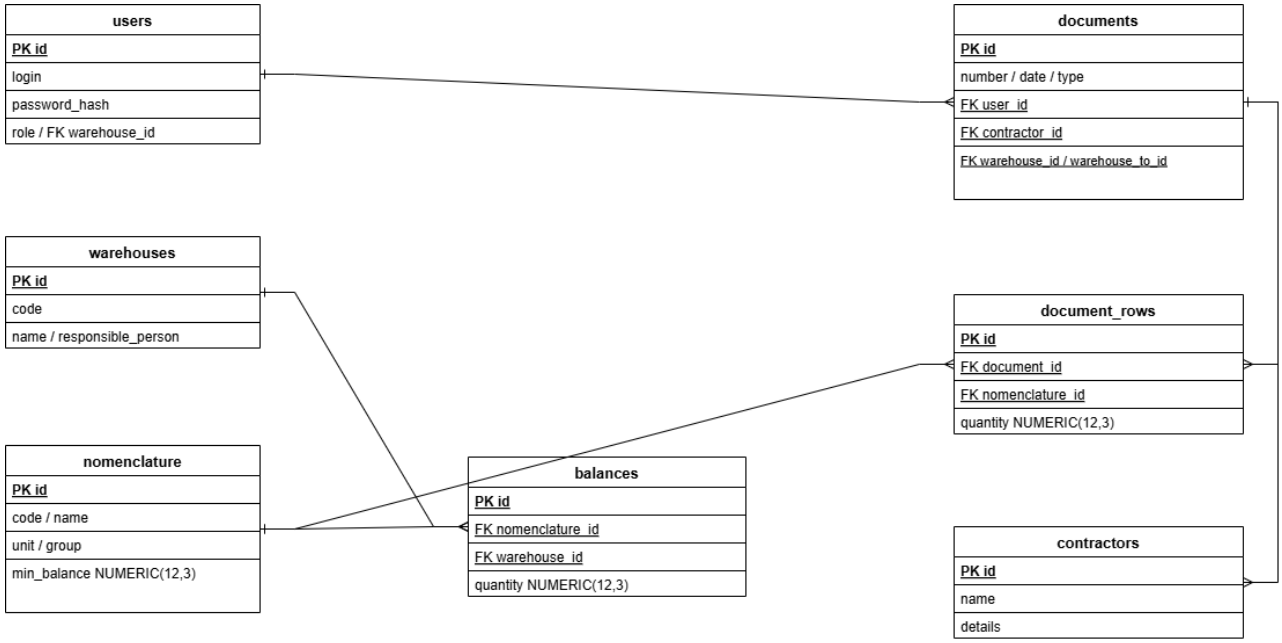


Рис. 7 Логічна модель даних у вигляді ER-діаграми

Опис сутностей та атрибутів

users (Користувач). Атрибути: id (первинний ключ), login, password_hash (хеш пароля), role (роль — комірник, бухгалтер, керівник), warehouse_id (закріплений склад), created_at.

warehouses (Склад). Атрибути: id, code, name, responsible_person (матеріально відповідальна особа).

contractors (Контрагент). Атрибути: id, name (найменування постачальника або отримувача), details (реквізити).

nomenclature (Номенклатура). Атрибути: id, code, name, unit (одиниця виміру), group_name (облікова група), min_balance (мінімальний залишок).

balances (Залишок). Атрибути: id, nomenclature_id, warehouse_id, quantity (поточна кількість, дробова). Пара «номенклатура + склад» унікальна.

documents (Документ руху). Атрибути: id, number, doc_date, type (тип операції), user_id, contractor_id, warehouse_id (основний склад), warehouse_to_id (склад-отримувач для переміщення), is_posted (ознака проведення).

document_rows (Рядок документа). Атрибути: id, document_id, nomenclature_id, quantity.

Опис зв'язків

- один користувач формує багато документів (1:N), через зовнішній ключ user_id у documents;
- один склад має багато записів залишків (1:N), через warehouse_id у balances;
- одна позиція номенклатури має багато записів залишків на різних складах (1:N), через nomenclature_id у balances;
- один документ складається з багатьох рядків (1:N), через document_id у document_rows; рядок не існує без документа;
- одна позиція номенклатури згадується в багатьох рядках документів (1:N), через nomenclature_id у document_rows;
- один контрагент може бути пов'язаний з багатьма документами (1:N, необов'язковий — для внутрішнього переміщення контрагент відсутній).

Обґрунтування відповідності третій нормальній формі

Розроблена модель відповідає вимогам реляційності та третій нормальній формі (3НФ), що доводиться послідовно.

Перша нормальна форма (1НФ). Усі атрибути сутностей атомарні: кожне поле зберігає одне неподільне значення, повторювані групи відсутні. Зокрема, перелік позицій документа винесено в окрему сутність document_rows, а не зберігається списком усередині документа.

Друга нормальна форма (2НФ). Модель перебуває в 1НФ, а кожен неключовий атрибут повністю залежить від первинного ключа своєї сутності. Усі сутності використовують простий первинний ключ id, тому часткова залежність від частини складеного ключа неможлива.

Третя нормальна форма (3НФ). Модель перебуває у 2НФ і не містить транзитивних залежностей — жоден неключовий атрибут не залежить від іншого неключового. Наприклад, у сутності documents не зберігаються найменування

контрагента чи назва складу: замість них використано зовнішні ключі contractor_id та warehouse_id, а самі найменування зберігаються одноразово у відповідних сутностях. Таким чином усувається надмірність і виключаються аномалії оновлення.

Окремо зазначимо, що зберігання поточного залишку в окремій сутності balances є контрольованим винятком на користь швидкодії: значення quantity є похідним від суми проведених операцій, але зберігається явно, щоб уникнути обчислення залишку складним агрегатним запитом при кожному читанні. Узгодженість цього значення з документами руху гарантується тригером бази даних, описаним у підрозділі 2.3.

2.2 Вибір системи управління базами даних

Оскільки логічна модель є реляційною (відповідає 3НФ, дані мають плоску структуру з чіткими зв'язками за зовнішніми ключами), для її реалізації обрано реляційну систему управління базами даних. Реляційна СУБД зберігає дані у вигляді таблиць із фіксованою структурою, де зв'язок між сутностями реалізується через відповідність значень ключів, та забезпечує цілісність даних обмеженнями (первинні та зовнішні ключі, унікальність, перевірки CHECK) і транзакційність операцій.

Розглянуто два поширені вільні рішення цього класу — PostgreSQL та MySQL. Порівняльну характеристику наведено в таблиці 2.1.

Таблиця 2.1

Порівняння реляційних СУБД

Параметр	**PostgreSQL**	**MySQL**
Тип моделі	Реляційна, об'єктно-реляційна	Реляційна
Підтримка обмежень цілісності	Повна (CHECK, FK, UNIQUE)	Повна

Параметр	PostgreSQL	MySQL
Транзакції та ізоляція	Повна підтримка ACID	Повна (InnoDB)
Тригери та збережені функції	PL/pgSQL, інші мови	Обмеженіша мова
Точні числові типи	NUMERIC довільної точності	DECIMAL
Ліцензія	Вільна (PostgreSQL License)	Вільна (GPL) / комерційна

За результатами аналізу основною СУБД проєкту обрано PostgreSQL. Вибір обґрунтовано такими міркуваннями.

Точні числові типи. Тип NUMERIC у PostgreSQL зберігає числа з довільною заданою точністю без похибки округлення. Це критично для системи, де кількість матеріалів обліковується дробовими значеннями (кілограми, метри, літри): кількість і залишки оголошено як NUMERIC(12,3), що гарантує точність до тисячних часток одиниці виміру.

Засоби забезпечення цілісності. PostgreSQL підтримує повний набір обмежень — зовнішні ключі з каскадними діями, унікальні обмеження та перевірки CHECK. Зокрема, перевірка CHECK (quantity >= 0) на таблиці залишків не дозволяє провести списання понад наявну кількість на рівні самої бази, незалежно від логіки застосунку.

Збережені функції та тригери. Процедурна мова PL/pgSQL дозволяє реалізувати автоматичний перерахунок залишків при проведенні документа безпосередньо в базі даних у вигляді тригера, що гарантує узгодженість даних незалежно від того, який саме компонент системи ініціював операцію.

Транзакційність. Повна підтримка ACID гарантує, що операція проведення документа виконується атомарно: або всі залишки перераховуються, або не змінюється нічого. Це унеможливлює часткове оновлення даних у разі збою.

Відкритість та вартість. PostgreSQL поширюється за вільною ліцензією без ліцензійних відрахувань, що відповідає вимозі економічної доцільності та можливості розгортання у власному периметрі підприємства.

2.3 Розробка структури інформаційної бази

На цьому етапі логічну модель реалізовано у вигляді фізичної структури бази даних PostgreSQL. Інформаційна база складається з таблиць, обмежень цілісності, індексів для прискорення типових запитів та тригерної функції для підтримки узгодженості залишків. Реалізацію цих компонентів розкрито в пунктах нижче.

2.3.1 Створення таблиць та обмежень

Структуру бази задано мовою визначення даних SQL (DDL). Для типів операцій та ролей користувачів оголошено перелічувані типи (ENUM), що обмежують допустимі значення на рівні бази. Фрагмент скрипту створення ключових таблиць — номенклатури, залишків та документів — наведено нижче.

```sql
CREATE TYPE operation_type AS ENUM
    ('receipt', 'writeoff', 'transfer');

CREATE TABLE nomenclature (
    id          SERIAL PRIMARY KEY,
    code        VARCHAR(64)   NOT NULL UNIQUE,
    name        VARCHAR(255)  NOT NULL,
    unit        VARCHAR(32)   NOT NULL,
    group_name  VARCHAR(128),
    min_balance NUMERIC(12,3) NOT NULL DEFAULT 0
);

CREATE TABLE balances (
    id              SERIAL PRIMARY KEY,
    nomenclature_id INTEGER NOT NULL
        REFERENCES nomenclature(id) ON DELETE CASCADE,
    warehouse_id    INTEGER NOT NULL
        REFERENCES warehouses(id)   ON DELETE CASCADE,
    quantity        NUMERIC(12,3) NOT NULL DEFAULT 0,
    CONSTRAINT uq_balance
        UNIQUE (nomenclature_id, warehouse_id),
    CONSTRAINT chk_qty_nonneg CHECK (quantity >= 0)
);

```

Рис. 8 Фрагмент DDL-скрипту створення таблиць

Обмеження UNIQUE (nomenclature_id, warehouse_id) гарантує, що для кожної пари «позиція — склад» існує не більше одного запису залишку. Обмеження CHECK (quantity >= 0) забороняє від'ємний залишок на рівні бази даних. Зовнішні ключі з дією ON DELETE CASCADE забезпечують узгоджене видалення залежних записів.

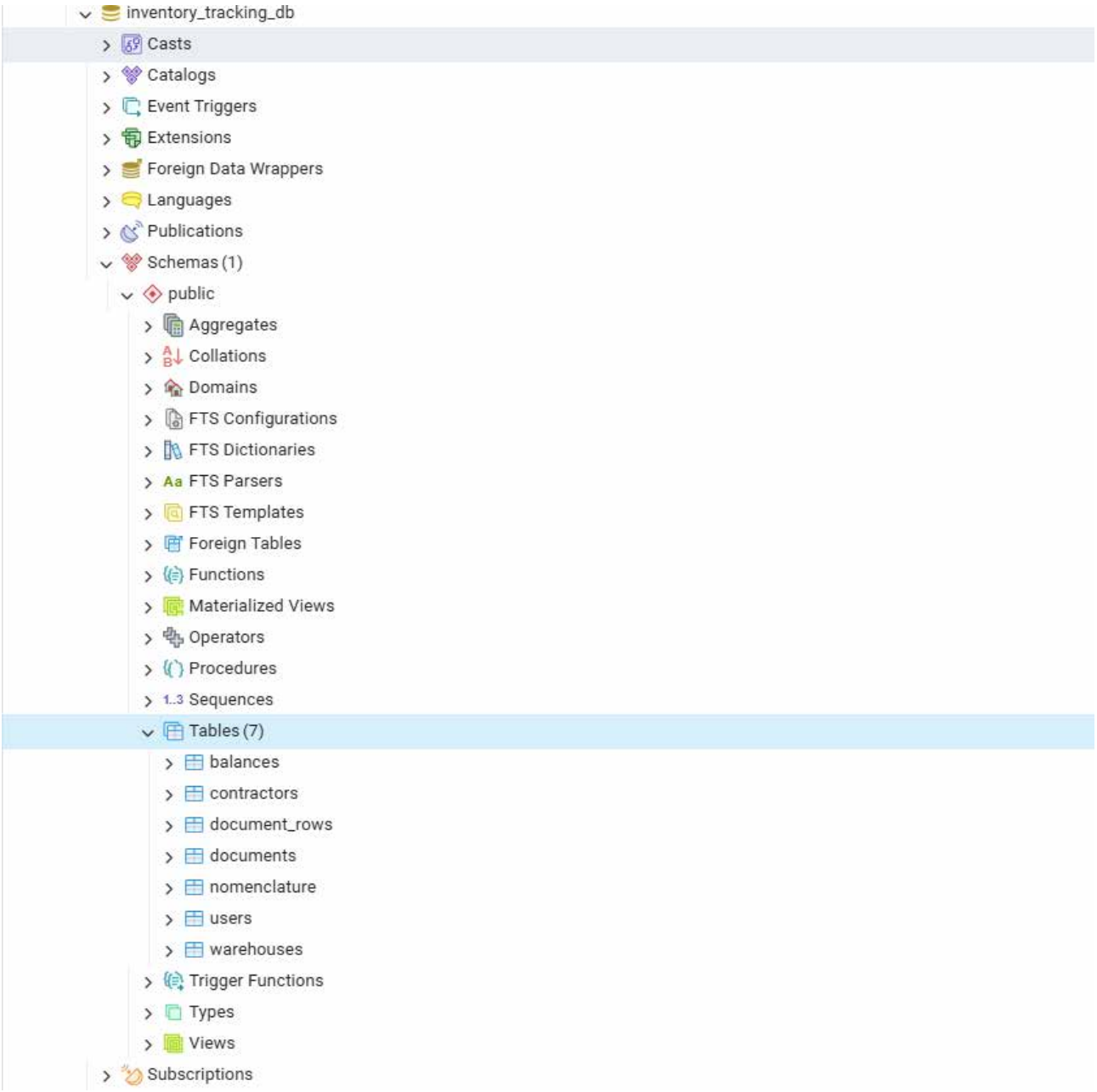


Рис. 9 Створені таблиці бази даних у середовищі адміністрування

2.3.2 Створення індексів

Для прискорення типових запитів користувача над таблицями створено індекси. Окрім автоматично створених індексів за первинними ключами, додано індекси за полями, що найчастіше беруть участь у фільтрації та сортуванні [14].

Query Query History

```
1  CREATE INDEX idx_documents_type
2      ON documents (type, doc_date);
3  CREATE INDEX idx_rows_document
4      ON document_rows (document_id);
5  CREATE INDEX idx_balances_nom
6      ON balances (nomenclature_id);
7
```

Рис. 10 Декларація індексів

Індекс за полями type та doc_date оптимізує побудову журналу документів з фільтром за типом операції та сортуванням за датою; індекс за document_id прискорює вибірку рядків конкретного документа; індекс за nomenclature_id — отримання залишків окремої позиції.

2.3.3 Створення тригерної функції

Автоматичний перерахунок залишків при проведенні документа реалізовано тригерною функцією на мові PL/pgSQL [15]. Функція спрацьовує в момент, коли документ позначається проведеним, і по кожному рядку документа змінює відповідний запис у таблиці balances залежно від типу операції: надходження збільшує залишок на складі документа, списання зменшує його, переміщення зменшує залишок на складі-джерелі та збільшує на складі-отримувачі. Фрагмент функції наведено нижче на рис. 11.

Query Query History

```
CREATE OR REPLACE FUNCTION fn_post_document()
RETURNS TRIGGER AS $$
DECLARE r RECORD;
BEGIN
  IF NEW.is_posted AND NOT OLD.is_posted THEN
    FOR r IN SELECT nomenclature_id, quantity
             FROM document_rows
             WHERE document_id = NEW.id LOOP
      IF NEW.type = 'receipt' THEN
        INSERT INTO balances
          (nomenclature_id, warehouse_id, quantity)
        VALUES (r.nomenclature_id, NEW.warehouse_id, r.quantity)
        ON CONFLICT (nomenclature_id, warehouse_id)
        DO UPDATE SET quantity =
          balances.quantity + r.quantity;
      ELSIF NEW.type = 'writeoff' THEN
        UPDATE balances SET quantity = quantity - r.quantity
        WHERE nomenclature_id = r.nomenclature_id
          AND warehouse_id = NEW.warehouse_id;
      END IF;  -- (гілка transfer — у повному лістингу)
    END LOOP;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Рис. 11 Тригерна функція перерахунку залишків

Якщо під час списання чи переміщення поточного залишку недостатньо, операція оновлення порушує обмеження CHECK (quantity >= 0), база даних повертає помилку, і вся транзакція проведення документа відкочується. Таким чином контроль достатності залишку реалізовано на рівні самої бази, без покладання на коректність логіки застосунку.

2.3.4 Проведення документа на стороні застосунку

Серверна частина системи реалізована мовою Go. Проведення документа виконується в межах однієї транзакції бази даних: застосунок установлює ознаку проведення документа, після чого тригер автоматично перераховує залишки. Якщо база повертає помилку (зокрема через нестачу залишку), транзакція відкочується повністю. Відповідний фрагмент коду наведено нижче на рис. 12.

```go
func (r *Repo) PostDocument(ctx context.Context,
    docID int) error {
  tx, err := r.db.BeginTx(ctx, &sql.TxOptions{})
  if err != nil {
    return fmt.Errorf("початок транзакції: %w", err)
  }
  defer tx.Rollback()

  _, err = tx.ExecContext(ctx,
    `UPDATE documents SET is_posted = TRUE
     WHERE id = $1 AND is_posted = FALSE`, docID)
  if err != nil {
    return fmt.Errorf("проведення: %w", err)
  }
  return tx.Commit()
```

Рис. 12 Проведення документа в межах транзакції (Go)

2.4 Схема та фізична структура бази даних

Фізична структура бази даних системи складається із семи таблиць, пов'язаних зовнішніми ключами відповідно до логічної моделі.

- **users** — облікові записи користувачів; ідентифікатор ролі визначає доступні дії, поле warehouse_id закріплює користувача за складом;
- **warehouses** — склади підприємства із зазначенням матеріально відповідальної особи;
- **contractors** — постачальники та отримувачі цінностей;
- **nomenclature** — довідник ТМЦ з одиницею виміру та мінімальним рівнем залишку;
- **balances** — поточні залишки в розрізі пари «позиція — склад»; кількість зберігається з фіксованою точністю NUMERIC(12,3);
- **documents** — заголовки документів руху з типом операції, відповідальним користувачем, контрагентом та складами;
- **document_rows** — рядки специфікації документів з посиланням на позицію номенклатури та кількістю.

Особливості фізичної структури полягають у трьох ключових рішеннях.

Контроль цілісності на рівні бази. Усі зв'язки між таблицями оголошено зовнішніми ключами, тому база самостійно не допускає посилання на неіснуючі записи. Видалення позиції номенклатури чи складу каскадно видаляє пов'язані записи залишків, а від'ємний залишок неможливий через обмеження CHECK.

Явне зберігання залишку з тригерним перерахунком. Поточний залишок зберігається в окремій таблиці balances і автоматично оновлюється тригером при проведенні документа. Це поєднує швидкість читання залишків (одна вибірка замість агрегатного запиту по всій історії) з гарантією узгодженості, бо перерахунок виконує сама база.

Транзакційне проведення документів. Зміна залишків відбувається виключно в межах транзакції проведення документа. Завдяки властивостям ACID часткове оновлення неможливе: у разі будь-якої помилки база повертається до стану до початку операції.

На рис. 13 наведено приклад вмісту таблиці balances у середовищі адміністрування, що ілюструє описану структуру зберігання поточних залишків.

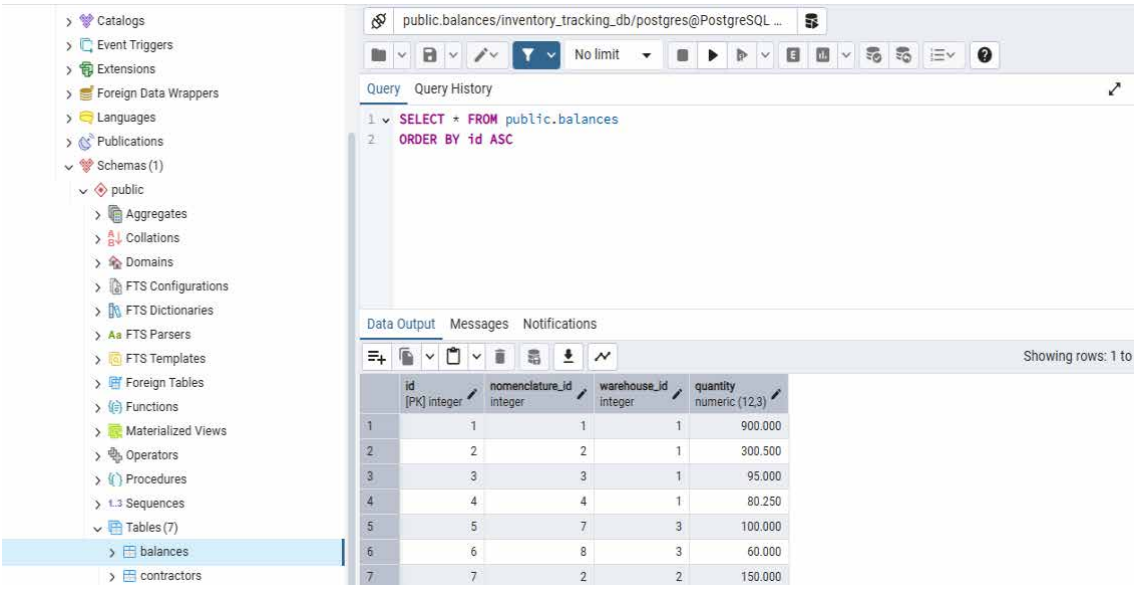


Рис. 13 Приклад вмісту таблиці залишків

Підсумовуючи, фізична архітектура бази даних побудована за принципами реляційної моделі та повністю задовольняє вимоги третьої нормальної форми (3НФ). Підтримка структурної цілісності та консистентності облікової інформації делегована безпосередньо системі управління базами даних: для

цього застосовуються вбудовані декларативні обмеження, серверні тригери та транзакційні механізми обробки запитів. Таке інженерне рішення оптимально адаптоване до специфіки складського обліку, зокрема оперування дробовими показниками кількості матеріалів, та слугує надійним гарантом безпечного збереження інформаційних активів підприємства.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Абстракції предметної області

Процес перетворення неформалізованого текстового опису предметної області на чітку структуровану модель базується на ідентифікації базових абстракцій. Ці абстракції виступають узагальненими концептами, що консолідують споріднені об'єкти та бізнес-процеси досліджуваного середовища. Кожному такому виділеному елементу призначається специфічний набір атрибутів (властивостей) та поведінкових функцій (обов'язків). Атрибути фіксують параметри, що безпосередньо визначають стан і логіку роботи системи, тоді як обов'язки чітко регламентують перелік операцій, за виконання яких несе відповідальність конкретна сутність [16].

Для предметної області обліку інвентарю та матеріалів виокремлено сім абстракцій, що відповідають інформаційним об'єктам, визначеним у попередніх розділах.

– **Номенклатура** — картка матеріалу з кодом, найменуванням, одиницею виміру та мінімальним залишком; відповідає за зберігання нормативно-довідкових даних про позицію та перевірку, чи опустився залишок нижче мінімуму;

– **Склад** — фізичне місце зберігання цінностей із закріпленою матеріально відповідальною особою; відповідає за надання поточних залишків у своєму розрізі;

– **Залишок** — поточна кількість конкретної позиції на конкретному складі; відповідає за коректне застосування зміни кількості при русі;

– **Документ руху** — господарська операція (надходження, списання, переміщення); відповідає за валідацію складу позицій та власне проведення зі зміною залишків;

– **Рядок документа** — окрема позиція у складі документа із зазначеною кількістю;

– **Контрагент** — зовнішній постачальник або отримувач цінностей;

– **Користувач** — обліковий запис працівника з певною роллю; відповідає за перевірку прав на виконання операції.

Відмінною рисою предметної області є чіткий розподіл відповідальності між обліком довідкових даних (номенклатура, склади, контрагенти) та реєстрацією операцій руху (документи та їхні рядки). Поточний стан запасів (залишки) є похідним від проведених операцій і змінюється виключно під час проведення документів. Цей поділ безпосередньо враховано при подальшому проєктуванні архітектури, де доступ до даних, бізнес-логіка та представлення розділені на окремі шари.

Виокремлені абстракції є основою для побудови діаграми класів та діаграм кооперацій, що описують статичні зв'язки між сутностями та сценарії їх взаємодії. Властивості абстракцій стають атрибутами класів, обов'язки — методами, а характер взаємозв'язків визначає тип асоціації, агрегації чи композиції між класами.

3.2 Моделювання структури та взаємодії об'єктів

Після виокремлення абстракцій наступним кроком є моделювання структури та взаємодії об'єктів системи. На цьому етапі визначають не лише статичний перелік сутностей з атрибутами й методами, а й сценарії, у яких ці сутності взаємодіють для реалізації конкретної функціональності [16]. Для моделювання застосовано мову UML; побудовано діаграму класів (статична структура) та діаграми кооперацій (динаміка окремих сценаріїв).

3.2.1 Діаграма класів

Діаграма класів зображує статичну об'єктну структуру системи: основні класи, їхні атрибути, методи та зв'язки між ними із зазначенням кратності. Розроблену діаграму наведено на рис. 14.

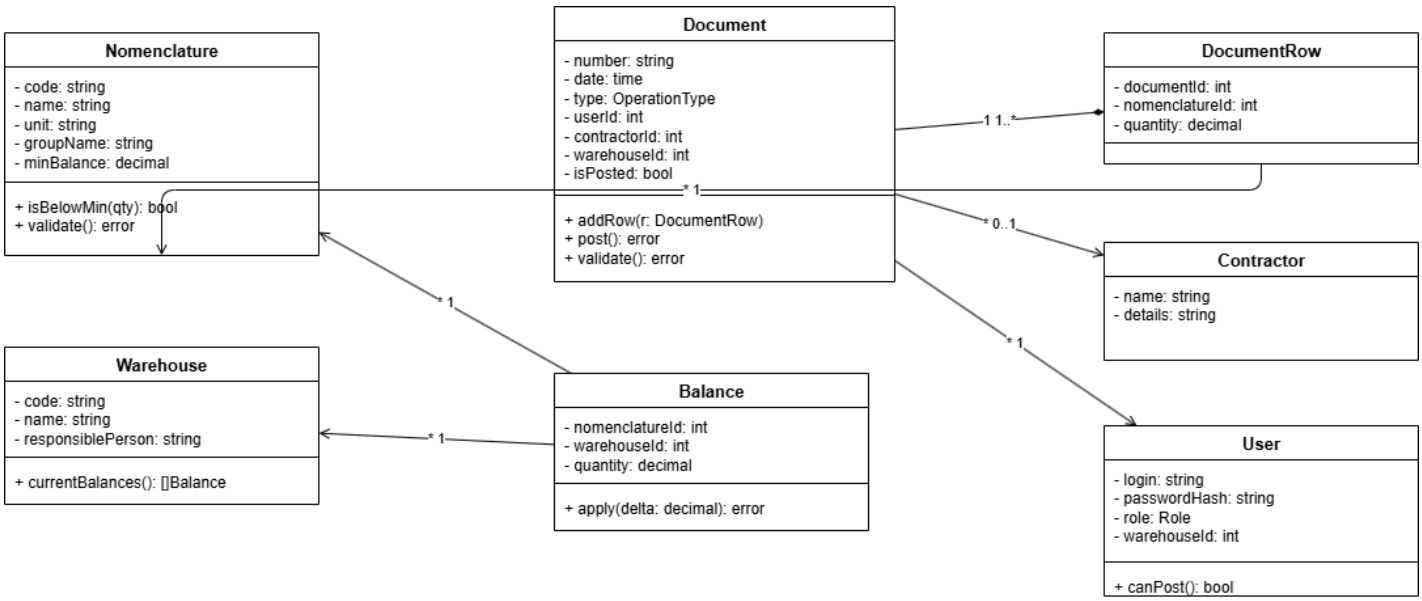


Рис. 14 Діаграма класів із асоціаціями

Діаграма містить сім класів, що відповідають виокремленим абстракціям. Для кожного визначено атрибути та методи. Зокрема, клас Document має метод post(), що ініціює проведення документа, та validate() для перевірки складу позицій; клас Nomenclature — метод isBelowMin() для контролю мінімального залишку; клас User — метод canPost(), що визначає право проводити документи за роллю.

Зв'язки між класами відображають природу відношень предметної області. Документ і його рядки пов'язані композицією (1 до 1..*): рядок не існує без документа. Рядок документа та залишок посилаються на номенклатуру асоціацією «багато до одного». Залишок пов'язує номенклатуру зі складом. Документ належить одному користувачеві та, для операцій надходження чи реалізації, одному контрагентові (кратність 0..1, оскільки для внутрішнього переміщення контрагент відсутній).

3.2.2 Діаграми кооперацій

Діаграми кооперацій моделюють динамічні аспекти системи. На відміну від діаграми класів, кооперація розглядає лише ті класи та зв'язки, що беруть участь у конкретному сценарії, що дозволяє детально дослідити окремі процеси [16]. Для системи виокремлено три кооперації на основі тих самих класів.

Кооперація «Реєстрація надходження». Користувач створює документ надходження та додає до нього позиції. Задіяні класи Document, DocumentRow та Nomenclature. Зв'язок Document — DocumentRow є композицією, DocumentRow — Nomenclature — асоціацією. Кооперацію наведено на рис. 15.

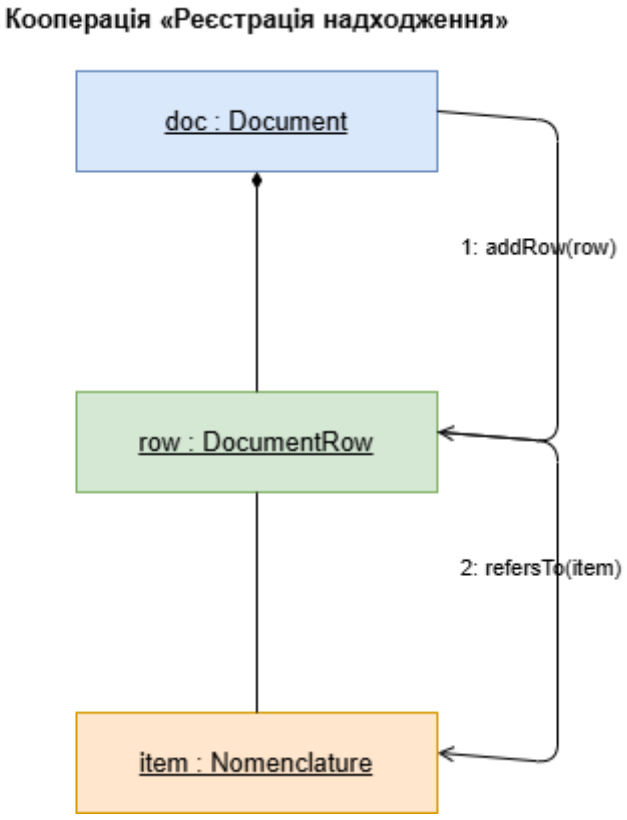


Рис. 15 Діаграма кооперації «Реєстрація надходження»

Кооперація «Проведення документа». Під час проведення документ змінює залишки відповідних позицій. Задіяні класи Document, DocumentRow, Balance. Зв'язок Document — Balance реалізується опосередковано через рядки документа: кожен рядок змінює відповідний залишок методом apply(). Кооперацію наведено на рис. 16.

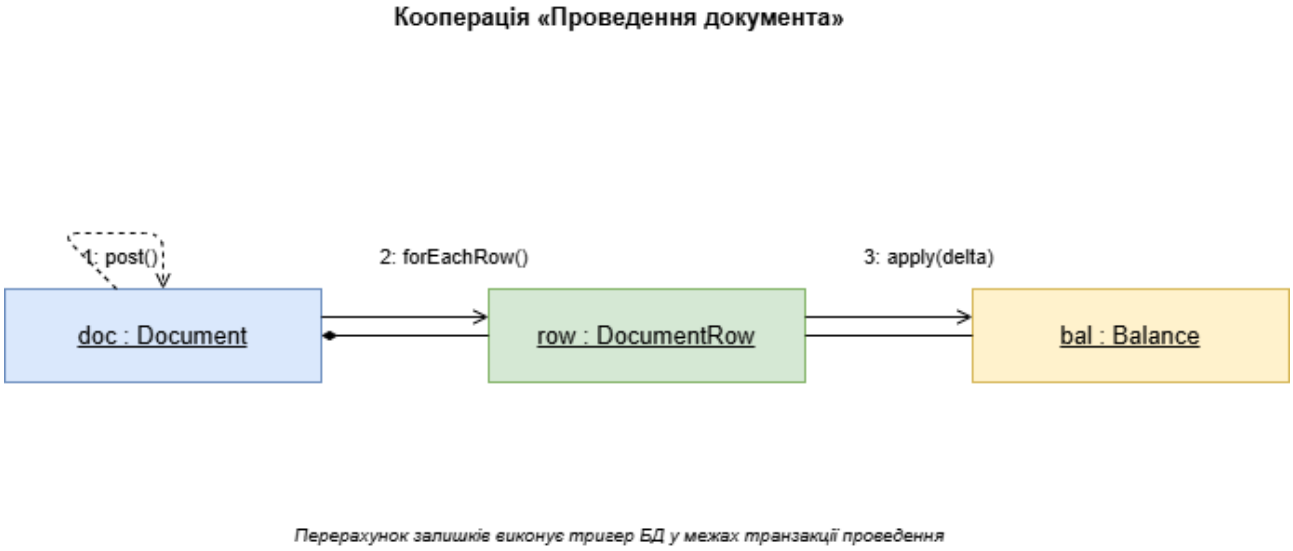


Рис. 16 Діаграма кооперації «Проведення документа»

Кооперація «Контроль мінімального залишку». Після зміни залишку система перевіряє, чи не опустився він нижче мінімуму. Задіяні класи Balance та Nomenclature; зв'язок — асоціація, перевірку виконує метод isBelowMin() класу Nomenclature. Кооперацію наведено на рис. 17.

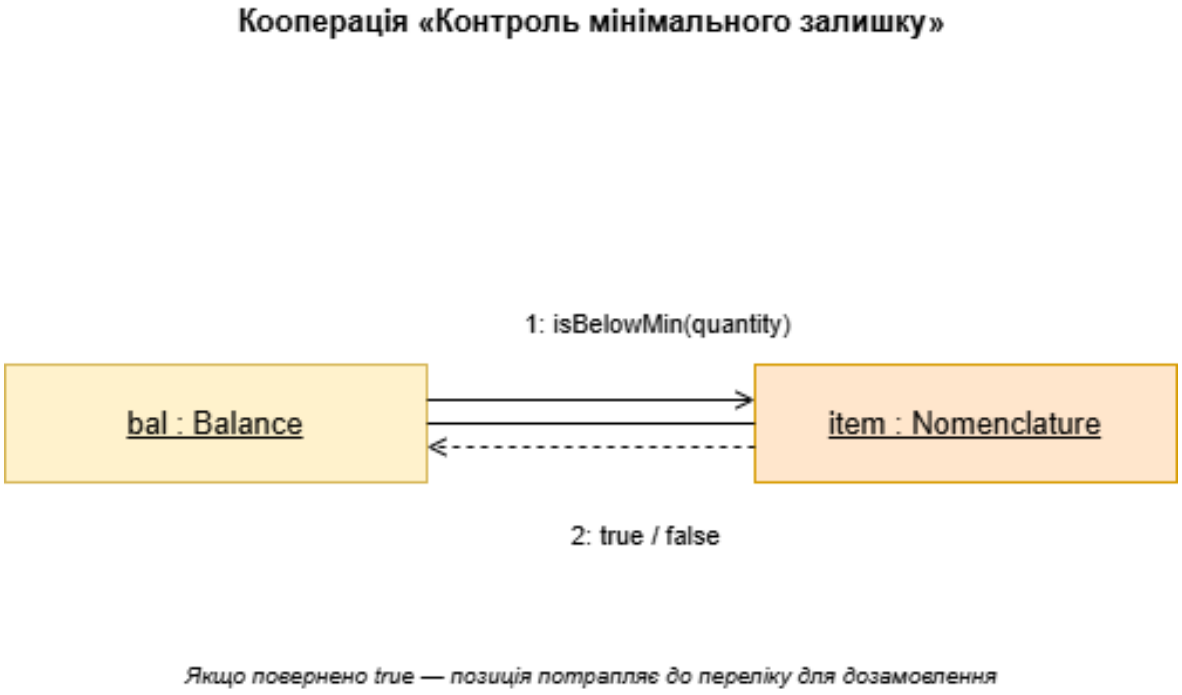


Рис. 17 Діаграма кооперації «Контроль мінімального залишку»

Описані кооперації деталізують механіку основних сценаріїв і слугують основою для проєктування організаційної структури програмного забезпечення.

3.3 Організаційна структура програмного забезпечення

Для високорівневого представлення архітектури використано діаграму пакетів, що показує, як функціональність групується у логічні модулі з власною зоною відповідальності та які залежності існують між ними [16]. Діаграму пакетів наведено в Додатку А.

Система побудована за класичною багатошаровою архітектурою, у якій кожен шар звертається лише до сусіднього нижчого. Це забезпечує односпрямованість залежностей та спрощує тестування й супровід.

Опис пакетів

– **handlers** — шар обробки HTTP-запитів. Приймає запити від браузера, перевіряє права доступу, викликає відповідні методи сервісного шару та повертає згенеровані HTML-сторінки або HTMX-фрагменти. Бізнес-логіки не містить;

– **services** — шар бізнес-логіки. Реалізує операції предметної області: проведення документів, контроль залишків, інвентаризацію, формування звітів. Координує роботу, але не звертається до бази даних напряму — лише через шар repository;

– **repository** — шар доступу до даних. Інкапсулює SQL-запити та керування транзакціями через бібліотеку pgx. Лише цей шар знає про структуру бази даних;

– **models** — структури даних предметної області (Nomenclature, Document, Balance тощо), спільні для всіх шарів;

– **templates** — HTML-шаблони Go Templates, які шар handlers наповнює даними для відповіді браузеру.

Залежності між пакетами

Напрям залежностей суворо односпрямований: handlers залежить від services, services — від repository, repository — від models та бази даних. Шар представлення (templates) використовується лише пакетом handlers. Така структура виключає циклічні залежності: зміна деталей доступу до бази (наприклад, оптимізація запиту) зачіпає лише repository й не впливає на бізнес-

логіку та представлення. Це відповідає принципу розділення відповідальності та полегшує модульне тестування кожного шару окремо.

3.4 Компонентна структура системи

Компонентну структуру представлено діаграмою компонентів, що моделює фізичну архітектуру системи — виконувані середовища, артефакти та зовнішні сервіси під час розгортання [16]. На відміну від діаграми пакетів, що фіксує логічний поділ, діаграма компонентів зображує систему такою, якою вона існує під час експлуатації. Діаграму наведено в Додатку В.

Система має трирівневу архітектуру розгортання, у якій три фізично відокремлені вузли взаємодіють через мережу [17].

Робоче місце користувача. Вебоглядач, у якому відображається інтерфейс системи. Інтерактивність забезпечується бібліотекою HTMX, що надсилає запити на сервер та оновлює окремі фрагменти сторінки без повного перезавантаження, а зовнішній вигляд — фреймворком Bootstrap. Жодного клієнтського програмного забезпечення встановлювати не потрібно.

Сервер застосунку. Go-застосунок, скомпільований у єдиний виконуваний файл. Містить компоненти трьох шарів (handlers, services, repository) та артефакт шаблонів. Завдяки компіляції в один бінарник розгортання зводиться до копіювання одного файлу на сервер, без потреби встановлювати середовище виконання чи залежності.

Сервер баз даних. СУБД PostgreSQL із таблицями, тригерами та індексами, спроєктованими у другому розділі. Go-застосунок звертається до неї через драйвер pgx за протоколом PostgreSQL.

Описана компонентна структура реалізує класичну вебархітектуру, де браузер відповідає за представлення, Go-застосунок — за обробку запитів і бізнес-логіку, а PostgreSQL — за зберігання та цілісність даних. Компактність розгортання (один бінарник плюс база) є прямим наслідком вибору технологічного стеку й спрощує впровадження на сервері підприємства або приватному VPS.

3.5 Вибір інструментарію для створення прикладного програмного забезпечення

Вибір інструментальних засобів обґрунтовано вимогами до системи: компактність розгортання, висока швидкодія, надійність та простота супроводу.

Мова програмування. Серверну частину реалізовано мовою Go. Вибір зумовлено компіляцією у єдиний виконуваний файл без зовнішніх залежностей, статичною типізацією, що зменшує кількість помилок, та високою швидкодією. Розробка ведеться у середовищі Goland, керування залежностями — вбудованим інструментом Go Modules.

Вебінтерфейс. Інтерфейс побудовано на стандартному пакеті html/template (Go Templates) із серверним відтворенням сторінок. Для динамічного оновлення окремих блоків без повного перезавантаження сторінки застосовано бібліотеку HTMX, що дозволяє надсилати запити безпосередньо з HTML-атрибутів. Зовнішній вигляд та адаптивність забезпечує фреймворк Bootstrap. Такий підхід зосереджує логіку на стороні сервера й уникає складності, властивої повноцінним клієнтським фреймворкам.

База даних. Для зберігання даних використано PostgreSQL — реляційну СУБД з підтримкою транзакцій, обмежень цілісності та тригерів. Доступ до бази з Go-застосунку реалізовано через драйвер pgx, що забезпечує ефективну роботу з типами PostgreSQL, зокрема з типом NUMERIC для дробових кількостей.

Розгортання. Готовий застосунок компілюється в один виконуваний файл і розгортається на сервері підприємства або приватному VPS. Доступ користувачів здійснюється через вебоглядач захищеним каналом HTTPS.

3.6 Алгоритмізація та програмування програмних модулів

Перед переходом до коду доцільно зафіксувати алгоритмічну сторону ключового бізнес-процесу: послідовність кроків, умови переходу та реакцію на типові помилки. Для демонстрації обрано алгоритм проведення документа руху — центральну операцію системи, що поєднує перевірку прав доступу,

транзакційну зміну даних, спрацювання тригера перерахунку залишків та контроль достатності залишку. Блок-схему алгоритму наведено на рис. 18.

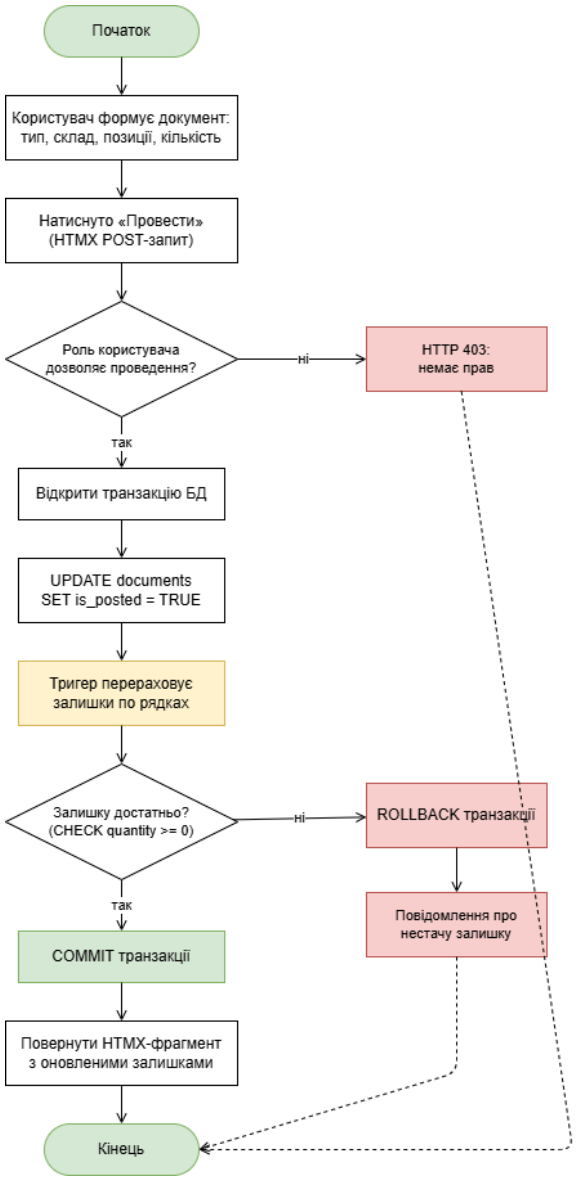


Рис. 18 Блок-схема алгоритму проведення документа руху

Процес починається з того, що користувач формує документ (тип операції, склад, перелік позицій з кількістю) та натискає кнопку «Провести». Запит надсилається на сервер засобом HTMX. Спершу обробник перевіряє, чи дозволяє роль користувача проведення документів; за відсутності прав повертається відповідь із кодом 403. За наявності прав відкривається транзакція бази даних, у якій документ позначається проведеним. Це активує тригер, що перераховує залишки по кожному рядку документа. Якщо для списання чи переміщення

залишку недостатньо, спрацьовує обмеження бази даних, транзакція відкочується, а користувач отримує повідомлення про нестачу. За успішного перерахунку транзакція фіксується, і сервер повертає HTMX-фрагмент з оновленою таблицею залишків.

Обробку запиту реалізовано на рівні шару handlers. Обробник перевіряє права доступу, делегує проведення сервісному шару та формує відповідь. Фрагмент коду наведено на рис. 19.

```
func (h *Handler) PostDocument(w http.ResponseWriter,
    r *http.Request) {
  user := models.UserFromContext(r.Context())
  if !user.CanPost() {
    w.WriteHeader(http.StatusForbidden)
    h.render(w, "partials/error.html",
      "Недостатньо прав")
    return
  }
  docID, _ := strconv.Atoi(r.PathValue("id"))
  if err := h.docService.Post(r.Context(),
      docID); err != nil {
    w.WriteHeader(http.StatusUnprocessableEntity)
    h.render(w, "partials/error.html", err.Error())
    return
  }
  balances, _ := h.docService.BalancesForDocument(
    r.Context(), docID)
  h.render(w, "partials/balances_table.html", balances)
}
```

Рис. 19 Обробник проведення документа (handlers)

Власне бізнес-логіку винесено в сервісний шар. Метод Post відкриває транзакцію, перевіряє стан документа та виставляє ознаку проведення; порушення обмеження цілісності розпізнається як нестача залишку й перетворюється на зрозуміле користувачеві повідомлення. Фрагмент наведено на рис. 20.

```go
func (s *DocumentService) Post(ctx context.Context,
    docID int) error {
  err := s.repo.WithTx(ctx, func(tx repository.Tx) error {
    doc, err := tx.GetDocument(ctx, docID)
    if err != nil { return err }
    if doc.IsPosted {
      return errors.New("документ уже проведено")
    }
    return tx.MarkPosted(ctx, docID)
  })
  if repository.IsCheckViolation(err) {
    return errors.New(
      "недостатньо залишку для проведення")
  }
  return err
}
```

Рис. 20 Сервіс проведення документа в транзакції (services)

Інтерфейсну частину реалізовано шаблоном Go Templates. Кнопка проведення містить HTMX-атрибути, що надсилають POST-запит та замінюють блок залишків відповіддю сервера. Фрагмент шаблону наведено на рис. 21.

```
{{if not .IsPosted}}
<button class="btn btn-primary"
        hx-post="/documents/{{.ID}}/post"
        hx-target="#balances"
        hx-swap="innerHTML"
        hx-confirm="Провести документ?">
  Провести
</button>
{{else}}
<span class="badge bg-success">Проведено</span>
{{end}}
```

Рис. 21 Фрагмент шаблону з HTMX-кнопкою проведення

Описана реалізація демонструє кілька архітектурних рішень. Перевірка прав доступу відокремлена від бізнес-логіки й виконується на рівні обробника. Бізнес-логіка ізольована у сервісному шарі та не залежить від деталей HTTP чи бази даних. Контроль достатності залишку забезпечується самою базою через обмеження та тригер, тож коректність даних гарантована незалежно від можливих помилок у коді застосунку. Застосування HTMX дозволяє оновлювати лише потрібний фрагмент сторінки, що зменшує обсяг переданих даних та підвищує чутливість інтерфейсу.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Якість програмного забезпечення підтверджується тестуванням — перевіркою того, що реалізована поведінка системи відповідає поставленим вимогам. Для розробленої системи обліку матеріалів тестування мало на меті переконатися, що облікові дані залишаються узгодженими після будь-яких операцій руху, що контроль достатності залишку та мінімального рівня запасу спрацьовує коректно, а інтерфейс коректно відображає стан запасів у вебоглядачі.

Перевірку організовано на двох рівнях деталізації. На нижчому рівні тестувалися окремі функції бізнес-логіки незалежно одна від одної — це модульне тестування (Unit Testing). Перевірялися, зокрема, функція перерахунку залишку для кожного типу операції, функція контролю достатності залишку при списанні та функція виявлення позицій, що опустилися нижче мінімуму. Для кожної з них формувався набір вхідних значень із заздалегідь відомим результатом, після чого фактичний результат звірявся з очікуваним [19].

На вищому рівні застосовано системне тестування (System Testing), за якого перевірялася вже зібрана система в умовах, наближених до робочих: Go-застосунок працював разом із базою даних PostgreSQL, а перевірка охоплювала повні ланцюжки дій — створення документа, його проведення, автоматичний перерахунок залишків та формування звітності. Окрему увагу приділено поведінці системи за некоректних дій користувача, зокрема спробі списати більше, ніж наявно на складі [20].

Для оцінки системи з позиції користувача застосовано підхід «чорної скриньки» (Black Box Testing), за якого тестувальник не зважає на внутрішню реалізацію, а працює з інтерфейсом за наперед визначеним сценарієм і фіксує, чи збігається отриманий результат із очікуваним [21]. Саме такий сценарій, що

відтворює типовий робочий день комірника, розглянуто нижче як ілюстрацію цілісної роботи системи.

Сценарій перевірки методом «чорної скриньки»

Сценарій складається з послідовних кроків, кожен із яких відповідає окремому екрану системи; для кожного кроку зафіксовано очікуваний і фактичний стан інтерфейсу.

Крок 1. Авторизація. Користувач відкриває систему у вебоглядачі, вводить логін і пароль та підтверджує вхід. Після перевірки облікових даних відкривається головна сторінка з обсягом прав, що відповідає ролі. Вигляд сторінки входу подано на рис. 22.

Рис. 22 Сторінка входу до системи

Крок 2. Огляд поточного стану запасів. На головній сторінці відображаються зведені показники (кількість позицій, складів і документів, число позицій нижче мінімуму) та таблиця залишків із можливістю фільтрації за складом. Позиції, що потребують поповнення, виділяються окремо. Головний екран подано на рис. 23.

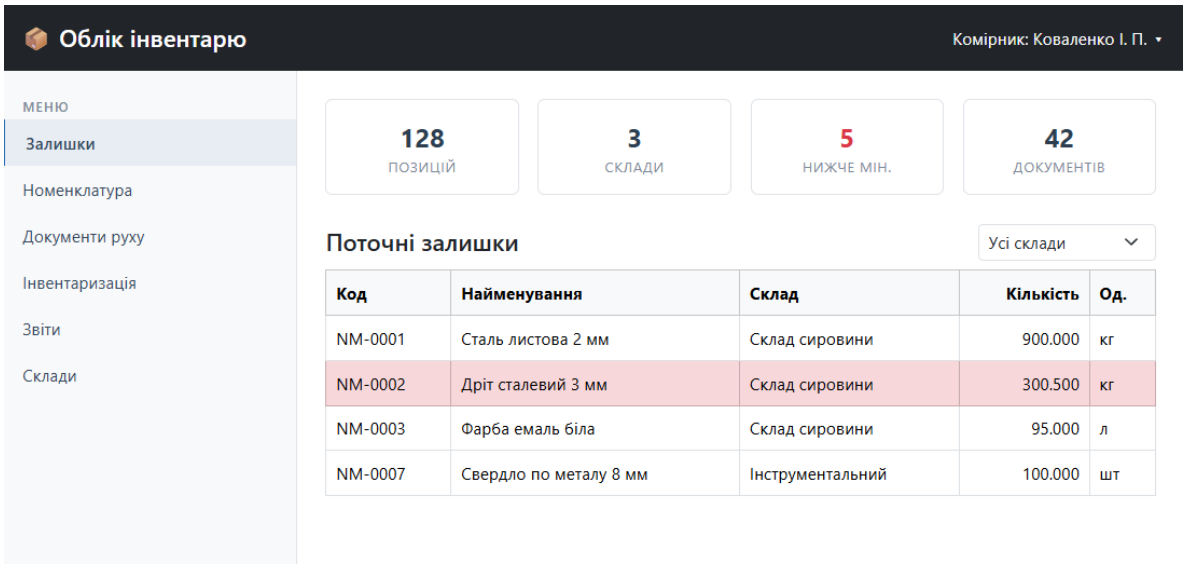


Рис. 23 Головний екран із поточними залишками

Крок 3. Робота з довідником номенклатури. Користувач переходить до довідника ТМЦ, користується пошуком за кодом чи назвою та переглядає параметри позицій, зокрема встановлений мінімальний рівень залишку. Сторінку подано на рис. 24.

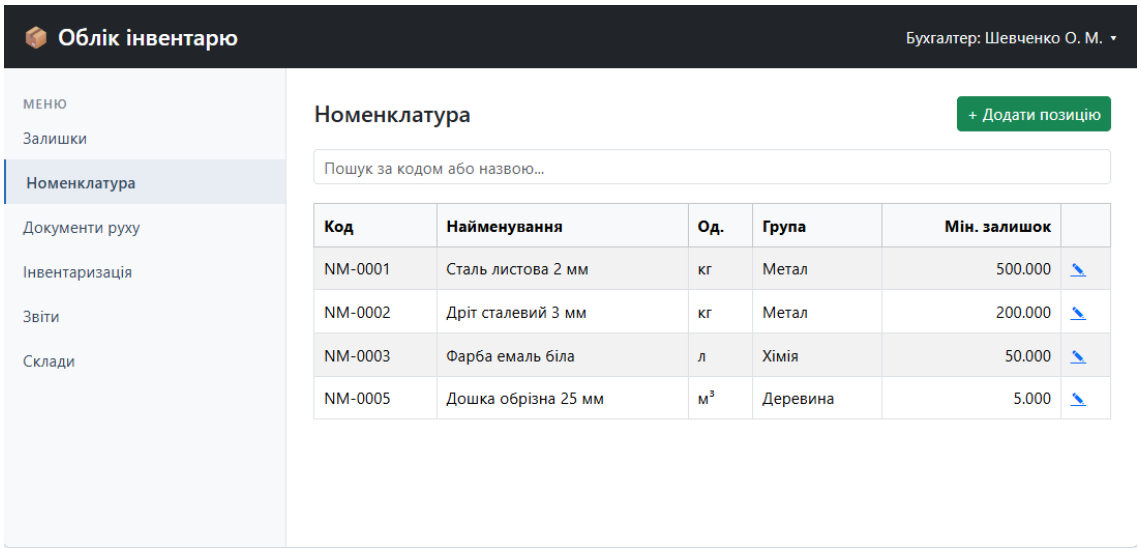


Рис. 24 Довідник номенклатури

Крок 4. Оформлення надходження. Користувач створює документ надходження, зазначає склад і постачальника, додає позиції з кількістю та проводить документ. Система в межах однієї транзакції збільшує залишки відповідних позицій, що відразу видно на сторінці залишків. Форму документа подано на рис. 25.

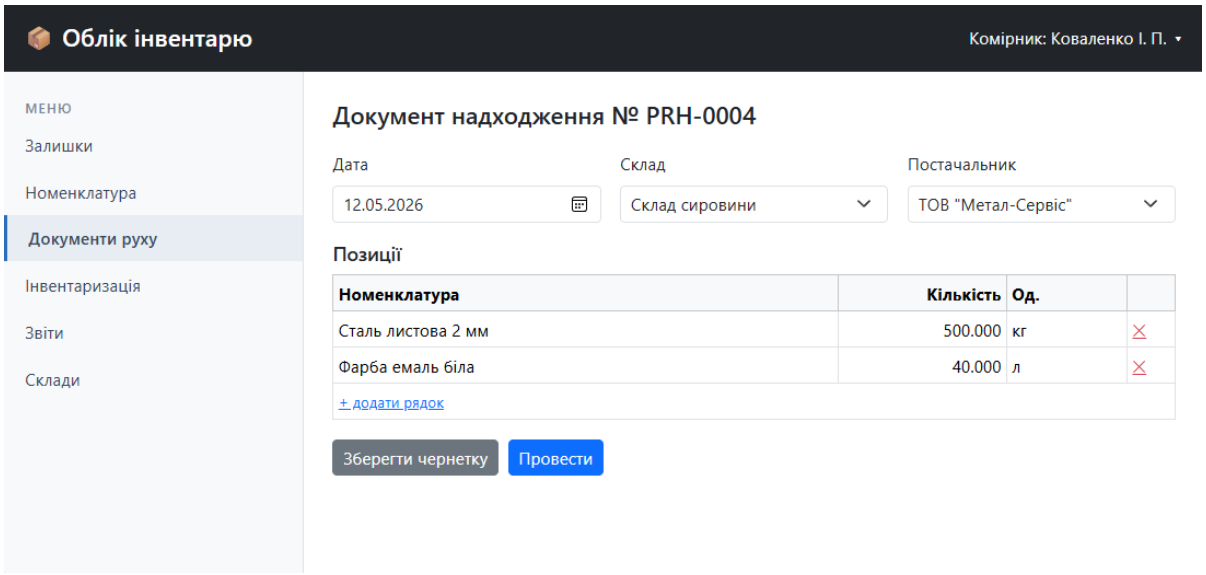


Рис. 25 Форма документа надходження

Крок 5. Звіряння під час інвентаризації. Користувач відкриває інвентаризацію за обраним складом, вносить фактично перелічену кількість, а система обчислює відхилення від облікових даних і виділяє нестачі та надлишки. Відомість подано на рис. 26.

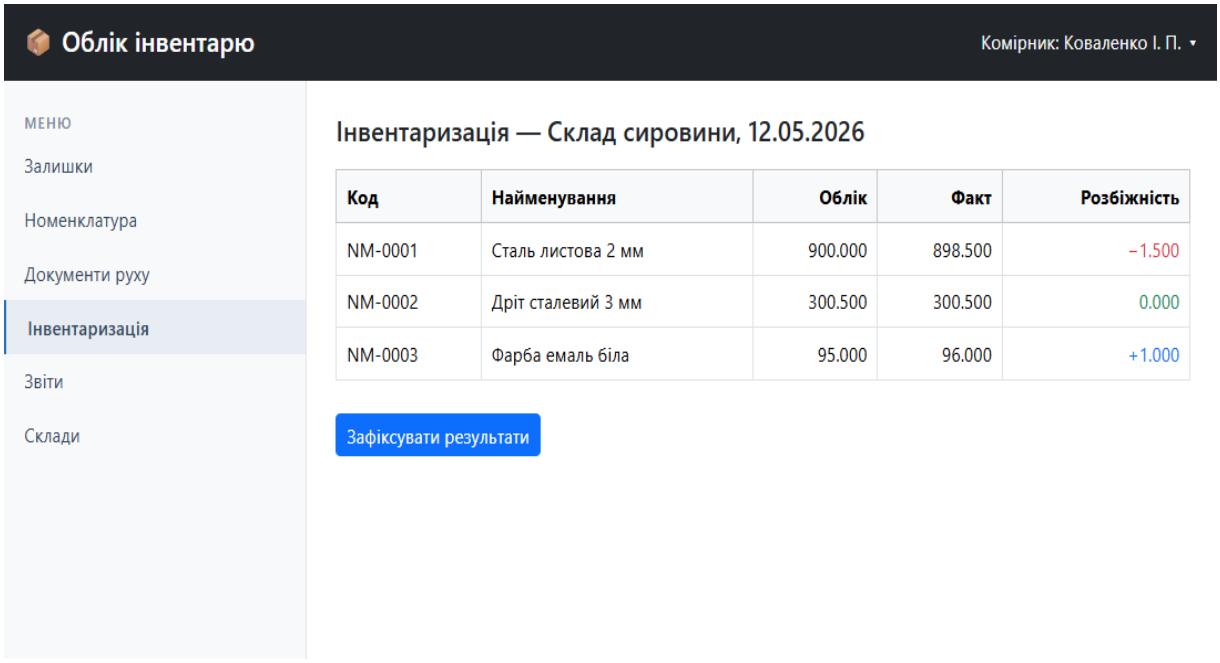


Рис. 26 Відомість результатів інвентаризації

Підсумок перевірки

За результатами проходження сценарію всі кроки відпрацювали відповідно до очікувань. Вхід та розмежування прав за ролями працювали коректно,

проведення документів супроводжувалося точним перерахунком залишків з урахуванням дробових кількостей, спроба списати понад наявний залишок блокувалася системою, а інвентаризація формувала правильні відхилення. Дрібні зауваження до зручності інтерфейсу та повідомлень про помилки, виявлені під час перевірки, було усунено в межах доопрацювання.

4.2 Вимоги до апаратного та програмного забезпечення

Для коректної роботи розробленої системи необхідно дотримуватися певних вимог до апаратного та програмного забезпечення, що забезпечить стабільність, швидкодію та безпеку даних. Система має просту дворівневу архітектуру розгортання: сервер застосунку та сервер баз даних (які можуть бути розміщені на одному фізичному вузлі), а доступ користувачів здійснюється через вебоглядач. Діаграму розгортання наведено в Додатку Д.

Серверну частину реалізовано як єдиний виконуваний застосунок мовою Go, що не потребує встановлення середовища виконання чи додаткових залежностей. База даних PostgreSQL може бути розгорнута на тому самому сервері або на окремому вузлі. Клієнтська частина не потребує встановлення — користувач працює через звичайний вебоглядач. Нижче наведено вимоги, згруповані за середовищами.

Вимоги до серверної частини

Сервер обслуговує запити користувачів та зберігає базу даних. Для невеликого підприємства достатньо помірних обчислювальних ресурсів.

Апаратне забезпечення:

– процесор від 2 vCPU або аналогічний;
– оперативна пам'ять — 4 ГБ і більше;
– вільне місце на накопичувачі — не менше 20 ГБ SSD;
– мережеве підключення з пропускною здатністю від 100 Мбіт/с.

Програмне забезпечення:

– операційна система Ubuntu 22.04 / 24.04 LTS або сумісний дистрибутив Linux (можливе розгортання і на Windows Server);
– СУБД PostgreSQL версії 14 або новіша;
– вебсервер або зворотний проксі (наприклад, Nginx) для термінації HTTPS — за потреби.

Окремого середовища виконання для застосунку встановлювати не потрібно: Go-бінарник містить усе необхідне й запускається безпосередньо.

Вимоги до клієнтської частини

Клієнтська частина не потребує встановлення програмного забезпечення — система працює у вебоглядачі.

– сучасний вебоглядач: Google Chrome, Microsoft Edge, Mozilla Firefox або Safari актуальної версії;
– пристрій (комп'ютер, ноутбук або планшет) з доступом до мережі підприємства;
– обліковий запис користувача в системі, наданий адміністратором.

Завдяки серверному відтворенню сторінок та використанню бібліотеки HTMX інтерфейс залишається чутливим навіть на пристроях з обмеженими ресурсами, оскільки основне обчислювальне навантаження припадає на сервер.

4.3 Склад інсталяційного пакету

Підготовлений інсталяційний пакет дозволяє розгорнути систему на сервері підприємства за визначеною послідовністю кроків. Пакет містить усі необхідні компоненти для повноцінної роботи серверної частини; окремого клієнтського пакета не передбачено, оскільки користувачі працюють через вебоглядач.

Склад інсталяційного пакету

– **Виконуваний файл застосунку** — скомпільований Go-бінарник серверної частини, що містить бізнес-логіку, обробники HTTP-запитів та вбудовані HTML-шаблони;

- **Скрипти ініціалізації бази даних** — SQL-файли schema.sql (створення таблиць, обмежень та індексів) і trigger.sql (тригерна функція перерахунку залишків), що виконуються одноразово при першому розгортанні;
- **Шаблон файлу конфігурації** — зразок файлу налаштувань із переліком необхідних параметрів: рядок підключення до бази даних, порт застосунку та параметри сесій;
- **Статичні ресурси** — файли стилів Bootstrap та бібліотеки HTMX, що поставляються разом із застосунком для роботи інтерфейсу;
- **Документація** — файл README.md з інструкцією з розгортання, переліком вимог та послідовністю кроків ініціалізації системи.

Особливості розгортання

1. на сервері встановлюється та запускається СУБД PostgreSQL, створюється порожня база даних для системи;
2. виконуються скрипти schema.sql та trigger.sql, що створюють структуру бази та тригер перерахунку залишків;
3. у файлі конфігурації зазначається рядок підключення до створеної бази даних та порт застосунку;
4. запускається виконуваний файл застосунку; за потреби налаштовується автозапуск як системної служби (systemd) та зворотний проксі для HTTPS;
5. створюється початковий обліковий запис адміністратора, після чого система готова до роботи; користувачі отримують доступ через вебоглядач за адресою сервера.

Завдяки компіляції застосунку в єдиний виконуваний файл розгортання значно простіше, ніж для систем із багатокомпонентною інфраструктурою: не потрібно встановлювати середовище виконання, менеджери пакетів чи додаткові залежності. Це знижує вимоги до кваліфікації персоналу, що здійснює впровадження, та зменшує ймовірність помилок конфігурації.

Таким чином, інсталяційний пакет забезпечує розгортання та налаштування серверної частини розробленої системи, що гарантує її готовність до експлуатації в робочому середовищі підприємства.

ВИСНОВКИ

У межах бакалаврської кваліфікаційної роботи на тему «Програмне забезпечення для автоматизованого обліку інвентарю та матеріалів на підприємстві» пройдено всі етапи створення програмного продукту — від дослідження предметної області та проєктування інформаційного забезпечення до програмної реалізації, тестування й підготовки інсталяційних артефактів.

Дослідження предметної області показало, що складський облік матеріалів на багатьох підприємствах досі ведеться вручну — у паперових картках та електронних таблицях, що призводить до неузгодженості даних, помилок під час введення та фінансових втрат через неточне визначення залишків. Особливо вразливим є облік матеріалів, кількість яких вимірюється дробовими величинами, де похибки округлення накопичуються непомітно. Огляд наявних систем (BAS Управління торгівлею, Zoho Inventory, inFlow Inventory) виявив, що комплексні рішення надлишкові й дорогі для підприємства, якому потрібен лише облік матеріалів, а закордонні хмарні сервіси прив'язані до зовнішніх серверів та валютної підписки. Це обґрунтувало доцільність розробки власної системи, зосередженої саме на складському обліку, з розгортанням у периметрі підприємства.

Для формалізації предметної області побудовано комплекс UML-моделей: діаграму прецедентів із розмежуванням ролей користувачів (комірник, бухгалтер, керівник), діаграму активності процесу руху цінностей та діаграму класів, що визначила основні сутності системи. На основі цих моделей сформовано постановку завдання з переліком вхідних і вихідних даних та функціональних і нефункціональних вимог.

Спроєктовано інформаційне забезпечення системи у вигляді реляційної моделі даних. Побудовано ER-діаграму із семи сутностей (користувачі, склади, контрагенти, номенклатура, залишки, документи руху та їхні рядки) і доведено її відповідність третій нормальній формі. Як систему управління базами даних обрано PostgreSQL — за підтримку точних числових типів для дробових кількостей, повний набір обмежень цілісності, транзакційність та тригери.

Контроль достатності залишку реалізовано обмеженням на рівні бази, а автоматичний перерахунок залишків при проведенні документа — тригерною функцією, що гарантує узгодженість даних незалежно від логіки застосунку.

Програмне забезпечення реалізовано за багатошаровою архітектурою мовою Go. Шар обробників приймає HTTP-запити, шар бізнес-логіки виконує операції предметної області, а шар доступу до даних інкапсулює роботу з базою через драйвер pgx. Інтерфейс побудовано на серверному відтворенні сторінок засобами Go Templates із застосуванням бібліотеки HTMX для часткового оновлення сторінки без повного перезавантаження та фреймворку Bootstrap для оформлення. Такий підхід зосереджує логіку на стороні сервера й уникає складності повноцінних клієнтських фреймворків.

Центральною функцією системи є проведення документів руху з автоматичним контролем залишків. Під час проведення система в межах однієї транзакції перераховує залишки відповідних позицій, блокує спроби списати більше, ніж наявно на складі, та сигналізує про досягнення мінімального рівня запасу. Це усуває рутинні ручні звіряння та знижує вплив людського фактору на достовірність обліку.

Розроблена система перевірена методами модульного та системного тестування, а також тестуванням «чорної скриньки» за сценарієм типової роботи користувача. Підготовлено інсталяційний пакет для розгортання серверної частини. Завдяки компіляції застосунку в єдиний виконуваний файл розгортання значно простіше, ніж для систем із багатокомпонентною інфраструктурою: достатньо запустити бінарник та підключити базу даних, без встановлення середовища виконання чи додаткових залежностей. Користувачі працюють із системою через вебоглядач без встановлення клієнтського програмного забезпечення.

Сформульовану у вступі мету досягнуто, а поставлені дослідницькі та інженерні завдання — виконано. Розроблена система автоматизує ключові процеси складського обліку матеріалів, поєднуючи централізоване зберігання даних, контроль залишків з фіксованою точністю, розмежування ролей користувачів та просте розгортання у периметрі підприємства.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бутинець Ф. Ф. Бухгалтерський фінансовий облік: підручник. 8-ме вид. Житомир: ПП «Рута», 2009. 912 с.

2. Про затвердження Положення (стандарту) бухгалтерського обліку 9 «Запаси»: наказ Міністерства фінансів України від 20.10.1999 № 246. URL: https://zakon.rada.gov.ua/laws/show/z0751-99 (дата звернення: 18.05.2026).

3. Дудзяний І. М. Об'єктно-орієнтоване моделювання програмних систем: навчальний посібник. Львів: Видавничий центр ЛНУ імені Івана Франка, 2007. 108 с.

4. OMG. OMG® Unified Modeling Language® (OMG UML) Specification, Version 2.5.1. Object Management Group, 2017. URL: https://www.omg.org/spec/UML/2.5.1/ (дата звернення: 18.05.2026).

5. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston: Addison-Wesley Professional, 2003. 208 p.

6. BAS Управління торгівлею: опис рішення. URL: https://www.bas-soft.eu/soft/bas-corp/bas-upravlinnja-torgivleju/ (дата звернення: 18.05.2026).

7. Zoho Inventory: Inventory Management Software. URL: https://www.zoho.com/inventory/ (дата звернення: 18.05.2026).

8. inFlow Inventory: Inventory Management Software. URL: https://www.inflowinventory.com/ (дата звернення: 18.05.2026).

9. Чен П. П. Модель «сутність-зв'язок» — крок до єдиного бачення даних. ACM Transactions on Database Systems. 1976. Т. 1, № 1. С. 9–36.

10. Date C. J. An Introduction to Database Systems. 8th ed. Boston: Pearson/Addison-Wesley, 2003. 1024 p.

11. The PostgreSQL Global Development Group. PostgreSQL 16 Documentation. URL: https://www.postgresql.org/docs/16/ (дата звернення: 18.05.2026).

12. The PostgreSQL Global Development Group. PostgreSQL: Trigger Functions (PL/pgSQL). URL: https://www.postgresql.org/docs/current/plpgsql-trigger.html (дата звернення: 18.05.2026).

13. The Go Programming Language Specification. URL: https://go.dev/ref/spec (дата звернення: 18.05.2026).

14. Donovan A. A., Kernighan B. W. The Go Programming Language. Boston: Addison-Wesley Professional, 2015. 380 p.

15. HTMX: high power tools for HTML. Documentation. URL: https://htmx.org/docs/ (дата звернення: 18.05.2026).

16. Bootstrap: The most popular HTML, CSS, and JS library. Documentation. URL: https://getbootstrap.com/docs/ (дата звернення: 18.05.2026).

17. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. 2nd ed. Boston: Addison-Wesley Professional, 2005. 475 p.

18. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3rd ed. Upper Saddle River: Prentice Hall, 2004. 736 p.

19. Sommerville I. Software Engineering. 10th ed. Boston: Pearson, 2015. 816 p.

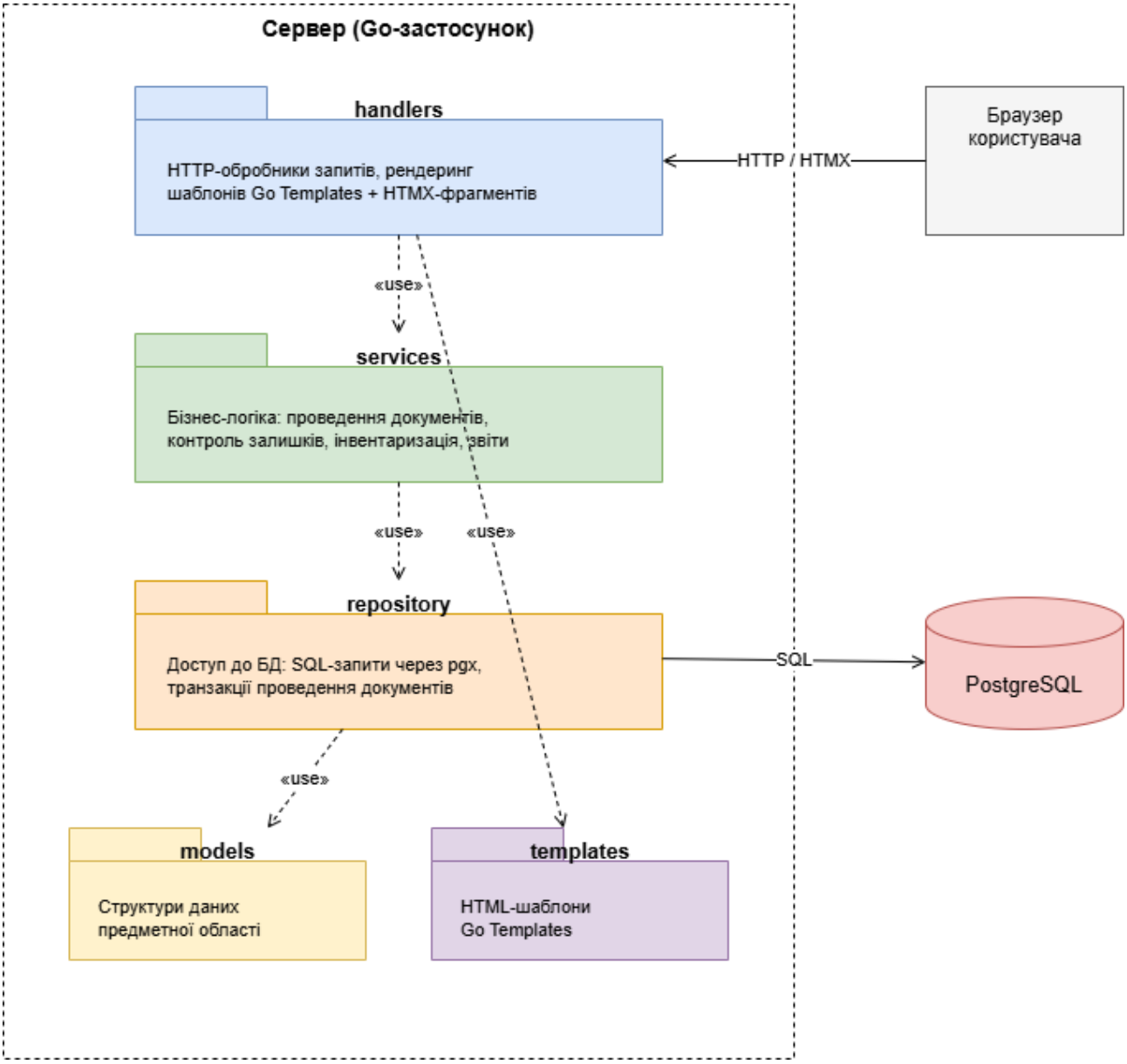
20. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 8th ed. New York: McGraw-Hill, 2014. 976 p.

21. Beizer B. Black-Box Testing: Techniques for Functional Testing of Software and Systems. New York: John Wiley & Sons, 1995. 320 p.

ДОДАТКИ

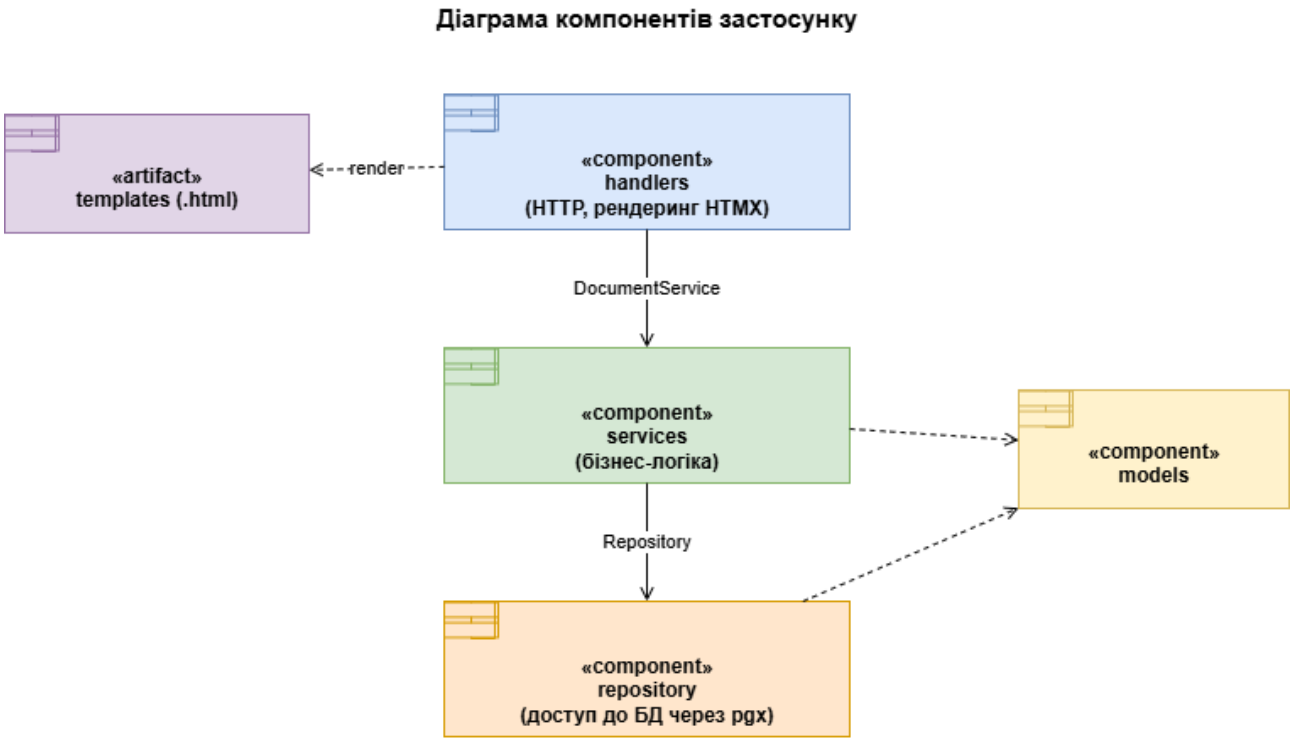
Додаток А

Діаграма пакетів



Додаток В

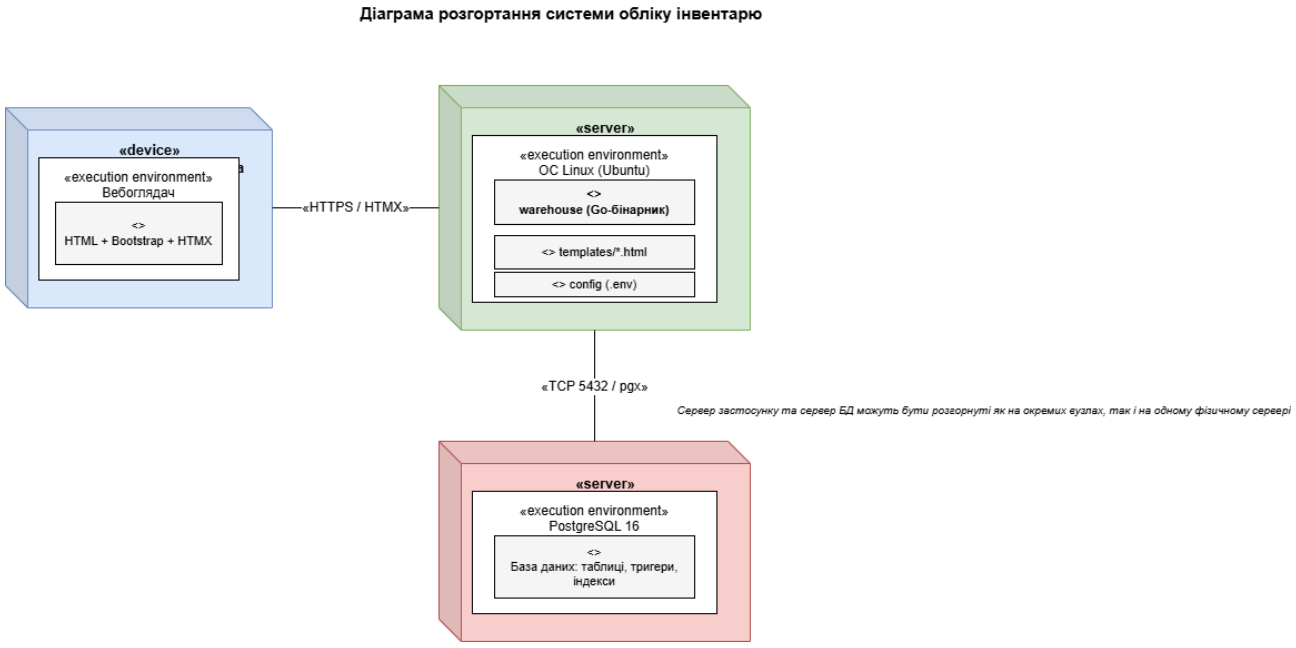
Діаграма компонентів



Залежності спрямовані зверху вниз: кожен шар використовує лише сусідній нижчий. models — спільні структури даних, templates — HTML-шаблони представлення.

Додаток Д

Діаграма розгортання



Додаток Ж

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Юманов Андрій Дмитрович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення для автоматизованого обліку інвентарю та матеріалів на підприємстві» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

- [ ] інструменти генеративного ШІ не використовувалися.
- [×] інструменти ШІ (ChatGPT) були використані для:
  - [×] перекладу іншомовних джерел;
  - [×] стилістичного редагування та перевірки граматики;
  - [×] допомоги у написанні/відлагодженні програмного коду;
  - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

« » 2026 р. _____ **Андрій ЮМАНОВ**

(підпис)