НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ 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 ідентифікації об'єктів за зображенням»

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми к.т.н., доцент	_____ (підпис)	**Ганна ВАЙГАНГ**
Керівник бакалаврської кваліфікаційної роботи старший викладач	_____ (підпис)	**Юрій МІЛОВІДОВ**
Виконав	_____ (підпис)	**Іван ФІЛОНЕНКО**

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

Філоненку Іван Олександровичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи

«Програмне забезпечення для ідентифікації об'єктів за зображенням»

затверджена наказом від «19» грудня 2025 р. № 3204 «С»
Термін подання завершеної роботи на кафедру «22» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту): Правила призначення академічних стипендій у Національному університеті біоресурсів і природокористування України від 22.11.2023, Форма додатка до диплома європейського зразка.
Перелік питань, що підлягають дослідженню:
Інтелектуальна каса самообслуговування як об'єкт дослідження, проєктування інформаційного забезпечення, розробка програмного забезпечення, рекомендації щодо впровадження та експлуатації системи.
Дата видачі завдання «19» грудня 2026 р.

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Юрій МІЛОВІДОВ**

Завдання взяв до виконання _____ (підпис) **Іван ФІЛОНЕНКО**

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

КСО — каса самообслуговування

ПЗ — програмне забезпечення

БД — база даних

СУБД — система управління базами даних

API — Application Programming Interface

UI — User Interface

UML — Unified Modeling Language

ER — Entity-Relationship

SQL — Structured Query Language

JSON — JavaScript Object Notation

HTTP — HyperText Transfer Protocol

REST — Representational State Transfer

ORB — Oriented FAST and Rotated BRIEF

OpenCV — Open Source Computer Vision Library

ВСТУП

Актуальність. У сучасній роздрібній торгівлі дедалі більшого значення набувають технології самообслуговування, які дозволяють прискорити процес купівлі товарів, зменшити навантаження на персонал магазину та підвищити зручність для покупців [1]. Традиційні касові системи залишаються ефективними, однак вони часто потребують участі касира або обов'язкового сканування штрихкоду кожного товару. Це може створювати черги, ускладнювати процес обслуговування у години пікового навантаження та знижувати загальну швидкість здійснення покупки.

Одним із напрямів розвитку систем самообслуговування є використання комп'ютерного зору та програмних засобів розпізнавання об'єктів за зображенням [2]. Такий підхід дає змогу ідентифікувати товар не лише за штрихкодом, а й за його зовнішнім виглядом. Це є актуальним для інтелектуальних кас самообслуговування, де користувач може самостійно розмістити товар перед камерою, запустити сканування та отримати результат розпізнавання без постійної участі працівника магазину.

Водночас системи розпізнавання об'єктів мають певні обмеження. Якість їхньої роботи залежить від освітлення, положення товару, фону, якості камери та наявності еталонних зображень [2]. Тому в межах даної роботи розглядається контрольований сценарій ідентифікації одного товару за кадром, отриманим із вебкамери.

Мета розробки. Мета розробки полягає в автоматизації окремих етапів процесу самообслуговування покупця в роздрібній торгівлі на основі створення програмного забезпечення інтелектуальної каси самообслуговування з функцією розпізнавання товару за зображенням.

Основна ідея полягає в тому, щоб зменшити залежність користувача від ручного пошуку товару в каталозі або сканування штрихкоду. Розроблене програмне забезпечення дозволяє користувачу розмістити один товар перед вебкамерою, запустити сканування та отримати результат розпізнавання в

інтерфейсі системи. У разі підтвердженого результату товар автоматично додається до віртуального чека.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати предметну область систем самообслуговування в роздрібній торгівлі;
- розглянути існуючі аналогічні рішення та технології розпізнавання об’єктів;
- сформувати функціональні й нефункціональні вимоги до програмного забезпечення;
- спроєктувати інформаційне забезпечення системи;
- розробити клієнтську та серверну частини програмного продукту;
- реалізувати механізм отримання кадру з вебкамери та передавання його на серверну частину;
- реалізувати механізм розпізнавання товару за еталонними зображеннями;
- забезпечити відображення результату розпізнавання в інтерфейсі;
- реалізувати адміністративний режим для додавання об’єктів каталогу та еталонних зображень;
- провести тестування основних сценаріїв роботи системи.

Об’єктом дослідження є процес самообслуговування покупця в роздрібній торгівлі з використанням програмних систем кас самообслуговування.

Предметом дослідження є методи, моделі та програмні засоби розпізнавання товару за зображенням у веборієнтованій системі інтелектуальної каси самообслуговування.

Методи та технології. Під час виконання роботи використано методи аналізу предметної області, моделювання програмних систем, об’єктно-орієнтованого проєктування, проєктування інформаційного забезпечення, клієнт-серверної взаємодії та тестування програмного забезпечення. Для опису структури й поведінки системи застосовано UML-діаграми [3].

Програмне забезпечення реалізовано у вигляді клієнт-серверного вебзастосунку. Для клієнтської частини використано React, TypeScript, Vite і Tailwind CSS. Серверна частина побудована з використанням Python, FastAPI та OpenCV. Розпізнавання реалізовано на основі OpenCV ORB feature matching, тобто порівняння візуальних ознак отриманого кадру з еталонними зображеннями. Для зберігання структурованих даних використовується SQLite, а еталонні зображення зберігаються у файловій структурі серверної частини.

Поточна версія програмного забезпечення підтримує локальний запуск і хмарне розгортання. Клієнтська та серверна частини розгорнуті як окремі вебзастосунки в Azure App Service, що дозволяє демонструвати роботу системи через браузер.

Апробація результатів роботи. Результати розробки були апробовані на VIII Всеукраїнській науково-практичній конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп’ютерних систем 2026». За результатами роботи було подано тези з назвою «Програмне забезпечення для ідентифікації об'єктів за зображенням», а основні результати дослідження були представлені у вигляді доповіді.

Записка кваліфікаційної роботи складається зі вступу, чотирьох розділів та висновків. Список використаних джерел та додатки винесені в кінці записки. Загальний обсяг записки – 78 сторінки основного тексту, кількість використаних джерел – 27, кількість додатків – 13, у тому числі Декларація про академічну доброчесність та використання ШІ у додатку Р.

У першому розділі розглянуто інтелектуальну касу самообслуговування як об’єкт дослідження, проаналізовано предметну область, змодельовано основні процеси, розглянуто існуючі аналогічні рішення та сформульовано постановку завдання.

У другому розділі спроєктовано інформаційне забезпечення системи: побудовано логічну модель даних, обґрунтовано вибір системи управління базою даних, описано структуру інформаційної бази та її фізичну реалізацію.

У третьому розділі описано розробку програмного забезпечення: абстракції предметної області, структуру та взаємодію об’єктів, організаційну й компонентну структуру системи, вибір інструментарію та алгоритмізацію основних програмних модулів.

У четвертому розділі наведено результати тестування системи, визначено вимоги до апаратного та програмного забезпечення, а також описано склад інсталяційного пакету й особливості експлуатації програмного продукту.

1 ІНТЕЛЕКТУАЛЬНА КАСА САМООБСЛУГОВУВАННЯ ЯК ОБ'ЄКТ ДОСЛІДЖЕННЯ

1.1 Аналіз систем самообслуговування в роздрібній торгівлі як предметної області

Роздрібна торгівля є однією з галузей, у якій швидкість і зручність обслуговування покупців мають важливе значення. У магазинах і супермаркетах щоденно виконується велика кількість однотипних операцій: вибір товару, його ідентифікація, визначення ціни, формування чека, оплата та завершення покупки. У традиційній моделі ці дії виконуються за участі касира, який сканує товар, перевіряє його назву та ціну, формує підсумкову суму покупки й приймає оплату від покупця.

З розвитком інформаційних технологій у роздрібній торгівлі поширення набули системи самообслуговування. Вони дають змогу покупцю самостійно виконувати частину касових операцій без постійної участі працівника магазину. Одним із прикладів таких систем є каси самообслуговування (КСО), які дозволяють користувачу сканувати товари, переглядати список доданих позицій, контролювати загальну суму покупки та переходити до оплати [1]. Працівник магазину в такому процесі виконує переважно допоміжну або контролюючу функцію.

Типовий процес використання КСО починається з того, що покупець підходить до пристрою та розпочинає оформлення покупки. Далі він послідовно ідентифікує товари за допомогою сканера штрихкодів або електронного каталогу. Після додавання позиції система відображає назву товару, його кількість, ціну та загальну суму покупки. Після завершення сканування покупець переходить до етапу оплати, а система формує чек і завершує операцію. Такий сценарій дозволяє зменшити навантаження на звичайні каси та надати покупцю більше самостійності.

Водночас традиційні КСО мають певні обмеження. Основним способом ідентифікації товару в більшості таких систем залишається сканування штрихкоду [4]. Якщо штрихкод пошкоджений, закритий упаковкою, погано

надрукований або користувач неправильно розміщує товар перед сканером, процес оформлення покупки ускладнюється. У таких випадках покупець змушений повторювати сканування, шукати товар у каталозі вручну або звертатися по допомогу до працівника магазину.

Ще одним обмеженням є залежність процесу самообслуговування від правильності дій користувача. Для покупців, які не мають досвіду роботи з КСО, пошук штрихкоду, вибір товару з каталогу або виправлення помилок може бути незручним. Через це виникає потреба в удосконаленні способів ідентифікації товарів та спрощенні взаємодії користувача з касовою системою.

Одним із перспективних напрямів розвитку КСО є використання комп'ютерного зору, тобто програмних методів аналізу зображень для розпізнавання об'єктів [2]. У такому випадку товар може бути ідентифікований не лише за штрихкодом, а й за зовнішнім виглядом. Покупець розміщує товар перед камерою, система отримує зображення, аналізує його та визначає, який об'єкт знаходиться в кадрі. Такий підхід може зробити процес самообслуговування більш природним, оскільки користувачу не потрібно шукати штрихкод або вручну вибирати позицію з каталогу.

У предметній області інтелектуальної КСО можна виділити кілька основних учасників і об'єктів. Покупець взаємодіє з інтерфейсом системи, розміщує товар перед засобом ідентифікації та переглядає результат. Працівник магазину або адміністратор може допомагати користувачам, контролювати роботу системи та підтримувати актуальність товарних даних. Основними об'єктами предметної області є товар, засіб отримання зображення, інтерфейс каси самообслуговування, модуль ідентифікації товару, товарні дані та віртуальний чек.

Основним об'єктом обробки в такій системі є товар. Для коректної роботи КСО товар має бути пов'язаний із певними даними: назвою, ціною, описом та візуальними або іншими ознаками, які можуть використовуватися для його ідентифікації. Після розпізнавання система повинна показати користувачу зрозумілий результат. Якщо товар визначено коректно, відповідна позиція може

бути відображена у віртуальному чеку. Якщо результат є некоректним або недостатньо впевненим, система має передбачати повторне сканування або інше повідомлення для користувача.

Для використання розпізнавання за зображенням важливими є умови роботи системи: якість освітлення, положення товару, фон, якість камери та наявність еталонних зображень [2]. На відміну від звичайного сканування штрихкоду, аналіз зображення потребує врахування зовнішнього вигляду товару та умов зйомки. Тому такі системи доцільно розглядати як інтелектуальне доповнення до традиційних КСО, а не як повну заміну всіх існуючих касових механізмів.

Таким чином, інтелектуальна каса самообслуговування є програмно-апаратною системою, що поєднує процеси самообслуговування покупця, роботу з товарними даними, отримання зображення та програмну ідентифікацію об’єкта. Її використання дозволяє розглядати новий підхід до розпізнавання товарів у роздрібній торгівлі, де важливу роль відіграють не лише штрихкоди, а й аналіз візуальних ознак товару. Саме тому дослідження та розробка програмного забезпечення інтелектуальної КСО є актуальним завданням у межах інженерії програмного забезпечення.

1.2 Моделювання предметної області

Для подальшого проєктування програмного забезпечення необхідно виконати моделювання предметної області каси самообслуговування. Моделювання дозволяє формалізувати основні процеси, учасників, послідовність дій та взаємозв’язки між елементами, які беруть участь в оформленні покупки. Завдяки цьому можна краще зрозуміти логіку роботи КСО ще до переходу до проєктування інформаційного забезпечення та програмної реалізації.

У межах предметної області каси самообслуговування основним процесом є самостійне оформлення покупки покупцем. Покупець розпочинає покупку, сканує товари за допомогою сканера штрихкодів, переглядає сформований чек,

за потреби коригує його, переходить до оплати та завершує покупку. У разі виникнення помилки або труднощів покупець може звернутися по допомогу до працівника магазину. Окрему роль виконує адміністратор, який відповідає за оновлення бази даних товарів, налаштування системи та контроль її працездатності.

Для опису предметної області використано UML-діаграми, які дають змогу розглянути КСО з різних точок зору. Діаграма прецедентів відображає основних акторів і функції, які вони виконують. Діаграма послідовності показує порядок взаємодії між учасниками та елементами системи під час оформлення покупки. На основі діаграми послідовності побудовано діаграму активності, яка подає загальний потік дій і можливі розгалуження процесу [3].

1.2.1 Діаграма прецедентів. Діаграма прецедентів використовується для відображення основних учасників предметної області та дій, які вони виконують у межах каси самообслуговування. Така діаграма дозволяє визначити межі системи, основні ролі користувачів і типові сценарії взаємодії з КСО.

У предметній області каси самообслуговування доцільно виділити трьох основних акторів: покупця, працівника магазину та адміністратора. Покупець є головним учасником процесу, оскільки саме він самостійно оформлює покупку. До його основних дій належать початок покупки, сканування товарів, перегляд чека, коригування чека, оплата покупки, отримання чека та звернення по допомогу в разі потреби.

Працівник магазину бере участь у процесі тоді, коли покупець стикається з помилкою або не може самостійно завершити дію. Його основним прецедентом є надання допомоги покупцю. Адміністратор відповідає за підтримку роботи КСО: оновлення бази даних товарів, налаштування системи та контроль її функціонування. Було побудовано діаграму прецедентів (рис. 1.1)

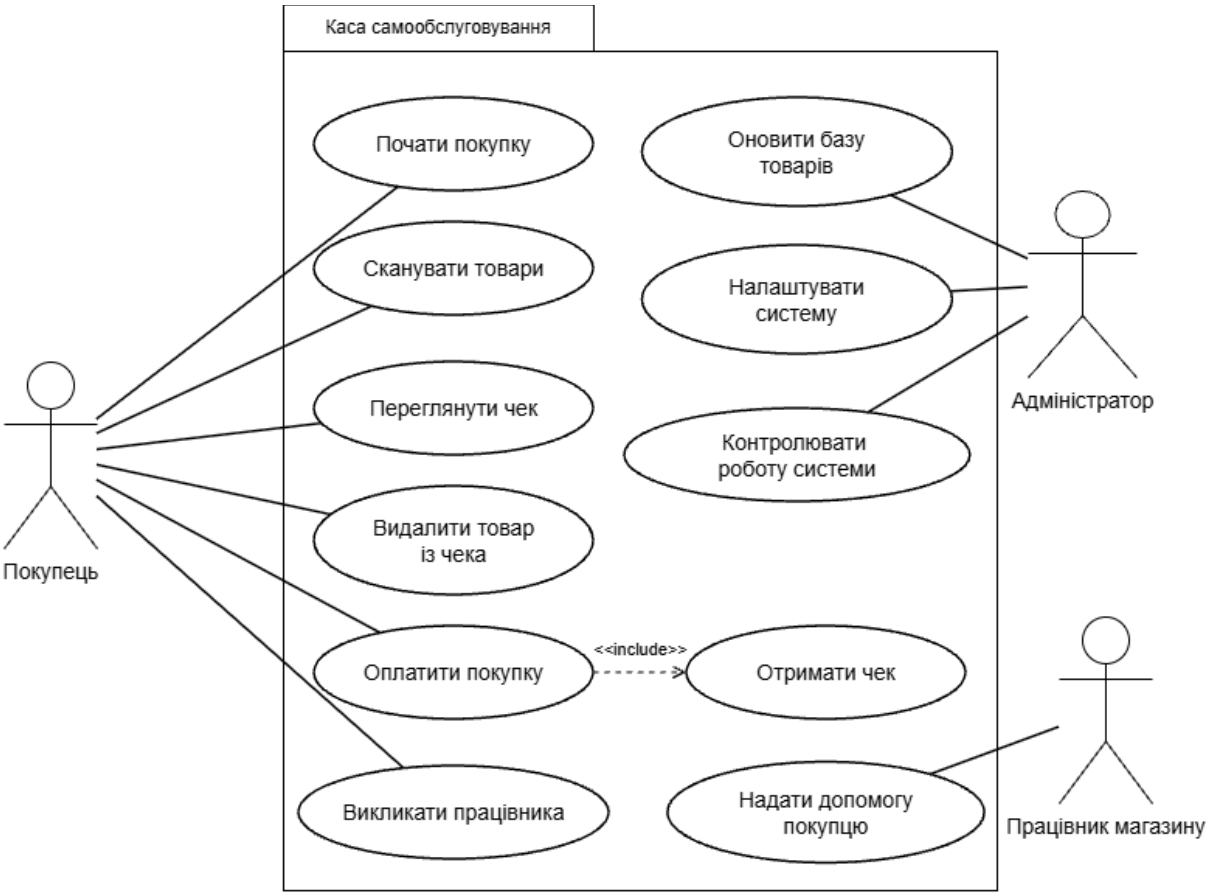


Рис. 1.1 Діаграма прецедентів предметної області каси самообслуговування

Діаграма прецедентів демонструє, що основна взаємодія з КСО зосереджена навколо покупця та процесу оформлення покупки. Водночас система не є повністю автономною, оскільки для її стабільної роботи потрібні працівник магазину та адміністратор. Працівник магазину забезпечує підтримку користувача в нестандартних ситуаціях, а адміністратор підтримує актуальність бази даних товарів і налаштувань системи.

1.2.2 Діаграма послідовності. Діаграма послідовності використовується для моделювання взаємодії між учасниками та об’єктами предметної області в часі. Вона дозволяє показати, у якому порядку відбуваються повідомлення та дії під час типового сценарію оформлення покупки через КСО.

У цьому сценарії основними учасниками є покупець, інтерфейс КСО, сканер штрихкодів, база даних товарів, чек, платіжний модуль і працівник магазину. Покупець розпочинає покупку через інтерфейс КСО, після чого переходить до сканування товарів. Після ідентифікації товару система отримує

відповідні товарні дані, звертається до бази даних товарів, знаходить відповідну позицію, отримує її назву та ціну, після чого додає товар до чека.

Після сканування товарів покупець переглядає чек. Якщо чек сформовано правильно, покупець переходить до оплати. Платіжний модуль обробляє оплату та повертає системі результат. Після успішної оплати КСО надає покупцю чек. Якщо під час перевірки чека виявлено помилку, покупець може видалити товар із чека або викликати працівника магазину. Працівник магазину аналізує ситуацію, допомагає покупцю виправити помилку, після чого процес оформлення покупки може бути продовжений. Було побудовано діаграму послідовності (рис. 1.2)

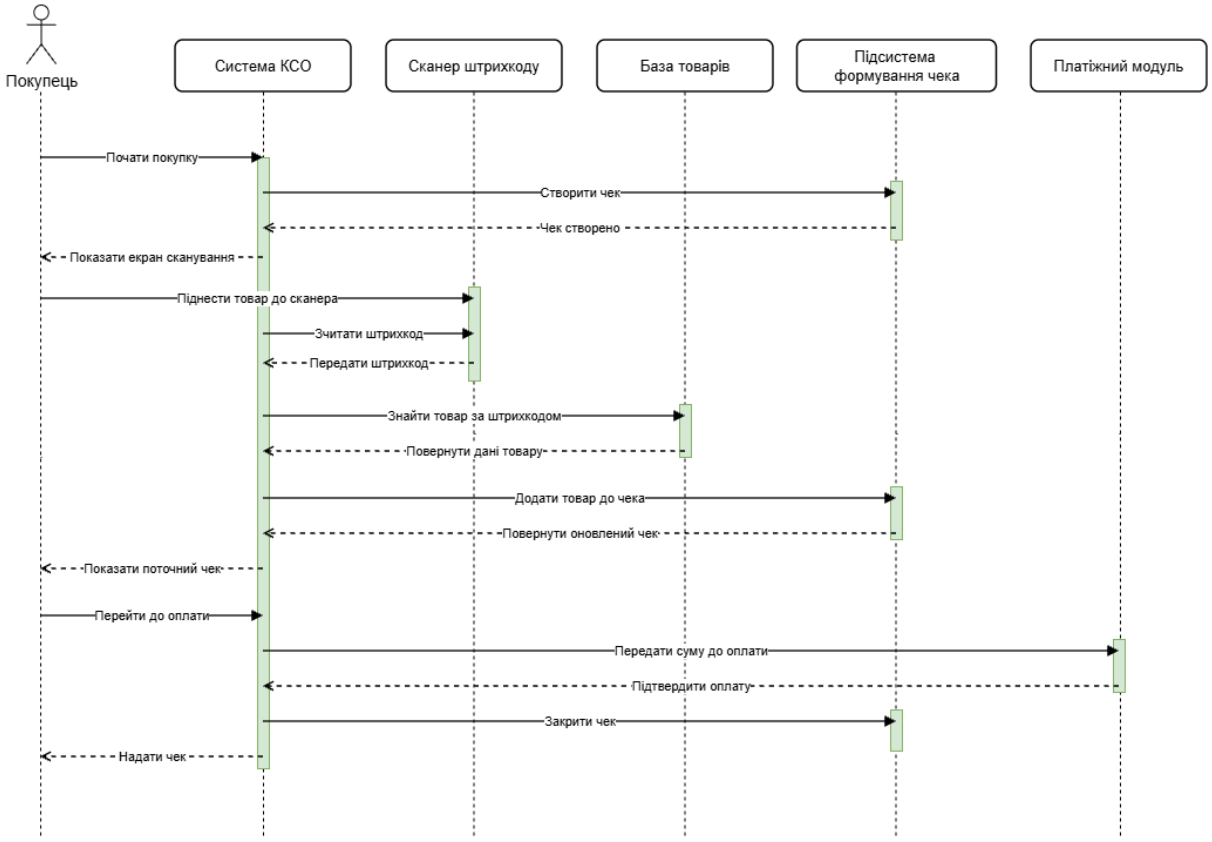


Рис. 1.2 Діаграма послідовності процесу оформлення покупки через касу самообслуговування

Діаграма послідовності показує часову логіку взаємодії між покупцем, КСО та допоміжними елементами предметної області. Особливу роль у типовому процесі відіграє етап сканування товару, оскільки саме після нього система отримує товарні дані та формує позицію в чеку. Таким чином, діаграма

дозволяє зрозуміти, як окремі дії покупця перетворюються на послідовність операцій у межах КСО.

1.2.3 Діаграма активності. На основі діаграми послідовності доцільно побудувати діаграму активності, яка відображає загальний потік дій під час оформлення покупки через касу самообслуговування. Якщо діаграма послідовності показує взаємодію між учасниками в часі, то діаграма активності зосереджується на логіці процесу, умовах переходу між діями та можливих розгалуженнях.

Процес оформлення покупки починається з того, що покупець розпочинає роботу з КСО. Далі він переходить до сканування товару, після чого система отримує відповідні товарні дані та додає позицію до чека. Після додавання товарів покупець перевіряє чек.

Якщо чек сформовано коректно, покупець переходить до оплати, виконує оплату, після чого система відображає повідомлення про успішне завершення операції та надає чек. Якщо чек сформовано некоректно, покупець може видалити помилково доданий товар або викликати працівника магазину. Після видалення товару або отримання допомоги працівника покупець повторно перевіряє чек і продовжує процес оформлення покупки. Графічне подання логіки цього процесу у вигляді діаграми активності наведено в додатку А.

Діаграма активності демонструє, що процес роботи з КСО має послідовний характер, але містить розгалуження у випадку помилок або необхідності коригування чека. Основний сценарій передбачає сканування товарів, перевірку чека, оплату та отримання чека. Альтернативні сценарії пов’язані з видаленням товару або зверненням по допомогу до працівника магазину. Така модель дозволяє краще зрозуміти логіку предметної області та визначити операції, які можуть бути автоматизовані в програмному забезпеченні.

Таким чином, у межах підрозділу було побудовано моделі предметної області каси самообслуговування. Діаграма прецедентів визначає основних акторів і їхні дії, діаграма послідовності показує часову взаємодію учасників процесу, а діаграма активності узагальнює логіку оформлення покупки.

Побудовані моделі є основою для подальшого аналізу існуючих рішень, постановки завдання та проєктування програмного забезпечення інтелектуальної КСО.

1.3 Огляд існуючих аналогічних систем і технологій розпізнавання об’єктів

У сучасній роздрібній торгівлі існує значна кількість рішень, спрямованих на автоматизацію процесу купівлі товарів. Частина з них є класичними касами самообслуговування, де покупець самостійно сканує штрихкод товару, формує чек та здійснює оплату. Інша частина рішень використовує більш складні технології: комп’ютерний зір, машинне навчання, сенсорні системи та автоматичне розпізнавання товарів. Огляд таких систем дозволяє визначити їхні переваги, недоліки та місце розроблюваного програмного забезпечення серед існуючих аналогів.

NCR Voyix Self-Checkout. Одним із відомих рішень у сфері кас самообслуговування є NCR Voyix Self-Checkout. Це промислова система самообслуговування для роздрібної торгівлі, яка орієнтована на організацію самостійного процесу сканування, пакування та оплати товарів покупцем. Виробник позиціонує такі рішення як засіб оптимізації процесу купівлі та підвищення зручності покупця. Окремо NCR Voyix виділяє технологію Picklist Assist, яка використовує машинне навчання та комп’ютерний зір для спрощення пошуку товарів без штрихкоду або PLU-кодів [1].

Рис. 1.4 Зовнішній вигляд системи NCR Voyix Self-Checkout

Перевагами NCR Voyix Self-Checkout є промисловий рівень реалізації, підтримка різних конфігурацій обладнання, інтеграція з POS-інфраструктурою магазину та орієнтація на використання у реальних торговельних мережах. Такі системи можуть бути адаптовані під різні формати магазинів і підтримувати різні варіанти касового обладнання.

Недоліком такого рішення є його складність і залежність від спеціалізованого обладнання. Для впровадження потрібна повноцінна касова інфраструктура, інтеграція з внутрішніми системами магазину, налаштування товарної бази та обслуговування обладнання. Для невеликого навчального або прототипного проєкту така система є надмірно складною. Крім того, основний сценарій традиційної КСО часто залишається пов’язаним зі скануванням штрихкоду, тоді як у даній роботі розглядається окремий контрольований сценарій розпізнавання товару за зображенням із вебкамери.

Diebold Nixdorf DN Series EASY ONE. Ще одним прикладом промислового рішення є Diebold Nixdorf DN Series EASY ONE. Це модульна система, яка підтримує режими self-service, assisted-service та semi-assisted-service. Виробник зазначає, що рішення підходить не лише для продуктової

роздрібної торгівлі, а й для інших напрямів, зокрема магазинів одягу, паливних станцій, поштових сервісів та закладів швидкого обслуговування [5].

Рис. 1.5 Зовнішній вигляд системи Diebold Nixdorf DN Series EASY ONE

Перевагою DN Series EASY ONE є модульність. Система може мати різні конфігурації та підтримувати роботу в різних сценаріях обслуговування покупця. Також вона може поєднувати сенсорні екрани, сканери, принтери, платіжні термінали, периферійні пристрої та програмне забезпечення для інтеграції з POS-додатками.

Недоліком є те, що така система є комплексним комерційним рішенням, яке потребує спеціального обладнання та інтеграції з торговельною інфраструктурою. Вона добре підходить для реальних магазинів, але її складність є надмірною для задачі створення навчального програмного продукту.

Mashgin AI Checkout. Найбільш близьким за ідеєю до теми даної роботи є рішення Mashgin AI Checkout. На відміну від класичних КСО, Mashgin робить акцент саме на використанні штучного інтелекту та комп’ютерного зору. Користувач розміщує товари в зоні сканування, після чого система аналізує їх за

допомогою камер і автоматично визначає позиції без необхідності ручного сканування штрихкодів. Виробник зазначає, що система використовує кілька 3D-камер, які формують модель товарів і дають змогу ідентифікувати позиції за зовнішніми ознаками [6].

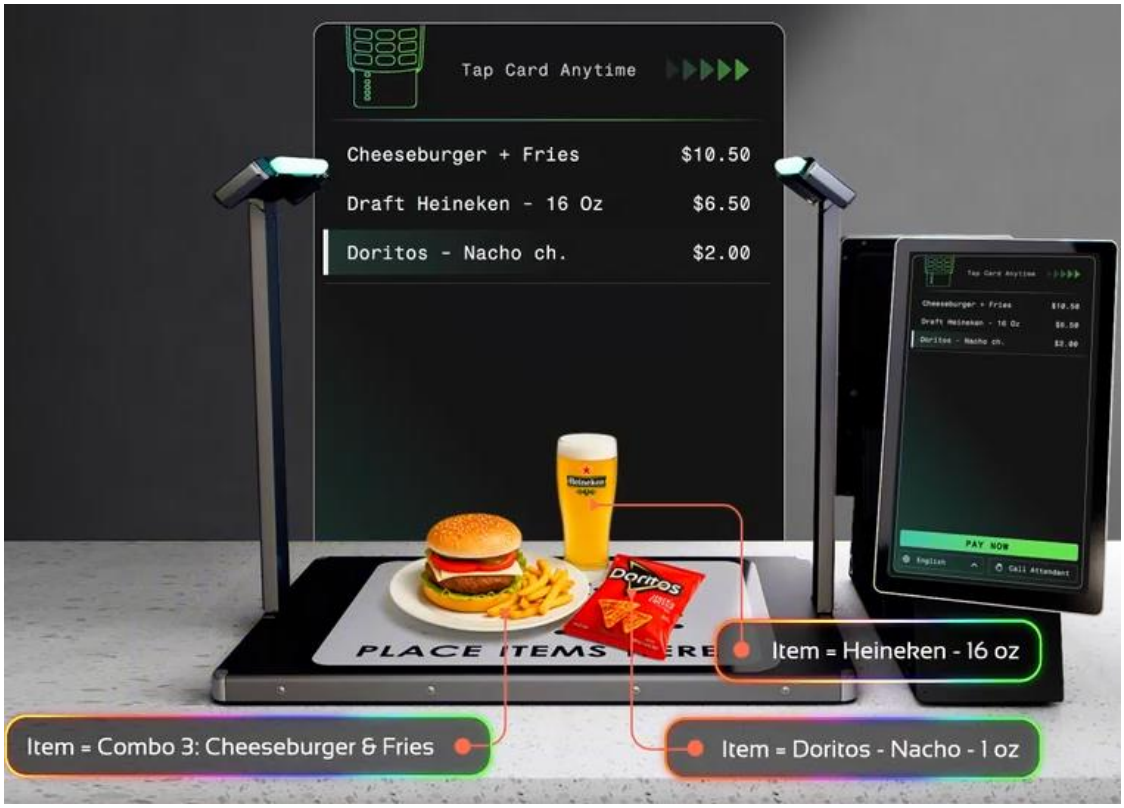


Рис. 1.6 Зовнішній вигляд системи Mashgin AI Checkout

Перевагою Mashgin є природніший сценарій взаємодії покупця з касою: користувачу не потрібно шукати штрихкод на кожному товарі, оскільки система орієнтована на розпізнавання об’єктів за зовнішнім виглядом. Також виробник зазначає можливість швидкого додавання нових товарів і роботу без ручного сканування штрихкодів.

Недоліком Mashgin є закритість технології та залежність від спеціалізованого апаратного забезпечення. Для роботи використовуються не звичайні вебкамери, а спеціально спроєктоване обладнання, зокрема кілька 3D-камер. Тому в даній роботі використовується спрощений підхід: розпізнавання одного товару за кадром, отриманим із вебкамери, після натискання користувачем кнопки сканування.

Toshiba MxP Self-Checkout. Ще одним прикладом класичної каси самообслуговування є Toshiba MxP Self-Checkout. Це промислове рішення для роздрібної торгівлі, орієнтоване на організацію самостійного процесу оформлення покупки покупцем. Toshiba описує MxP як модульну платформу, що дозволяє підібрати self-checkout-рішення відповідно до потреб конкретного торговельного середовища [7].

Рис. 1.7 Зовнішній вигляд системи Toshiba MxP Self-Checkout

Перевагою Toshiba MxP Self-Checkout є модульність і орієнтація на різні формати роздрібної торгівлі. Такі системи можуть використовуватися як частина повноцінної касової інфраструктури магазину, підтримувати різні варіанти обладнання та забезпечувати зручний сценарій самообслуговування для покупця.

Недоліком такого рішення, як і в інших промислових КСО, є залежність від спеціалізованого обладнання та необхідність інтеграції з торговельною інфраструктурою. Для впровадження потрібні касові пристрої, периферійне

обладнання, налаштування товарної бази та підтримка з боку постачальника або IT-служби магазину.

Для узагальнення розглянутих рішень можна зазначити, що NCR Voyix Self-Checkout, Diebold Nixdorf DN Series EASY ONE і Toshiba MxP Self-Checkout є прикладами промислових кас самообслуговування, орієнтованих на реальне використання в торговельних мережах. Їхніми перевагами є надійність, модульність, підтримка касового обладнання та інтеграція з POS-інфраструктурою. Водночас такі системи потребують спеціалізованого обладнання, налаштування торговельної інфраструктури та технічного супроводу. Mashgin AI Checkout є ближчим до теми даної роботи за ідеєю автоматичного розпізнавання товарів, однак також залежить від спеціалізованого апаратного забезпечення та закритої технології. На відміну від цих рішень, розроблюване програмне забезпечення є спрощеним веборієнтованим прототипом, який демонструє контрольований сценарій розпізнавання одного товару за зображенням із вебкамери.

Огляд існуючих рішень показує, що сучасні системи самообслуговування розвиваються у двох основних напрямах. Перший напрям пов’язаний з удосконаленням класичних КСО, де основою залишається самостійне сканування товарів, але покращуються апаратна модульність, інтеграція з POS-системами, зручність інтерфейсу та засоби контролю. Другий напрям пов’язаний із використанням комп’ютерного зору та інтелектуальних алгоритмів для зменшення кількості ручних дій покупця під час ідентифікації товарів.

Розроблюване програмне забезпечення займає проміжне місце між цими підходами. Воно не претендує на повну заміну промислових КСО, однак демонструє базовий сценарій інтелектуальної каси самообслуговування: користувач розміщує один товар перед вебкамерою, запускає сканування, система отримує зображення, передає його на серверну частину та повертає результат розпізнавання. Такий підхід є простішим за промислові аналоги, але достатнім для дослідження принципів побудови програмного забезпечення КСО з функцією розпізнавання товару за зображенням.

1.4 Постановка завдання

Метою даного проєкту є розробка програмного забезпечення інтелектуальної каси самообслуговування, яке забезпечує ідентифікацію одного товару за зображенням, отриманим із вебкамери. Система повинна надати користувачу можливість запустити сканування, передати отриманий кадр на серверну частину, отримати результат розпізнавання та відобразити його в інтерфейсі каси самообслуговування.

Розроблюване програмне забезпечення має клієнт-серверну структуру. Клієнтська частина відповідає за взаємодію з користувачем, роботу з вебкамерою, запуск сканування, відображення результату та формування віртуального чека. Серверна частина приймає зображення, виконує його обробку, порівнює з еталонними зображеннями, формує відповідь для клієнтської частини та зберігає службову інформацію про виконане сканування.

У межах одного циклу роботи користувач розміщує перед вебкамерою один товар і натискає кнопку сканування. Система отримує один кадр із відеопотоку та надсилає його на серверну частину. Якщо результат розпізнавання є підтвердженим, товар автоматично додається до віртуального чека. Якщо результат непідтверджений, порожній або помилковий, товар до чека не додається, а користувач отримує відповідне повідомлення.

Окремо в системі передбачено адміністративний режим. Він призначений для додавання нових об’єктів каталогу та формування еталонних зображень. Адміністратор може створити новий об’єкт, вибрати сторону або ракурс і зберегти кадр із вебкамери як еталонне зображення для подальшого використання під час розпізнавання.

Поточна реалізація системи передбачає можливість роботи як у локальному середовищі розробки, так і в хмарному середовищі. Клієнтська та серверна частини розгортаються як окремі вебзастосунки в Azure App Service. Це дозволяє демонструвати роботу системи через браузер без необхідності запуску всієї інфраструктури на локальному комп’ютері.

Вхідними даними системи є:

- зображення товару, отримане з вебкамери після натискання кнопки сканування;
- дані об’єктів каталогу: технічна мітка, назва, ціна, опис та службова ознака активності;
- еталонні зображення об’єктів, які використовуються для порівняння під час розпізнавання;
- дані, введені адміністратором під час додавання нового об’єкта або еталонного кадру;
- дії користувача в інтерфейсі:
  - запуск сканування;
  - повторне сканування;
  - перегляд результату;
  - перехід до демонстраційного екрана оплати;
  - завершення сценарію покупки.

Вихідними даними системи є:

- результат розпізнавання товару за отриманим зображенням;
- повідомлення про:
- успішне сканування;
- непідтверджене сканування;
- порожнє сканування;
- помилкове сканування.
- дані розпізнаного товару, що відображаються в інтерфейсі;
- оновлений віртуальний чек і загальна сума покупки;
- службовий запис про виконане сканування в інформаційній базі;
- оновлений набір еталонних зображень після дій адміністратора.

Умовно-постійна інформація включає:

- каталог об’єктів, доступних для розпізнавання;
- технічні мітки, назви, ціни та описи товарів;
- еталонні зображення об’єктів;

- метадані еталонних зображень і шляхи до відповідних файлів;
- налаштування клієнт-серверної взаємодії.

До функціональних вимог системи належать:

- відображення відеопотоку з вебкамери в інтерфейсі користувача;
- отримання одного кадру після натискання кнопки сканування;
- передавання кадру на серверну частину для обробки;
- розпізнавання одного товару в межах одного циклу сканування;
- порівняння отриманого кадру з еталонними зображеннями;
- відображення результату розпізнавання або повідомлення про невдале сканування;
- автоматичне додавання товару до віртуального чека у разі підтвердженого результату;
- можливість повторного сканування;
- відображення демонстраційного екрана оплати та екрана успішного завершення сценарію;
- додавання нових об’єктів каталогу в адміністративному режимі;
- формування та збереження еталонних зображень;
- збереження службової інформації про виконані сканування.

У межах даної роботи не реалізується інтеграція з реальною платіжною системою. Екрани оплати використовуються як частина демонстраційного сценарію роботи каси самообслуговування.

До нефункціональних вимог системи належать:

- зручність і зрозумілість інтерфейсу для користувача;
- мінімальна кількість дій для запуску основного сценарію сканування;
- коректна обробка помилок доступу до камери, відсутності об’єкта або невдалого розпізнавання;
- можливість роботи клієнтської частини у сучасному веббраузері;
- відокремлення клієнтської та серверної частин для спрощення підтримки й подальшого розвитку;

- можливість локального запуску під час розробки та хмарного розгортання для демонстрації;
- захист адміністративного режиму від несанкціонованого доступу;
- можливість подальшого розширення каталогу об’єктів та еталонних зображень.

Система повинна працювати в контрольованих умовах: один об’єкт у кадрі, достатнє освітлення, стабільне положення камери та наявність якісних еталонних зображень. Розроблене програмне забезпечення не замінює повноцінну промислову касу самообслуговування, однак демонструє практичну можливість використання комп’ютерного зору, клієнт-серверної архітектури та хмарного розгортання для автоматизації ідентифікації товару за зображенням.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

Інформаційне забезпечення є однією з ключових складових програмного забезпечення інтелектуальної каси самообслуговування, оскільки воно визначає, які дані зберігаються в системі, як вони пов’язані між собою та яким чином використовуються під час роботи програмного продукту. Для розробленої системи інформаційне забезпечення охоплює каталог об’єктів, еталонні зображення для розпізнавання, журнал виконаних сканувань і результати роботи модуля розпізнавання.

На відміну від звичайної касової системи, яка переважно працює з товарним довідником і чеком, інтелектуальна каса самообслуговування додатково потребує даних, пов’язаних з ідентифікацією товару за зображенням. У розробленій системі це реалізовано через зв’язок між об’єктами каталогу та їх еталонними зображеннями. Під час сканування отриманий кадр порівнюється з еталонними даними, після чого система формує результат розпізнавання для відображення в інтерфейсі.

Інформаційна база також використовується для збереження службової історії розпізнавань. Це дає змогу фіксувати окремі спроби сканування, зберігати повідомлення про результат і технічні дані, необхідні для подальшого аналізу роботи системи. Такий підхід забезпечує не лише виконання поточного сценарію розпізнавання, а й створює основу для подальшого вдосконалення програмного продукту.

2.1 Логічна модель даних у вигляді ER-діаграми

Логічна модель даних є важливим етапом проєктування інформаційного забезпечення програмної системи, оскільки вона дозволяє визначити основні сутності, їхні атрибути та зв’язки між ними [8]. Для інтелектуальної каси самообслуговування така модель має відображати не лише каталог об’єктів, доступних для розпізнавання, а й еталонні зображення, які використовуються під час ідентифікації товару, а також службову історію виконаних сканувань.

У розробленій системі база даних використовується для підтримки трьох основних процесів: ведення каталогу об’єктів, зберігання відомостей про еталонні зображення та фіксації результатів розпізнавання. Поточна реалізація орієнтована на розпізнавання книг, однак структура даних побудована узагальнено, тому сутність objects може представляти й інші товари або об’єкти, які в майбутньому можуть бути додані до системи.

У межах логічної моделі визначено чотири основні сутності: objects - каталог об’єктів, доступних для розпізнавання; reference_images - еталонні зображення, пов’язані з об’єктами каталогу; recognition_runs - журнал запусків розпізнавання; recognition_run_items - підтверджені результати розпізнавання в межах конкретного запуску. Було побудовано ER-діаграму (додаток Б)

Сутність objects є центральною сутністю каталогу. Вона зберігає інформацію про об’єкти, які можуть бути розпізнані системою та відображені в інтерфейсі користувача. До її основних атрибутів належать унікальний ідентифікатор, технічна мітка, назва, ціна, опис, джерело запису, ознака активності та службові часові поля. Технічна мітка використовується для внутрішнього зіставлення результату розпізнавання з відповідним об’єктом каталогу, а назва й ціна потрібні для відображення товару в інтерфейсі.

Сутність reference_images призначена для зберігання відомостей про еталонні зображення. Кожне еталонне зображення пов’язане з конкретним об’єктом каталогу та використовується під час порівняння з кадром, отриманим у процесі сканування. У базі даних зберігаються метадані еталонного зображення та шлях до відповідного файлу, тоді як самі графічні файли розміщуються у файловій структурі серверної частини. Детальніше це рішення розглядається в наступних підрозділах, присвячених структурі інформаційної бази та фізичній реалізації.

Між сутностями objects і reference_images існує зв’язок типу один до багатьох. Один об’єкт каталогу може мати кілька еталонних зображень, наприклад для різних сторін або ракурсів, а кожне еталонне зображення

належить лише одному об’єкту каталогу. Такий зв’язок дозволяє розширювати набір еталонів для одного товару без дублювання інформації про сам об’єкт.

Сутність recognition_runs описує окремий запуск процесу розпізнавання. Вона фіксує факт виконання сканування, загальні відомості про отримане зображення, службове повідомлення про результат і дані, які характеризують виконання алгоритму. Ця сутність потрібна для ведення журналу спроб розпізнавання та подальшого аналізу роботи системи.

Сутність recognition_run_items призначена для зберігання підтверджених результатів розпізнавання в межах окремого запуску. У поточному сценарії система орієнтована на розпізнавання одного товару за один цикл сканування, однак винесення результатів в окрему сутність дозволяє відокремити загальний запис про запуск від конкретного розпізнаного об’єкта. До основних атрибутів цієї сутності належать посилання на запуск розпізнавання, посилання на об’єкт каталогу, технічна мітка результату, кількість, рівень упевненості та службові показники якості збігу.

Між сутностями recognition_runs і recognition_run_items існує зв’язок типу один до багатьох. Один запуск розпізнавання може мати пов’язані з ним результати, а кожен результат належить одному запуску. Також recognition_run_items пов’язана із сутністю objects, оскільки підтверджений результат може відповідати конкретному об’єкту каталогу. На логічному рівні цей зв’язок відображає зіставлення результату роботи модуля розпізнавання з товаром, який зберігається в каталозі.

Основні зв’язки логічної моделі можна описати так: один об’єкт каталогу може мати багато еталонних зображень; кожне еталонне зображення належить одному об’єкту; один запуск розпізнавання може містити результати розпізнавання; кожен результат належить одному запуску; один об’єкт каталогу може бути пов’язаний з багатьма результатами розпізнавання.

Побудована логічна модель відповідає реляційному підходу, оскільки дані поділені на окремі сутності, між якими встановлено зв’язки за допомогою ключових полів. Дані про об’єкти каталогу відокремлені від еталонних

зображень, а інформація про запуск розпізнавання відокремлена від конкретного підтвердженого результату. Це дозволяє уникнути зайвого дублювання даних, спростити підтримку каталогу та забезпечити можливість подальшого розширення системи.

Для перевірки логічної цілісності модель було розглянуто з погляду основних нормальних форм [9].

Перша нормальна форма виконується, оскільки атрибути таблиць є атомарними та не містять повторюваних груп. Наприклад, еталонні зображення не зберігаються списком у сутності objects, а винесені в окрему сутність reference_images.

Друга нормальна форма також виконується, оскільки кожна сутність має власний ідентифікатор, а її неключові атрибути описують саме той об’єкт, якому належить запис. Атрибути objects описують об’єкт каталогу, атрибути reference_images - конкретне еталонне зображення, атрибути recognition_runs - окремий запуск розпізнавання, а атрибути recognition_run_items - конкретний результат у межах запуску.

Третя нормальна форма загалом дотримується, оскільки довідкові дані про об’єкти зберігаються окремо від еталонних зображень і журналу розпізнавань. Поле з технічною міткою результату в recognition_run_items використовується для збереження службової історії роботи алгоритму та не замінює зв’язок із сутністю objects. Такий підхід дозволяє зберігати результат розпізнавання навіть у разі подальшої зміни даних каталогу.

Таким чином, логічна модель даних інтелектуальної каси самообслуговування є реляційною, структурованою та придатною для реалізації в SQLite. Вона охоплює основні інформаційні об’єкти системи: каталог об’єктів, еталонні зображення, запуски розпізнавання та підтверджені результати. Така структура забезпечує основу для адміністративного додавання еталонних зображень, виконання розпізнавання товарів, збереження історії сканувань і подальшого розвитку програмного забезпечення.

2.2 Вибір системи управління базою даних

Після побудови логічної моделі даних необхідно обрати систему управління базою даних, яка забезпечить зберігання каталогу об’єктів, метаданих еталонних зображень, журналу сканувань і результатів розпізнавання. Для розробленої системи важливо, щоб база даних була простою в розгортанні, не вимагала складного адміністрування, підтримувала реляційну структуру даних і могла працювати разом із серверною частиною вебзастосунку.

Оскільки логічна модель даних системи має реляційну структуру, для її реалізації доцільно використовувати реляційну систему управління базою даних. У розробленій системі дані поділено на окремі таблиці, між якими встановлено зв’язки за допомогою первинних і зовнішніх ключів. Такий підхід дозволяє забезпечити цілісність даних, уникнути надлишкового дублювання інформації та зручно працювати з каталогом об’єктів, еталонними даними й журналом розпізнавань.

У процесі вибору СУБД було розглянуто кілька можливих варіантів: SQLite, PostgreSQL, MySQL та Microsoft SQL Server. Кожна з цих систем може бути використана для зберігання структурованих даних, однак вони відрізняються складністю розгортання, вимогами до адміністрування, масштабованістю та доцільністю використання в межах конкретного проєкту.

SQLite є вбудованою реляційною СУБД, яка зберігає базу даних у вигляді окремого файлу [10]. Вона не потребує окремого серверного процесу, складної інсталяції або налаштування служби бази даних. Це робить її зручною для прототипів, навчальних проєктів і систем, де немає потреби в одночасній роботі великої кількості користувачів із високим навантаженням. У розробленій системі SQLite використовується для зберігання каталогу об’єктів, метаданих еталонних зображень, запусків розпізнавання та підтверджених результатів роботи алгоритму.

PostgreSQL і MySQL є повноцінними клієнт-серверними СУБД [11; 12]. Вони краще підходять для великих вебсистем, де потрібна підтримка значної кількості одночасних підключень, складні механізми розмежування доступу,

реплікація, резервне копіювання на рівні сервера та масштабування. Такі СУБД є доцільними для промислових систем, однак для поточного програмного забезпечення їх використання було б надмірним, оскільки база даних має невелику структуру та обслуговує контрольований сценарій розпізнавання одного об’єкта.

Microsoft SQL Server також є клієнт-серверною СУБД, яка забезпечує розвинені механізми безпеки, адміністрування, резервного копіювання та інтеграції з корпоративними системами [13]. Проте для даного проєкту її використання не є необхідним, оскільки система не потребує складної корпоративної інфраструктури баз даних, ролей користувачів, збережених процедур або окремого серверного адміністрування.

Для порівняння можливих варіантів СУБД наведено табл. 2.2.1.

Таблиця 2.1

Порівняльна характеристика можливих СУБД для системи інтелектуальної каси самообслуговування

Параметр	**SQLite**	**PostgreSQL**	**MySQL**	**Microsoft SQL Server**
Тип СУБД	Вбудована файлова реляційна СУБД	Клієнт-серверна реляційна СУБД	Клієнт-серверна реляційна СУБД	Клієнт-серверна реляційна СУБД
Окремий сервер БД	Не потрібен	Потрібен	Потрібен	Потрібен
Складність розгортання	Низька	Середня	Середня	Висока
Доцільність для невеликого каталогу	Висока	Надмірна	Надмірна	Надмірна
Підтримка реляційної структури	Є	Є	Є	Є
Зручність для дипломного прототипу	Висока	Середня	Середня	Низька
Масштабованість	Обмежена	Висока	Висока	Висока

З огляду на порівняння, для розробленої системи було обрано SQLite. Основною причиною такого вибору є відповідність SQLite фактичним потребам проєкту. Інтелектуальна каса самообслуговування в межах бакалаврської роботи працює з обмеженим каталогом об’єктів, еталонними даними та журналом

сканувань, тому немає потреби розгортати окремий сервер бази даних або впроваджувати складну систему адміністрування.

Важливою перевагою SQLite є простота інтеграції із серверною частиною, реалізованою мовою Python. База даних зберігається у файлі self_checkout.db, а її ініціалізація виконується серверною частиною під час запуску застосунку. Якщо таблиці ще не створені, система створює їх на основі SQL-схеми. Також під час запуску може виконуватися наповнення початкового каталогу об’єктів із seed-файлу, що спрощує підготовку системи до роботи.

У поточній реалізації база даних виконує практичну роль, а не є лише формальним елементом проєкту. Вона зберігає каталог об’єктів, прив’язку еталонних зображень до об’єктів, журнал запусків сканування та підтверджені результати роботи алгоритму розпізнавання. Детальна структура таблиць і фізична реалізація бази даних розглядаються в наступних підрозділах.

Окремо слід зазначити, що SQLite використовується лише для зберігання структурованих даних і метаданих. Еталонні зображення не зберігаються в базі даних як BLOB-об’єкти. Вони розміщуються у файловій структурі серверної частини, а база даних містить відомості про відповідний об’єкт, шлях до файлу та службові дані еталону. Такий підхід спрощує роботу модуля розпізнавання, не збільшує розмір файлу бази даних і полегшує адміністрування еталонних зображень.

SQLite є достатнім рішенням для поточного демонстраційного single-instance розгортання. Водночас для промислової експлуатації з великою кількістю користувачів, масштабуванням на кілька серверних екземплярів або складними вимогами до відмовостійкості доцільніше було б перейти на повноцінну клієнт-серверну СУБД, наприклад PostgreSQL. Однак для навчального проєкту, демонстраційного вебзастосунку та контрольованого сценарію розпізнавання одного об’єкта SQLite є достатнім і раціональним вибором.

Таким чином, SQLite було обрано як основну систему управління базою даних через її простоту, відповідність реляційній моделі, відсутність потреби в

окремому сервері БД, зручність інтеграції з Python/FastAPI та достатність для поточного обсягу даних. Такий вибір дозволяє зосередитися на основній задачі роботи – програмній ідентифікації об’єктів за зображенням у сценарії інтелектуальної каси самообслуговування – без надмірного ускладнення інфраструктури бази даних.

2.3 Розробка структури інформаційної бази

У процесі створення програмного забезпечення інтелектуальної каси самообслуговування було розроблено структуру інформаційної бази, яка забезпечує зберігання та зв’язування даних, необхідних для роботи системи. Інформаційна база використовується для ведення каталогу об’єктів, зберігання метаданих еталонних зображень, фіксації запусків сканування та збереження підтверджених результатів розпізнавання.

У поточній реалізації система працює з каталогом книг, однак структура інформаційної бази побудована узагальнено. Це означає, що в майбутньому до системи можуть бути додані інші типи товарів або об’єктів без повної зміни структури бази даних. Такий підхід відповідає задачі створення програмного забезпечення, яке може розширювати набір об’єктів для розпізнавання.

Інформаційна база реалізована у вигляді SQLite-бази даних self_checkout.db. Її схема описана у файлі schema.sql, а створення та ініціалізація виконуються серверною частиною застосунку під час запуску. У базі даних передбачено чотири основні таблиці: objects, reference_images, recognition_runs і recognition_run_items. Логічне призначення цих таблиць було розглянуто в підрозділі 2.1, а їхня фізична структура з типами даних, ключами та обмеженнями подається в підрозділі 2.4.

Таблиця objects використовується як каталог об’єктів, доступних для розпізнавання. У ній зберігаються технічна мітка об’єкта, назва, ціна, опис, джерело появи запису в каталозі та службові поля. У поточному стані каталог містить книги, але назва таблиці й структура полів не обмежують систему лише цим типом об’єктів. Поле catalog_source дає змогу відрізняти записи, додані з

початкового seed-файлу, від записів, створених через адміністративний інтерфейс. Це важливо, оскільки об’єкти, додані адміністратором, не повинні втрачатися під час оновлення початкового каталогу.

Для зберігання відомостей про еталонні кадри використовується таблиця reference_images. Вона містить зв’язок еталонного зображення з відповідним об’єктом каталогу, шлях до файлу зображення, початкову назву файлу та службові ознаки активності. Самі графічні файли не зберігаються в SQLite як BLOB-об’єкти. Вони розміщуються у файловій структурі серверної частини, а база даних зберігає лише метадані та відносні шляхи до файлів. Такий підхід спрощує роботу з еталонними кадрами, не збільшує розмір файлу бази даних і краще відповідає способу обробки зображень модулем розпізнавання.

Інформаційна база також підтримує збереження історії сканувань. Для цього використовується таблиця recognition_runs, у якій фіксується кожна спроба розпізнавання після натискання користувачем кнопки сканування. Один запис у цій таблиці відповідає одному запуску розпізнавання та містить загальні службові відомості про отримане зображення, повідомлення про результат і параметри виконання алгоритму.

Підтверджені результати розпізнавання зберігаються в таблиці recognition_run_items. Вона пов’язує конкретний запуск розпізнавання з об’єктом каталогу, який було визначено системою. У поточному сценарії система орієнтована на розпізнавання одного товару за один цикл сканування, однак винесення результатів в окрему таблицю робить структуру гнучкішою та дозволяє відокремити сам факт запуску сканування від конкретного розпізнаного результату.

Сумнівні або непідтверджені кандидати не зберігаються в таблиці recognition_run_items як окремі підтверджені записи. Інформація про такі випадки відображається через загальні поля запуску розпізнавання, зокрема повідомлення про результат і кількість непідтверджених позицій. Завдяки цьому таблиця результатів не перевантажується непевними значеннями, але система

все одно зберігає службову інформацію про невдалі або недостатньо впевнені сканування.

Створення інформаційної бази виконується автоматично під час запуску серверної частини. Для цього використовується механізм ініціалізації, який створює необхідні каталоги, відкриває підключення до SQLite, активує підтримку зовнішніх ключів і виконує SQL-схему створення таблиць. Якщо база даних ще не створена, вона формується на основі schema.sql, що спрощує локальний запуск і хмарне розгортання застосунку.

Початкове наповнення каталогу об’єктів виконується на основі seed-даних. Seed-файл містить базові записи каталогу, необхідні для першого запуску системи. Під час ініціалізації ці записи додаються до таблиці objects або оновлюються, якщо запис із відповідною технічною міткою вже існує. Якщо певний об’єкт прибрано з seed-даних, він може бути деактивований, а не видалений фізично, що дозволяє зберігати цілісність пов’язаних історичних даних.

Окремо передбачено адміністративний сценарій розширення інформаційної бази. Через адміністративний екран користувач з відповідним доступом може додати новий об’єкт каталогу та сформувати для нього еталонні зображення. У такому випадку в базі даних створюється запис про об’єкт, а еталонний кадр зберігається у файловій структурі серверної частини з подальшим додаванням відповідних метаданих до таблиці reference_images.

Запис результатів сканування виконується після обробки кадру серверною частиною. Спочатку створюється запис про сам запуск розпізнавання, а за наявності підтвердженого результату додається відповідний запис про розпізнаний об’єкт. Такий поділ дозволяє окремо зберігати факт виконання сканування та конкретний результат роботи алгоритму.

Таким чином, структура інформаційної бази розробленої системи забезпечує підтримку основних сценаріїв ідентифікації об’єктів за зображенням. Вона охоплює каталог об’єктів, метадані еталонних зображень, службову історію запусків розпізнавання та підтверджені результати. Крім того, структура

враховує адміністративне додавання нових об’єктів і еталонних кадрів, що створює основу для подальшого розширення системи.

2.4 Схема та фізична структура бази даних

Фізична структура бази даних визначає, як логічна модель даних реалізована на рівні конкретної системи управління базою даних. У межах розробленого програмного забезпечення інтелектуальної каси самообслуговування фізична структура реалізована у SQLite-базі даних self_checkout.db. Схема бази даних описана у файлі schema.sql, який містить SQL-команди створення таблиць, зовнішніх ключів, обмежень цілісності та індексів.

На фізичному рівні база даних складається з чотирьох основних таблиць: objects, reference_images, recognition_runs та recognition_run_items. Така структура відповідає логічній моделі, розглянутій у підрозділі 2.1, і забезпечує зберігання каталогу об’єктів, метаданих еталонних зображень, запусків розпізнавання та підтверджених результатів роботи алгоритму. Було побудовану схему фізичної структури бази даних (додаток В).

На відміну від логічної ER-діаграми, фізична схема відображає не лише сутності та зв’язки між ними, а й конкретні типи даних, первинні та зовнішні ключі, обмеження NOT NULL, UNIQUE, DEFAULT, CHECK, а також правила обробки пов’язаних записів. Саме тому фізична схема є ближчою до реальної SQL-структури, яка використовується в програмному коді.

Фізична структура бази даних реалізована з урахуванням особливостей SQLite. Для збереження текстових значень використовується тип TEXT, для цілих чисел - INTEGER, а для числових значень із дробовою частиною - REAL [14]. Логічні значення, такі як активність запису або факт наявності підтвердженого результату, зберігаються у вигляді цілих чисел 0 або 1. Дати й час зберігаються як текстові ISO-рядки, що є зручним рішенням для SQLite та подальшої обробки таких значень у серверній частині.

Таблиця objects є основною таблицею каталогу. Вона містить записи про об’єкти, які можуть бути розпізнані системою та відображені в інтерфейсі користувача. У поточній реалізації такими об’єктами є книги, однак структура таблиці не обмежує систему лише цим типом товарів. Для таблиці визначено первинний ключ id, унікальну технічну мітку label, обов’язкові поля назви й ціни, службове поле джерела запису catalog_source, ознаку активності та часові поля створення й оновлення. Для ціни використовується поле price_minor, яке зберігає значення в копійках, що дозволяє уникнути похибок під час роботи з грошовими значеннями.

Таблиця reference_images призначена для збереження метаданих еталонних зображень. Кожен запис цієї таблиці пов’язаний із конкретним об’єктом каталогу через зовнішній ключ object_id. У таблиці зберігається відносний шлях до файлу еталонного зображення, початкова назва файлу, ознака активності та дата створення запису. Самі графічні файли не зберігаються в SQLite як BLOB-об’єкти, а розміщуються у файловій структурі серверної частини. У базі даних зберігаються лише метадані та шлях до відповідного файлу.

Для зв’язку reference_images.object_id з objects.id використовується правило ON DELETE CASCADE. Це означає, що в разі видалення об’єкта з каталогу пов’язані з ним записи про еталонні зображення також видаляються з бази даних. Така поведінка є логічною, оскільки еталонне зображення не має сенсу без об’єкта, якому воно належить.

Таблиця recognition_runs використовується для збереження інформації про кожний запуск розпізнавання. Один запис у цій таблиці відповідає одній спробі сканування після натискання користувачем кнопки. Таблиця містить службові дані про отримане зображення, повідомлення про результат, назву та версію алгоритму, ознаку наявності підтвердженого результату й кількість непідтверджених позицій. Для логічного поля detected_any задано обмеження на значення 0 або 1, а для unresolved_count - обмеження, яке не дозволяє зберігати від’ємні значення.

Таблиця recognition_run_items зберігає підтверджені результати розпізнавання в межах конкретного запуску. Вона пов’язана з таблицею recognition_runs через поле run_id, а з таблицею objects - через поле object_id. У таблиці також зберігається технічна мітка, повернута алгоритмом, кількість, рівень упевненості та службові показники якості збігу. Для поля quantity використовується обмеження CHECK (quantity > 0), а для confidence - обмеження, яке дозволяє зберігати значення лише в межах від 0 до 1.

Для зв’язку recognition_run_items.run_id з recognition_runs.id використовується правило ON DELETE CASCADE. Це забезпечує видалення результатів розпізнавання разом із записом про відповідний запуск. Для зв’язку recognition_run_items.object_id з objects.id використовується правило ON DELETE SET NULL. Завдяки цьому історія розпізнавань може зберігатися навіть у випадку видалення або зміни об’єкта каталогу, оскільки посилання на об’єкт стає порожнім, але сам запис про результат не видаляється.

У фізичній схемі також передбачено індекси для полів, які часто використовуються під час пошуку, сортування або зв’язування таблиць. Індекс для objects.label прискорює пошук об’єкта за технічною міткою. Індекс для reference_images.object_id використовується для швидкого отримання еталонних зображень певного об’єкта. Індекс для recognition_runs.created_at допомагає працювати з історією запусків розпізнавання за датою створення. Індекси для recognition_run_items.run_id та recognition_run_items.object_id забезпечують швидкий доступ до результатів конкретного запуску або результатів, пов’язаних із певним об’єктом.

Під час ініціалізації бази даних серверна частина відкриває підключення до SQLite та виконує SQL-схему створення таблиць. У коді також активується підтримка зовнішніх ключів за допомогою PRAGMA foreign_keys = ON [15], що є необхідним для коректної роботи правил ON DELETE CASCADE і ON DELETE SET NULL. Лістинг SQL-схеми, що використовується для створення основних таблиць бази даних, наведено в додатку Д. У ньому подано створення

таблиць objects, reference_images, recognition_runs і recognition_run_items, а також індекси, необхідні для прискорення пошуку та зв’язування даних.

Отже, фізична структура бази даних відповідає поточним потребам розробленого програмного забезпечення. Вона забезпечує зберігання каталогу об’єктів, зв’язок об’єктів з еталонними зображеннями, журналювання запусків розпізнавання та збереження підтверджених результатів алгоритму. Використання первинних і зовнішніх ключів, обмежень цілісності, індексів і правил обробки пов’язаних записів забезпечує цілісність даних і зручність подальшої роботи з інформаційною базою.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі розглядається програмна реалізація системи ідентифікації об’єктів за зображенням у сценарії інтелектуальної каси самообслуговування. Якщо в попередніх розділах основну увагу було приділено предметній області, постановці задачі та проєктуванню інформаційного забезпечення, то в цьому розділі розглядаються структура програмного забезпечення, взаємодія його частин, організація frontend- і backend-модулів, вибір інструментальних засобів та алгоритми роботи системи.

Розроблене програмне забезпечення реалізовано як веборієнтовану клієнт-серверну систему. Клієнтська частина відповідає за взаємодію з користувачем, роботу з вебкамерою, відображення стану сканування, формування віртуального чека та перехід між екранами користувацького сценарію. Серверна частина приймає зображення, виконує його обробку, звертається до еталонних зображень і каталогу об’єктів, формує результат розпізнавання та зберігає службову інформацію про виконане сканування.

Основний сценарій системи передбачає обробку одного кадру, отриманого з вебкамери після дії користувача. У разі підтвердженого результату розпізнаний об’єкт автоматично додається до віртуального чека, а у випадку невдалого або непевного розпізнавання користувач отримує відповідне повідомлення. Детальніше структура модулів, взаємодія компонентів і алгоритм роботи системи розглядаються в наступних підрозділах.

3.1 Абстракції предметної області

Під час розробки програмного забезпечення важливо визначити основні абстракції предметної області, оскільки вони використовуються для подальшого моделювання структури програми, взаємодії об’єктів і побудови програмних модулів. Абстракції дозволяють відокремити ключові поняття системи від деталей конкретної реалізації та сформувати основу для подальшого проєктування.

У межах даної роботи предметною областю є ідентифікація об’єктів за зображенням у сценарії інтелектуальної каси самообслуговування. Основними поняттями цієї області є покупець, адміністратор, об’єкт каталогу, еталонне зображення, кадр сканування, запуск розпізнавання, результат розпізнавання та віртуальний чек. Ці поняття відображають як основний користувацький сценарій, так і адміністративне додавання нових об’єктів і еталонних зображень.

У системі можна виділити такі основні абстракції.

Покупець - користувач, який взаємодіє з основним інтерфейсом каси самообслуговування. Він ініціює сканування, переглядає результат розпізнавання та бачить поточний стан віртуального чека. У разі підтвердженого результату система автоматично додає відповідну позицію до чека.

Адміністратор - користувач, який працює з адміністративним екраном для додавання нових об’єктів каталогу та формування еталонних зображень. Адміністратор може створити новий запис у каталозі, вибрати сторону або ракурс об’єкта та зберегти кадр із вебкамери як еталонне зображення.

Об’єкт каталогу - фізичний об’єкт, який може бути розпізнаний системою. У поточній реалізації такими об’єктами є книги, однак структура системи дає змогу надалі розширювати каталог. Для об’єкта використовуються технічна мітка, назва, ціна та опис.

Еталонне зображення - зразок зображення об’єкта, який використовується для порівняння з кадром, отриманим під час сканування. Еталонні зображення пов’язані з об’єктами каталогу та можуть відповідати різним сторонам або ракурсам об’єкта.

Кадр сканування - зображення, отримане з вебкамери в момент запуску сканування. Кадр є вхідними даними для модуля розпізнавання. Система не аналізує відеопотік безперервно, а обробляє один кадр після дії користувача.

Запуск розпізнавання - окрема спроба сканування, яка виникає після натискання користувачем кнопки сканування. Ця абстракція відображає факт виконання одного циклу розпізнавання та пов’язується з отриманим кадром і результатом обробки.

Модуль розпізнавання – програмна частина серверного рівня, яка обробляє кадр сканування та порівнює його з еталонними зображеннями. У поточній реалізації використовується OpenCV ORB feature matching, однак на рівні абстракцій цей елемент розглядається як сервіс, що приймає кадр і формує результат розпізнавання.

Результат розпізнавання – підсумок роботи алгоритму в межах одного запуску. Результат може бути підтвердженим, непідтвердженим або порожнім. Підтверджений результат означає, що система змогла зіставити кадр із певним об’єктом каталогу; непідтверджений або порожній результат не призводить до додавання товару в чек.

Віртуальний чек – інтерфейсна структура, у якій відображаються розпізнані позиції та загальна сума покупки. У межах поточного проєкту чек є частиною frontend-сценарію та не розглядається як окрема таблиця бази даних.

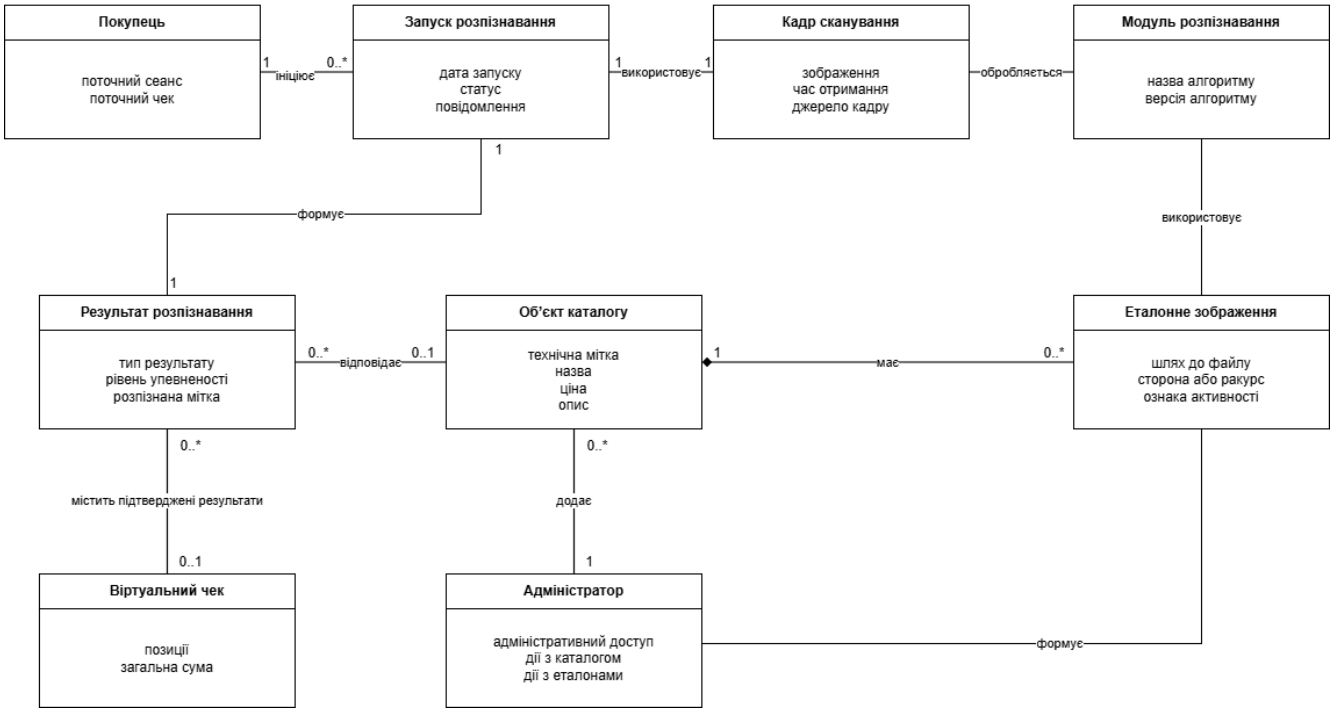


Рис. 3.1 Діаграма абстракцій предметної області системи ідентифікації об’єктів за зображенням

Основні зв’язки між абстракціями такі: покупець ініціює запуск розпізнавання; запуск використовує кадр сканування; кадр обробляється модулем розпізнавання; модуль використовує еталонні зображення; еталонні

зображення належать об’єктам каталогу; запуск формує результат розпізнавання; підтверджений результат відповідає об’єкту каталогу та може додавати позицію до віртуального чека. Адміністратор додає об’єкти каталогу та формує еталонні зображення для подальшого розпізнавання.

Виділення таких абстракцій дозволяє перейти до моделювання структури та взаємодії об’єктів програмного забезпечення. У наступному підрозділі ці поняття будуть уточнені через програмні компоненти frontend- і backend-частин, сервіси обробки зображень та структури даних, які використовуються під час роботи системи.

3.2 Моделювання структури та взаємодії об’єктів

Після визначення основних абстракцій предметної області необхідно перейти до їх формалізації у вигляді UML-моделей. Якщо в підрозділі 3.1 було виділено ключові поняття системи, то в цьому підрозділі ці поняття уточнюються через класи предметної області, їхні атрибути, операції та зв’язки між ними. Також розглядається взаємодія об’єктів у двох основних сценаріях: розпізнавання об’єкта покупцем і формування еталонного зображення адміністратором.

Для моделювання структури та взаємодії об’єктів використовується мова UML. У межах цього підрозділу доцільно використати діаграму класів і діаграми кооперації. Діаграма класів відображає статичну структуру предметної області, а діаграми кооперації показують, як об’єкти взаємодіють між собою під час виконання окремих сценаріїв роботи системи [3].

На відміну від ER-діаграми, яка була розглянута в розділі 2 і описує структуру бази даних, діаграма класів не деталізує таблиці, первинні ключі, зовнішні ключі або фізичні обмеження зберігання даних. Її призначення полягає в тому, щоб показати логічні об’єкти системи: покупця, адміністратора, запуск розпізнавання, кадр сканування, модуль розпізнавання, результат розпізнавання, об’єкт каталогу, еталонне зображення, віртуальний чек і позицію чека. Технічна

організація frontend- і backend-частин розглядається в наступних підрозділах, тому в цьому пункті вона не деталізується.

3.2.1 Діаграма класів. Діаграма класів використовується для відображення основних класів предметної області, їхніх властивостей, операцій і зв’язків. У цій роботі вона описує не окремі файли програмного коду, а логічну структуру системи ідентифікації об’єктів за зображенням у сценарії інтелектуальної каси самообслуговування. Було побудовано діаграму класів (додаток Е).

Клас Покупець відображає користувача, який працює з основним інтерфейсом каси самообслуговування. Він може запустити сканування та переглянути віртуальний чек. З покупцем пов’язаний клас Запуск розпізнавання, який описує окрему спробу сканування об’єкта. Один покупець може ініціювати багато запусків розпізнавання, а кожен запуск відповідає одному конкретному циклу сканування.

Клас Кадр сканування описує зображення, отримане в момент запуску сканування. Саме цей кадр є вхідними даними для подальшої обробки. У системі не виконується безперервний аналіз відеопотоку, тому в межах одного циклу роботи обробляється окремий кадр, отриманий після дії користувача. Кадр передається до модуля розпізнавання, який виконує його обробку та порівняння з еталонними зображеннями.

Клас Модуль розпізнавання описує логічну частину системи, відповідальну за визначення об’єкта на зображенні. На рівні діаграми класів він подається як об’єкт предметної області, що приймає кадр сканування, використовує еталонні зображення та формує результат розпізнавання. У програмній реалізації цьому класу відповідає серверна логіка обробки зображень, однак на діаграмі він не деталізується до конкретних файлів або модулів коду.

Клас Результат розпізнавання відображає підсумок виконаного сканування. Він містить тип результату, рівень упевненості та технічну мітку розпізнаного об’єкта. Результат може бути підтвердженим, непідтвердженим або

порожнім. Якщо результат підтверджений, він може відповідати певному об’єкту каталогу. Якщо система не змогла надійно визначити об’єкт, зв’язок із об’єктом каталогу може бути відсутнім.

Клас Об’єкт каталогу описує товар або інший об’єкт, який може бути розпізнаний системою. У поточній реалізації такими об’єктами є книги, однак модель побудована узагальнено, тому може бути розширена для інших типів товарів. Об’єкт каталогу має назву, ціну, опис, технічну мітку та ознаку активності. З ним пов’язаний клас Еталонне зображення, який описує зразок зображення об’єкта, що використовується під час розпізнавання. Один об’єкт каталогу може мати кілька еталонних зображень, наприклад для різних сторін або ракурсів.

Клас Віртуальний чек описує поточний чек користувача в межах сценарію самообслуговування. Він містить загальну суму та набір позицій. Клас Позиція чека описує окремий доданий до чека об’єкт, його кількість, ціну та суму позиції. Позиція чека посилається на об’єкт каталогу, оскільки кожна позиція відповідає конкретному розпізнаному товару. До чека додаються тільки ті позиції, для яких результат розпізнавання був підтверджений.

Клас Адміністратор відображає користувача, який виконує дії з підтримки каталогу та еталонних зображень. Адміністратор може додавати нові об’єкти каталогу та формувати для них еталонні зображення. Цей клас відповідає адміністративному сценарію роботи системи й відокремлений від основного сценарію покупця.

Основні зв’язки між класами відображають логіку роботи системи. Покупець ініціює запуск розпізнавання та переглядає віртуальний чек. Запуск розпізнавання використовує кадр сканування, який передається до модуля розпізнавання. Модуль розпізнавання обробляє кадр, використовує еталонні зображення та формує результат розпізнавання. Результат розпізнавання може відповідати об’єкту каталогу. Віртуальний чек містить позиції чека, а кожна позиція пов’язана з певним об’єктом каталогу. Адміністратор додає об’єкти каталогу та формує еталонні зображення для подальшого розпізнавання.

3.2.2 Діаграми кооперації. Діаграми кооперації використовуються для відображення взаємодії об’єктів у межах окремого сценарію роботи системи. Якщо діаграма класів показує статичну структуру предметної області, то діаграма кооперації пояснює, які об’єкти беруть участь у виконанні процесу та які дії пов’язують ці об’єкти між собою.

У межах розробленої системи доцільно розглянути дві кооперації. Перша описує основний користувацький сценарій розпізнавання об’єкта та додавання підтвердженого результату до віртуального чека. Друга описує адміністративний сценарій формування еталонного зображення, яке надалі використовується модулем розпізнавання. Було побудовано діаграму кооперацій (рис. 3.3). процесу розпізнавання об’єкта.

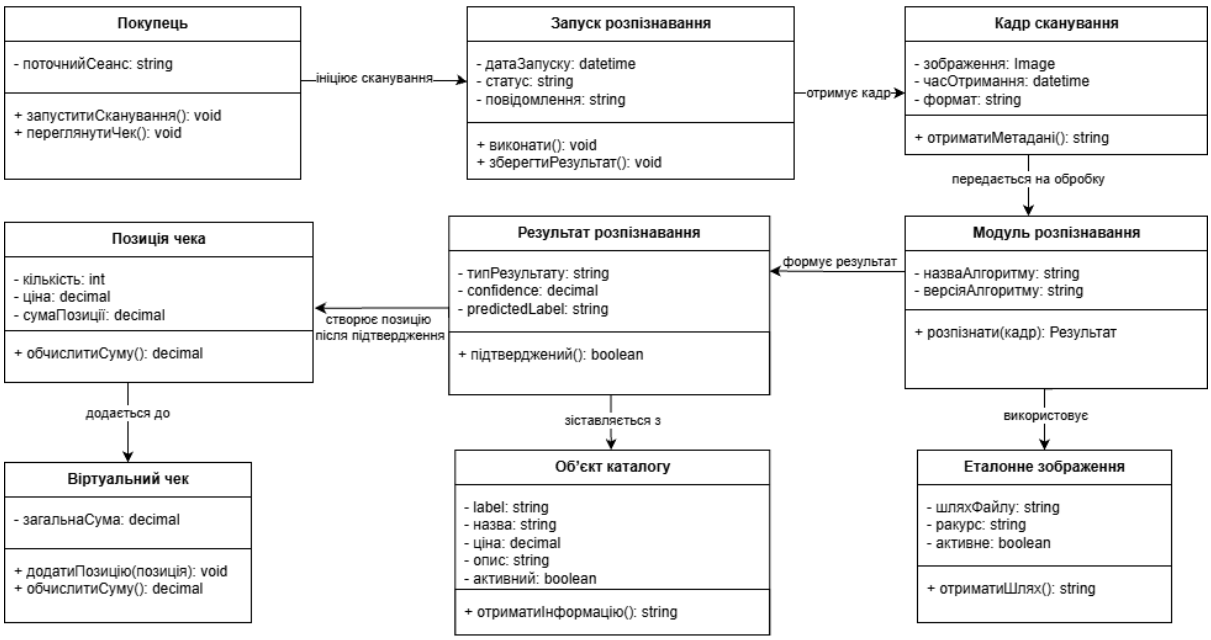


Рис. 3.3 Діаграма кооперації процесу розпізнавання об’єкта

У цьому сценарії покупець ініціює сканування, після чого створюється запуск розпізнавання. Запуск використовує кадр сканування, отриманий у момент дії користувача. Далі кадр передається до модуля розпізнавання, який використовує еталонні зображення для порівняння. На основі обробки кадру формується результат розпізнавання.

Якщо результат розпізнавання відповідає певному об’єкту каталогу і є підтвердженим, на його основі створюється позиція чека. Позиція додається до

віртуального чека, після чого покупець бачить оновлений стан покупки. Якщо результат непідтверджений або порожній, позиція до чека не додається, а користувач отримує відповідне повідомлення. Таким чином, діаграма кооперації показує не внутрішній програмний код, а взаємодію основних об’єктів предметної області під час виконання сценарію сканування. Було побудовано діаграму кооперацій (рис. 3.4) формування еталонного зображення.

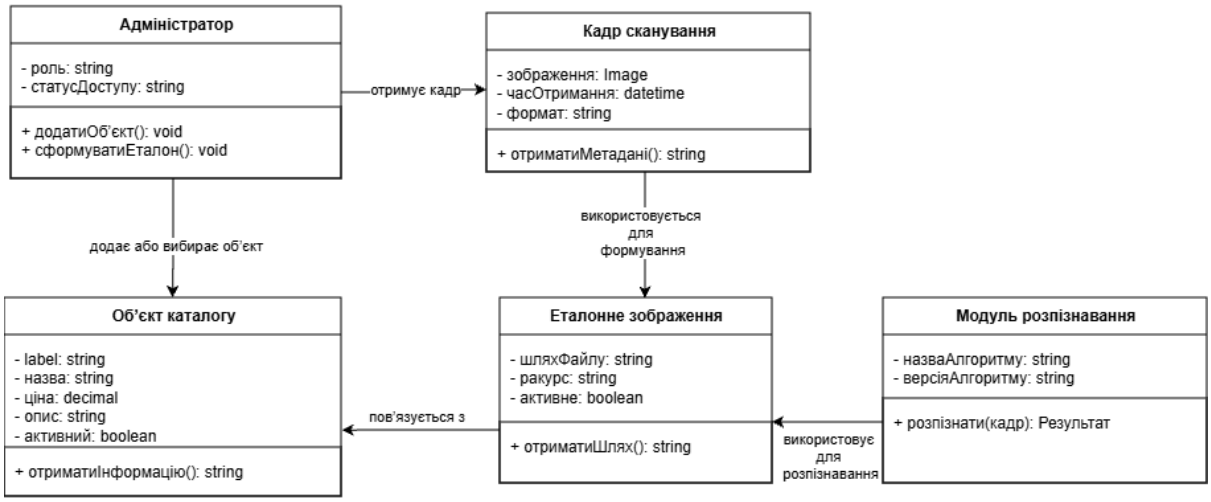


Рис. 3.4 Діаграма кооперації формування еталонного зображення

Адміністративний сценарій починається з дій адміністратора, який додає новий об’єкт каталогу або вибирає вже наявний об’єкт для формування еталону. Після цього отримується кадр сканування, задається сторона або ракурс об’єкта, а на основі кадру формується еталонне зображення. Еталонне зображення пов’язується з відповідним об’єктом каталогу та зберігається для подальшого використання під час розпізнавання.

Ця кооперація є важливою, оскільки без еталонних зображень система не зможе коректно зіставляти нові кадри з об’єктами каталогу. Адміністратор у цьому сценарії не бере участі в оформленні покупки, а забезпечує підготовку даних для майбутнього розпізнавання. Такий поділ дозволяє відокремити основний сценарій покупця від службового сценарію наповнення каталогу й еталонної бібліотеки.

Таким чином, моделювання структури та взаємодії об’єктів дозволяє уточнити логіку роботи системи на рівні предметної області. Діаграма класів

показує основні класи, їхні атрибути, операції та зв’язки, а діаграми кооперації пояснюють взаємодію цих класів у ключових сценаріях: розпізнаванні об’єкта та формуванні еталонного зображення. Це створює основу для подальшого опису організаційної та компонентної структури програмного забезпечення в наступних підрозділах.

3.3 Організаційна структура програмного забезпечення

Організаційна структура програмного забезпечення визначає, як вихідний код проєкту поділено на окремі частини та які функції виконує кожна з них. Для розробленої системи такий поділ є важливим, оскільки програмне забезпечення складається з клієнтської частини, серверної частини, модуля розпізнавання, шару доступу до даних, адміністративного інтерфейсу, файлової структури еталонних зображень і файлів розгортання.

Проєкт організовано як веборієнтовану клієнт-серверну систему. Клієнтська частина працює у браузері та відповідає за інтерфейс користувача, роботу з вебкамерою, запуск сканування, відображення результату розпізнавання та формування віртуального чека. Серверна частина приймає запити від клієнтської частини, виконує обробку зображення, працює з каталогом об’єктів і еталонними зображеннями, а також зберігає службову інформацію про виконані сканування.

Організаційну структуру програмного забезпечення доцільно розглядати через основні частини кодової бази: клієнтську частину, серверну частину, модулі розпізнавання, модулі доступу до даних, файлову структуру еталонних зображень, конфігураційні файли та файли розгортання. Такий поділ дозволяє відокремити інтерфейс користувача від серверної логіки, а логіку розпізнавання - від роботи з базою даних і службовими файлами.

Клієнтська частина проєкту розміщена в каталозі src. Вона реалізована як React-застосунок і відповідає за роботу користувацького інтерфейсу. Центральним файлом клієнтської частини є App.tsx, який координує основні режими роботи застосунку та перемикання між екранами. У поточній реалізації

не використовується окремий React Router; для вибору потрібного режиму застосовано простіший підхід на основі маршруту або hash-значення.

Клієнтська частина містить основні екрани користувацького сценарію: стартовий екран, екран сканування, демонстраційний екран оплати, екран успішного завершення сценарію та адміністративний екран /admin/reference-capture. Екрани оплати використовуються як частина демонстраційного сценарію каси самообслуговування без інтеграції з реальною платіжною системою. Основним екраном для виконання розпізнавання є екран сканування, на якому користувач працює з вебкамерою, запускає сканування та отримує результат.

До ключових частин клієнтської частини належать каталоги src/screens, src/components, src/services, src/utils, src/config і файл src/types.ts. Каталог screens містить основні екрани застосунку, зокрема ScanScreen і ReferenceCaptureScreen. Каталог components містить повторно використовувані інтерфейсні компоненти, серед яких важливим є CameraPreview, що відповідає за відображення відеопотоку з вебкамери. Каталог services використовується для організації API-запитів до серверної частини, а utils містить допоміжну логіку, зокрема отримання одного кадру з відеопотоку. Файл types.ts містить типи даних, які використовуються в клієнтській частині для опису відповіді розпізнавання, елементів чека та об’єктів каталогу.

Серверна частина проєкту розміщена в каталозі backend/app. Вона реалізована з використанням FastAPI та поділена на маршрути, core-модулі, модулі роботи з базою даних, репозиторії та схеми даних. Головним файлом серверної частини є backend/app/main.py, у якому створюється FastAPI-застосунок, налаштовується взаємодія з клієнтською частиною та підключаються маршрути розпізнавання й адміністративної бібліотеки еталонів.

Маршрути серверної частини розміщені в каталозі backend/app/routers. Маршрут predict.py відповідає за перевірку стану inference API та приймання кадру для розпізнавання. Через цей маршрут клієнтська частина передає отримане з вебкамери зображення на серверну частину. Маршрут

reference_library.py забезпечує роботу адміністративного сценарію: отримання списку об’єктів, додавання нового об’єкта каталогу та збереження еталонного кадру.

Основна логіка серверної частини зосереджена в каталозі backend/app/core. Модуль recognition_service.py координує процес розпізнавання: приймає зображення, отримує необхідні дані, звертається до модуля комп’ютерного зору та формує результат. Безпосереднє порівняння зображень виконується в модулі opencv_recognition.py на основі OpenCV ORB feature matching. Логіка додавання об’єктів каталогу та збереження еталонних кадрів реалізована в reference_library_service.py, а перевірка адміністративного доступу винесена в admin_auth.py.

Робота з інформаційним забезпеченням відокремлена від основної логіки розпізнавання. Каталог backend/app/db містить SQL-схему, модулі ініціалізації бази даних, seed-дані та синхронізацію файлової структури еталонних зображень. Детальна логічна й фізична структура бази даних була розглянута в розділі 2, тому в цьому підрозділі база даних розглядається лише як частина організації програмного коду.

Доступ до даних реалізовано через repository-модулі. Такий підхід дозволяє не змішувати SQL-операції з маршрутизацією API або логікою розпізнавання. У проєкті окремо виділено репозиторії для роботи з об’єктами каталогу, еталонними зображеннями та записами про запуски розпізнавання. Завдяки цьому серверні сервіси можуть звертатися до даних через окремий шар, не дублюючи SQL-логіку в різних частинах програми.

Еталонні зображення розміщуються у файловій структурі серверної частини, а база даних містить метадані та відносні шляхи до файлів. У межах організаційної структури це рішення означає, що графічні файли еталонів і структуровані дані про них зберігаються окремо. Такий підхід спрощує роботу модуля розпізнавання із зображеннями та не перевантажує SQLite-базу графічними даними.

Окреме місце в організаційній структурі займають конфігураційні файли та файли розгортання. Поточна версія проєкту підготовлена до хмарного розгортання в Microsoft Azure. Клієнтська та серверна частини розгортаються як два окремі вебзастосунки в Azure App Service [16]. Клієнтська частина збирається командою npm run build, після чого production-збірка з каталогу dist разом із server.mjs використовується для віддавання клієнтського застосунку. Серверна частина розгортається окремо як Python/FastAPI-застосунок із власними залежностями.

Для автоматизації розгортання використовуються GitHub Actions workflow-файли [17]. Вони відповідають за встановлення залежностей, збірку клієнтської частини, пакування серверної частини та розгортання відповідних частин у Azure App Service. Детальніше склад інсталяційного пакету, конфігураційні параметри та deployment-файли розглядаються в підрозділі 4.3.

Таким чином, організаційна структура програмного забезпечення є поділеною на зрозумілі функціональні частини. Клієнтська частина відповідає за інтерфейс і взаємодію з користувачем, серверна частина - за API, розпізнавання та адміністративні дії, repository-рівень - за доступ до даних, а файли розгортання - за підготовку й публікацію застосунку в хмарному середовищі. Такий поділ спрощує супровід проєкту, дозволяє окремо змінювати клієнтську частину, серверну частину або модуль розпізнавання та створює основу для подальшого розвитку системи.

3.4 Компонентна структура системи

Компонентна структура системи відображає поділ програмного забезпечення на логічні частини, які мають окремі зони відповідальності та взаємодіють між собою через визначені інтерфейси. Після опису загальної організаційної структури програмного забезпечення в підрозділі 3.3 доцільно розглянути, з яких компонентів складається система під час виконання основних сценаріїв роботи.

Розроблена система ідентифікації об’єктів за зображенням має клієнт-серверну компонентну структуру. Клієнтська частина працює у браузері та відповідає за взаємодію з користувачем, роботу з вебкамерою, отримання кадру, відображення результату розпізнавання та оновлення віртуального чека. Серверна частина приймає запити від клієнтської частини, виконує обробку зображення, запускає механізм розпізнавання, працює з еталонними зображеннями та зберігає службові результати сканувань. Такий поділ відповідає фактичній архітектурі системи, у якій клієнтська частина не виконує розпізнавання локально, а передає отриманий кадр на серверну частину для обробки. Було побудовано компонентку діаграму (додаток Ж).

На компонентній діаграмі систему поділено на три основні рівні: клієнтський рівень, серверний рівень і рівень зберігання даних. Клієнтський рівень охоплює вебінтерфейс користувача, компонент роботи з камерою, клієнтський API-сервіс, адміністративний екран еталонів і віртуальний чек. Серверний рівень містить серверний API, API розпізнавання, API еталонної бібліотеки, компонент розпізнавання, сервіс еталонної бібліотеки, перевірку адміністративного доступу, компонент доступу до даних і OpenCV-механізм розпізнавання. Рівень зберігання даних включає SQLite-базу даних і файлову структуру еталонних зображень.

Клієнтський рівень є частиною системи, з якою безпосередньо взаємодіє користувач. Вебінтерфейс користувача відповідає за відображення екрана сканування, повідомлень про стан розпізнавання, результату обробки та віртуального чека. Через цей компонент покупець запускає сканування, бачить результат і отримує оновлений стан чека після підтвердженого розпізнавання.

Компонент роботи з камерою забезпечує доступ до вебкамери засобами браузера, відображає відеопотік і дозволяє отримати один кадр після натискання кнопки сканування. Він не виконує розпізнавання самостійно, а лише формує вхідні дані для подальшої обробки серверною частиною. Це відповідає поточній реалізації системи, у якій не виконується безперервний аналіз відеопотоку, а обробляється окремий кадр, отриманий у момент дії користувача.

Клієнтський API-сервіс відповідає за передавання даних між клієнтською та серверною частинами. У користувацькому сценарії він передає кадр на серверну частину у форматі HTTP-запиту, а в адміністративному сценарії надсилає запити, пов’язані з додаванням об’єктів каталогу та збереженням еталонних зображень. Серверна частина повертає відповідь у структурованому вигляді, після чого клієнтська частина оновлює інтерфейс користувача.

Адміністративний екран еталонів є окремим компонентом клієнтського рівня. Він використовується не покупцем, а адміністратором для додавання нових об’єктів каталогу та формування еталонних зображень. Через цей екран адміністратор може вибрати або створити об’єкт, отримати кадр із вебкамери та зберегти його як еталон для подальшого розпізнавання.

Віртуальний чек є компонентом клієнтської частини, який відображає позиції, додані після підтвердженого розпізнавання, та загальну суму. Він не є окремою таблицею бази даних у поточній реалізації, а використовується як частина користувацького сценарію frontend-застосунку.

Серверний API є центральною точкою входу для запитів від клієнтської частини. Він приймає HTTP-запити, маршрутизує їх до відповідних API-компонентів і забезпечує зв’язок між frontend-застосунком та серверною логікою. На діаграмі окремо виділено API розпізнавання та API еталонної бібліотеки, оскільки вони відповідають за різні сценарії роботи системи.

API розпізнавання приймає кадр, отриманий із вебкамери, та передає його до компонента розпізнавання. Цей компонент відповідає за основний користувацький сценарій: приймання зображення, запуск обробки та повернення результату розпізнавання до клієнтської частини.

Компонент розпізнавання координує процес обробки кадру. Він отримує зображення, працює з каталогом, еталонами та журналом сканувань через компонент доступу до даних, а також передає кадр до OpenCV-механізму розпізнавання. Безпосереднє порівняння кадру з еталонними зображеннями виконується за допомогою OpenCV ORB feature matching. За результатами

обробки формується відповідь, яка може містити підтверджений, непідтверджений, порожній або помилковий результат.

OpenCV-механізм є спеціалізованим компонентом серверного рівня, який виконує порівняння отриманого кадру з еталонними зображеннями. Він використовує графічні файли еталонів, розміщені у файловій структурі серверної частини, і повертає службові показники розпізнавання, зокрема рівень упевненості та кількість знайдених збігів.

API еталонної бібліотеки забезпечує адміністративні запити, пов’язані з додаванням об’єктів каталогу та еталонних кадрів. Він передає такі запити до сервісу еталонної бібліотеки. Сервіс еталонної бібліотеки відповідає за додавання об’єктів, збереження кадрів як еталонних зображень і фіксацію відповідних метаданих. Для захисту адміністративних дій на серверному рівні передбачено компонент перевірки адміністративного доступу.

Компонент доступу до даних відокремлює роботу з інформаційною базою від маршрутизації API та логіки розпізнавання. Через нього система отримує дані каталогу, метадані еталонних зображень, шляхи до файлів еталонів і зберігає службову історію запусків розпізнавання. Такий компонентний поділ дозволяє не змішувати SQL-запити та роботу з файлами з логікою API або алгоритмом розпізнавання.

Рівень зберігання даних складається з SQLite-бази даних і файлової структури еталонних зображень. SQLite-база даних зберігає каталог об’єктів, метадані еталонів, службову історію запусків розпізнавання та підтверджені результати. Файлова структура еталонів містить графічні файли еталонних кадрів. У базі даних зберігаються лише метадані та шляхи до файлів, а не самі графічні файли. Такий підхід спрощує роботу OpenCV-механізму з еталонними зображеннями та не перевантажує SQLite-базу великими BLOB-даними.

Взаємодія компонентів у межах основного сценарію відбувається так: вебінтерфейс користувача використовує компонент роботи з камерою для отримання одного кадру, після чого клієнтський API-сервіс передає кадр на серверний API. Серверний API маршрутизує запит до API розпізнавання, який

передає зображення до компонента розпізнавання. Компонент розпізнавання звертається до компонента доступу до даних для роботи з каталогом, еталонами та журналом сканувань, а також використовує OpenCV-механізм для порівняння кадру з еталонними зображеннями. Після завершення обробки результат повертається до клієнтської частини. Якщо результат підтверджений, вебінтерфейс оновлює віртуальний чек.

Адміністративний сценарій використовує іншу гілку компонентної структури. Адміністративний екран еталонів через клієнтський API-сервіс надсилає запити до серверного API. Далі запит маршрутизується до API еталонної бібліотеки та сервісу еталонної бібліотеки. Перед виконанням адміністративних дій виконується перевірка адміністративного доступу. Після цього система може додати новий об’єкт каталогу або зберегти кадр як еталонне зображення. Файл еталону розміщується у файловій структурі, а його метадані зберігаються в SQLite-базі даних.

Особливістю компонентної структури є чітке відокремлення відповідальностей між частинами системи. Клієнтський рівень відповідає за інтерфейс, роботу з камерою та передачу запитів. Серверний рівень відповідає за маршрутизацію запитів, розпізнавання, адміністративні дії та доступ до даних. Рівень зберігання даних відповідає за збереження структурованих даних і графічних файлів еталонів. Такий поділ спрощує супровід програмного забезпечення, дозволяє окремо змінювати клієнтську частину, серверну логіку або механізм розпізнавання, а також створює основу для подальшого розширення системи.

Таким чином, компонентна структура системи відображає взаємодію основних програмних частин під час сканування об’єкта та формування еталонних зображень. Діаграма показує, що клієнтський вебзастосунок відповідає за отримання кадру й відображення результату, серверний рівень виконує обробку та розпізнавання, а рівень зберігання даних забезпечує роботу з каталогом, еталонами та службовою історією сканувань. Це створює основу для

подальшого опису вибору інструментарію та алгоритмізації програмних модулів.

3.5 Вибір інструментарію для створення програмного забезпечення

Вибір інструментарію для створення програмного забезпечення є важливим етапом розробки, оскільки від нього залежать швидкість реалізації системи, зручність підтримки коду, можливість інтеграції окремих компонентів і подальше розгортання програмного продукту. Для розроблення системи ідентифікації об’єктів за зображенням було обрано набір інструментів, який відповідає клієнт-серверній архітектурі, необхідності роботи з вебкамерою, обробки зображень, зберігання результатів і хмарного розгортання.

У попередніх підрозділах було розглянуто структуру та компоненти системи, тому в цьому пункті основна увага приділяється саме обґрунтуванню вибору технологій. Повторний опис структури каталогів, бази даних або повного сценарію сканування тут не наводиться.

Для клієнтської частини було обрано React у поєднанні з TypeScript [18; 19]. React є доцільним для побудови інтерфейсу з окремих компонентів, що відповідає структурі розробленого frontend-застосунку. У системі є кілька екранів і повторно використовуваних елементів інтерфейсу, зокрема екран сканування, компонент роботи з вебкамерою, елементи відображення результату, віртуальний чек та адміністративний екран. TypeScript використовується для типізації станів інтерфейсу, моделей відповіді серверної частини, об’єктів каталогу та елементів чека. Це зменшує ризик помилок під час обробки JSON-відповідей і спрощує супровід коду.

Для локальної розробки та production-збірки клієнтської частини використано Vite [20]. Цей інструмент забезпечує швидкий запуск застосунку під час розробки та формування готової збірки для розгортання. У локальному режимі Vite також використовується для проксіювання запитів до серверної частини, що спрощує розробку клієнт-серверної взаємодії.

Для стилізації інтерфейсу використано Tailwind CSS [21]. Його застосування дало змогу швидко створити простий і візуально зрозумілий kiosk-style інтерфейс без великої кількості окремих CSS-файлів. Це важливо для каси самообслуговування, оскільки користувацький сценарій має бути зрозумілим, а основні дії - помітними для користувача.

Серверна частина реалізована мовою Python із використанням фреймворку FastAPI [22]. Python було обрано через зручність роботи з бібліотеками комп'ютерного зору та обробки зображень. FastAPI використовується для створення HTTP API, яке приймає зображення від клієнтської частини, передає його до сервісу розпізнавання та повертає структуровану відповідь у форматі JSON. Такий підхід добре відповідає клієнт-серверній структурі системи.

Для обробки зображень і реалізації розпізнавання використано OpenCV [23]. У поточній реалізації застосовано підхід OpenCV ORB feature matching, тобто порівняння візуальних ознак кадру, отриманого з вебкамери, з еталонними зображеннями об'єктів [24; 25]. Важливо, що система не використовує нейромережеву модель, YOLO або окремий класифікатор із навчанням. Розпізнавання орієнтоване на контрольований сценарій: один об'єкт у кадрі, стабільне освітлення, якісні еталонні зображення та запуск сканування після дії користувача.

Для зберігання структурованих даних використовується SQLite. Детальне обґрунтування вибору цієї СУБД було наведено в підрозділі 2.2, тому в межах цього пункту достатньо зазначити її роль у складі інструментарію. SQLite використовується для зберігання каталогу об'єктів, метаданих еталонних зображень, запусків розпізнавання та підтверджених результатів. Еталонні зображення зберігаються у файловій структурі серверної частини, а база даних містить службову інформацію та шляхи до відповідних файлів.

Для обміну даними між клієнтською і серверною частинами використовується HTTP API. Клієнтська частина надсилає кадр на сервер у форматі multipart/form-data, а серверна частина повертає результат у форматі JSON. Такий формат взаємодії є зручним для вебзастосунку, оскільки дозволяє

передавати зображення та отримувати структуровану відповідь для подальшого відображення в інтерфейсі.

Для розгортання програмного забезпечення використано Azure App Service [16]. Поточна версія системи розгорнута в Microsoft Azure як два окремі вебзастосунки: окремо клієнтська частина та окремо серверна частина. Такий підхід відповідає клієнт-серверній архітектурі та дозволяє незалежно оновлювати frontend і backend. Клієнтська частина після збірки віддає вебінтерфейс, а серверна частина працює як Python/FastAPI API-сервіс.

Для автоматизації розгортання використано GitHub Actions [17]. Workflow-файли виконують встановлення залежностей, збірку клієнтської частини, підготовку серверного пакета та розгортання відповідних частин в Azure App Service. Для зберігання коду й контролю версій використовується GitHub, що забезпечує збереження історії змін і зв’язок із процесом автоматизованого розгортання.

Обраний набір інструментів відповідає практичним потребам розробленої системи. React, TypeScript, Vite і Tailwind CSS забезпечують створення зручного вебінтерфейсу; Python, FastAPI та OpenCV дають змогу реалізувати серверне API й обробку зображень; SQLite є достатнім для демонстраційного сценарію та зберігання обмеженого набору структурованих даних; Azure App Service і GitHub Actions забезпечують хмарне розгортання та автоматизацію оновлення програмного продукту.

Водночас обраний інструментарій має певні обмеження. Поточна реалізація OpenCV ORB найкраще працює в контрольованих умовах і не є повноцінною промисловою системою розпізнавання товарів у складному магазинному середовищі. SQLite і файлове зберігання еталонних зображень є прийнятними для демонстраційного single-instance розгортання, однак для масштабованої production-архітектури доцільно було б розглянути серверну СУБД і окреме хмарне сховище файлів. Ці обмеження не суперечать меті роботи, оскільки розроблена система демонструє практичну реалізацію ідентифікації

одного об’єкта за зображенням у сценарії інтелектуальної каси самообслуговування.

3.6 Алгоритмізація та програмування програмних модулів

Алгоритмізація програмних модулів є важливим етапом розробки, оскільки вона визначає послідовність виконання основних операцій системи та зв’язує між собою інтерфейс користувача, серверну обробку зображення, модуль розпізнавання і збереження службових результатів. У межах розробленого програмного забезпечення основними є два функціональні алгоритми: алгоритм розпізнавання об’єкта за кадром із вебкамери та алгоритм формування еталонних зображень в адміністративному режимі.

Перший алгоритм відповідає основному користувацькому сценарію. Він починається з дії покупця, який запускає сканування, і завершується відображенням результату в інтерфейсі. У разі підтвердженого результату розпізнаний товар автоматично додається до віртуального чека. Другий алгоритм використовується адміністратором для розширення каталогу об’єктів і створення еталонних кадрів, які надалі застосовуються під час розпізнавання.

У цьому підрозділі основна увага приділяється тому, як програмні модулі виконують свої дії під час роботи системи.

Основний алгоритм розпізнавання реалізує сценарій одноразового сканування. Система не аналізує відеопотік безперервно, а отримує один кадр після натискання кнопки сканування. Такий підхід зменшує навантаження на серверну частину та відповідає поточній реалізації проєкту, де клієнтська частина формує snapshot із вебкамери й надсилає його на серверну частину для обробки. Фрагмент програмного коду, що реалізує отримання одного кадру з відеопотоку вебкамери на клієнтській стороні, наведено в додатку К.

У додатку И наведено блок-схему алгоритму розпізнавання об’єкта за зображенням.

Алгоритм розпізнавання виконується в такій послідовності:

- користувач розміщує об’єкт перед вебкамерою та натискає кнопку сканування;
- клієнтська частина отримує поточний кадр із відеопотоку та перетворює його у зображення, придатне для передавання на сервер;
- зображення надсилається на серверну частину у форматі multipart/form-data;
- серверна частина перевіряє отриманий файл і готує його до обробки;
- модуль розпізнавання завантажує активні еталонні зображення, пов’язані з об’єктами каталогу;
- отриманий кадр порівнюється з еталонними зображеннями;
- для знайдених збігів обчислюються службові метрики, зокрема кількість якісних збігів, кількість внутрішніх відповідностей і рівень упевненості;
- система визначає стан результату: підтверджений, непідтверджений, порожній або помилковий;
- серверна частина формує відповідь для клієнтської частини та зберігає службову інформацію про запуск сканування;
- клієнтська частина оновлює інтерфейс і, у разі підтвердженого результату, автоматично додає товар до віртуального чека.

Фрагмент програмного коду, що відповідає за формування запиту multipart/form-data та надсилання кадру на серверну частину, наведено в додатку Л.

Програмно цей алгоритм розподілений між клієнтськими та серверними модулями. Клієнтська частина відповідає за відображення відеопотоку, отримання одного кадру, формування запиту та оновлення інтерфейсу після отримання відповіді. Розпізнавання не виконується в браузері. Основна логіка обробки розміщена на серверному рівні, що дозволяє централізовано використовувати OpenCV, каталог еталонів і журнал розпізнавань.

У поточній реалізації для розпізнавання використовується підхід OpenCV ORB feature matching. Він базується не на навчанні нейронної мережі, а на

знаходженні та порівнянні ключових ознак між вхідним кадром і еталонними зображеннями [24; 25]. Під час обробки кадр декодується, перетворюється у формат, зручний для аналізу, після чого для нього визначаються ключові точки та дескриптори. Такі самі ознаки отримуються для активних еталонних зображень, після чого виконується їх порівняння.

Фрагмент програмного коду, що реалізує порівняння отриманого кадру з еталонними зображеннями за допомогою OpenCV ORB feature matching, наведено в додатку М.

Узагальнено логіка OpenCV-розпізнавання передбачає такі дії:

- декодування отриманого зображення;
- підготовку кадру до обробки;
- визначення ключових точок і дескрипторів;
- завантаження активних еталонних зображень;
- порівняння вхідного кадру з еталонами;
- відсіювання слабких збігів;
- перевірку геометричної узгодженості надійних збігів;
- формування кандидатів на розпізнавання;
- групування кандидатів за технічною міткою об’єкта;
- визначення фінального результату.

Важливо, що система обробляє контрольований сценарій: один об’єкт перед камерою, фіксований набір еталонів і сканування після дії користувача. Це відповідає меті дипломної роботи й не потребує реалізації повноцінної системи безперервного відеодетектування.

Після завершення розпізнавання серверна частина формує структуровану відповідь. Якщо об’єкт розпізнано достатньо впевнено, відповідь містить підтверджений результат. Якщо система не може надійно визначити об’єкт, формується непідтверджений, порожній або помилковий результат. У базі даних детально зберігаються підтверджені результати, а непевні випадки

відображаються через загальні поля запуску розпізнавання, зокрема повідомлення та кількість непідтверджених позицій.

Після отримання відповіді клієнтська частина аналізує стан результату. Якщо результат є підтвердженим, система автоматично додає розпізнаний товар до віртуального чека. Якщо результат непідтверджений, порожній або помилковий, товар не додається, а користувач отримує відповідне повідомлення. У поточній реалізації користувач не виконує окрему дію “додати до чека” після розпізнавання.

Другим важливим алгоритмом є формування еталонного зображення через адміністративний екран. Цей сценарій потрібен для розширення бібліотеки об’єктів, які система може розпізнавати. На відміну від основного сценарію покупця, адміністративний режим призначений не для сканування товару в чек, а для підготовки даних, які надалі використовуються модулем розпізнавання.

У додатку Н наведено блок-схему алгоритму формування еталонного зображення в адміністративному режимі.

Алгоритм адміністративного формування еталону виконується в такій послідовності:

- адміністратор відкриває адміністративний екран;
- система перевіряє пароль адміністративного доступу;
- адміністратор переглядає активний каталог об’єктів або додає новий об’єкт;
- якщо додається новий об’єкт, серверна частина створює запис у каталозі та готує файлову структуру для еталонних зображень;
- адміністратор вибирає об’єкт каталогу;
- адміністратор вибирає сторону або ракурс еталонного кадру;
- система отримує поточний кадр із вебкамери;
- кадр надсилається на серверну частину для збереження;
- серверна частина записує файл еталонного зображення у відповідну папку;

- система додає або оновлює метадані еталонного зображення в SQLite;
- еталон стає доступним для наступних запусків розпізнавання.

Такий алгоритм є важливим для практичного використання системи, оскільки дозволяє формувати еталони в тих самих умовах, у яких потім відбувається сканування. Це підвищує узгодженість між еталонними зображеннями та кадрами, отриманими під час роботи покупця. Адміністративний режим дозволяє додати новий об’єкт у каталог, створити для нього структуру еталонних зображень, вибрати сторону або ракурс і зберегти кадр із вебкамери як reference image.

Програмно адміністративний сценарій поділяється між frontend-екраном, API-клієнтом, backend-маршрутом і сервісом еталонної бібліотеки. Клієнтська частина відповідає за вибір об’єкта, вибір сторони та отримання кадру. Серверна частина відповідає за перевірку адміністративного доступу, створення або оновлення запису каталогу, збереження файлу та синхронізацію метаданих з базою даних. Такий поділ дозволяє не змішувати адміністративну логіку з основним сценарієм покупця.

Лістинги ключових програмних модулів, що реалізують описані алгоритми, доцільно навести в додатках. До таких модулів належать компоненти отримання кадру з вебкамери, API-клієнти frontend-частини, маршрути серверної частини, сервіс розпізнавання, OpenCV-механізм порівняння зображень, сервіс адміністративної бібліотеки еталонів і репозиторій збереження запусків розпізнавання.

У результаті алгоритмізація та програмування модулів забезпечують практичну реалізацію двох ключових сценаріїв системи: розпізнавання об’єкта покупцем і формування еталонів адміністратором. Основний сценарій пов’язує отримання кадру з серверною обробкою та автоматичним оновленням чека. Адміністративний сценарій забезпечує можливість розширення каталогу без ручного втручання у файлову структуру або базу даних. Разом ці алгоритми формують основу роботи програмного забезпечення та забезпечують зв’язок між

інтерфейсом, модулем комп’ютерного зору й інформаційним забезпеченням системи.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

У цьому розділі розглядаються питання перевірки працездатності, впровадження та подальшої експлуатації розробленого програмного забезпечення. Якщо в попередньому розділі було описано програмну реалізацію системи, її компоненти, інструментарій та алгоритми роботи, то в цьому розділі основна увага приділяється практичному використанню застосунку, тестуванню основних сценаріїв, вимогам до середовища запуску та складу інсталяційного пакета.

Розроблена система є веборієнтованим клієнт-серверним застосунком, тому її впровадження пов’язане не лише з розміщенням програмних компонентів, а й з перевіркою взаємодії клієнтської та серверної частин, доступності вебкамери, коректності обробки запитів і збереження службових даних. У межах розділу подано результати тестування основних сценаріїв роботи, визначено вимоги до апаратного та програмного забезпечення, а також описано склад пакета, необхідного для локального запуску й хмарного розгортання системи.

4.1 Тестування системи

Тестування системи проводилося з метою перевірки коректності роботи основних функцій програмного забезпечення, зручності користувацького сценарію, стабільності взаємодії клієнтської та серверної частин, а також правильності роботи адміністративного режиму. Оскільки система призначена для ідентифікації одного об’єкта за кадром із вебкамери, основна увага приділялася перевірці сценаріїв сканування, обробки результату розпізнавання та оновлення віртуального чека.

У межах підрозділу розглянуто ручне функціональне тестування основних сценаріїв роботи системи. Такий тип тестування передбачає перевірку поведінки програмного забезпечення з погляду користувача: задається певна дія, визначається очікуваний результат, після чого фактична поведінка системи

порівнюється з очікуваною [26]. Для розробленого застосунку перевірка охоплювала основний користувацький сценарій сканування, негативні сценарії розпізнавання, роботу адміністративного режиму, збереження службових даних і доступність серверної частини після локального та хмарного запуску.

Тестування проводилося як у локальному середовищі розробки, так і на розгорнутій версії вебзастосунку. Локально клієнтська частина запускалася через Vite dev server, а серверна частина — через FastAPI/uvicorn. Для перевірки базової доступності серверної частини використовувалися кінцеві точки перевірки працездатності, а для перевірки повного сценарію — реальна взаємодія з вебкамерою та еталонними зображеннями.

Першим етапом було перевірено завантаження користувацького інтерфейсу та готовність системи до сканування. Під час цього тесту оцінювалося, чи відкривається застосунок у браузері, чи коректно відображаються основні елементи інтерфейсу, чи доступні блок вебкамери, віртуальний чек, кнопки керування та повідомлення про стан системи. Також перевірялося, що запит розпізнавання виконується через серверний API, а клієнтська частина відповідає за захоплення кадру та відображення отриманого результату.

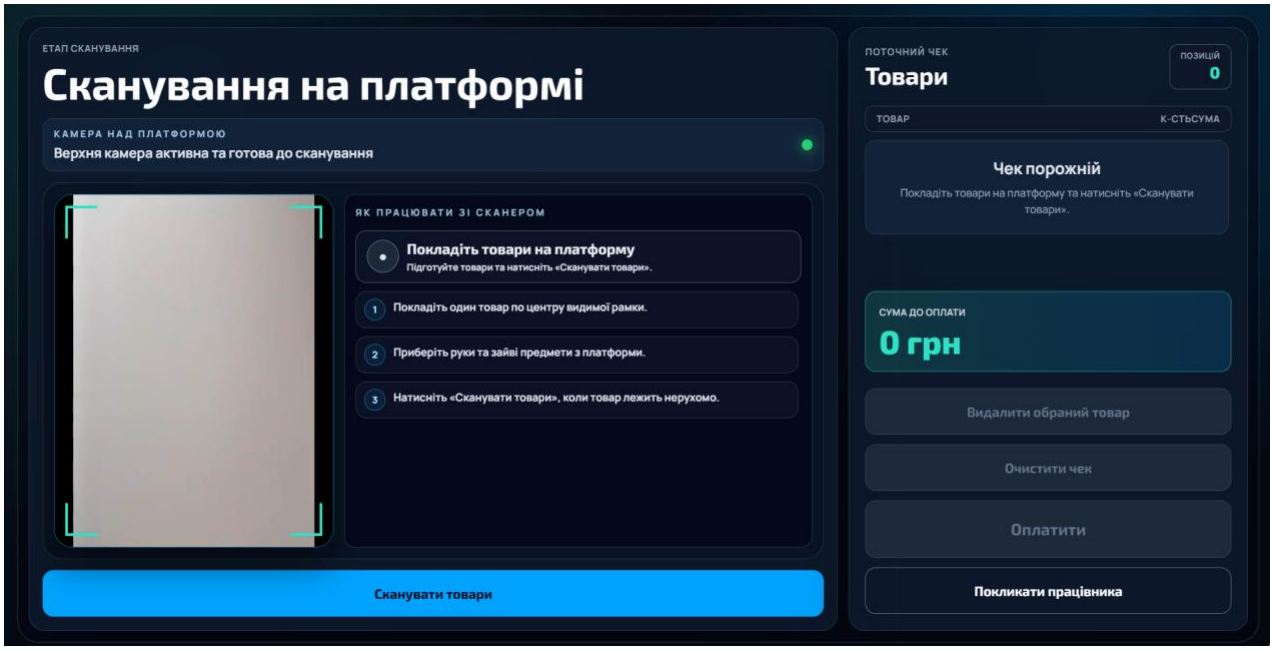


Рис. 4.1 Початковий екран сканування інтелектуальної каси самообслуговування

Другим етапом було перевірено роботу вебкамери. Система повинна запитувати дозвіл браузера на доступ до камери, після чого відображати відеопотік у відповідному компоненті інтерфейсу. У разі відсутності доступу до камери або відмови користувача система має показати зрозуміле повідомлення, а не завершувати роботу з помилкою. Після отримання доступу до камери інтерфейс відображає область сканування та інструкції для користувача.

Третім етапом перевірялося сканування об’єкта. Користувач розміщував книгу перед камерою та натискав кнопку сканування. Після цього клієнтська частина отримувала один кадр із відеопотоку та надсилала його на серверну частину у форматі multipart/form-data. Серверна частина виконувала обробку зображення, порівнювала кадр з еталонними зображеннями та повертала структуровану відповідь у форматі JSON. Такий підхід відповідає поточній реалізації системи, де розпізнавання виконується не безперервно у відеопотоці, а після натискання кнопки сканування для одного кадру.

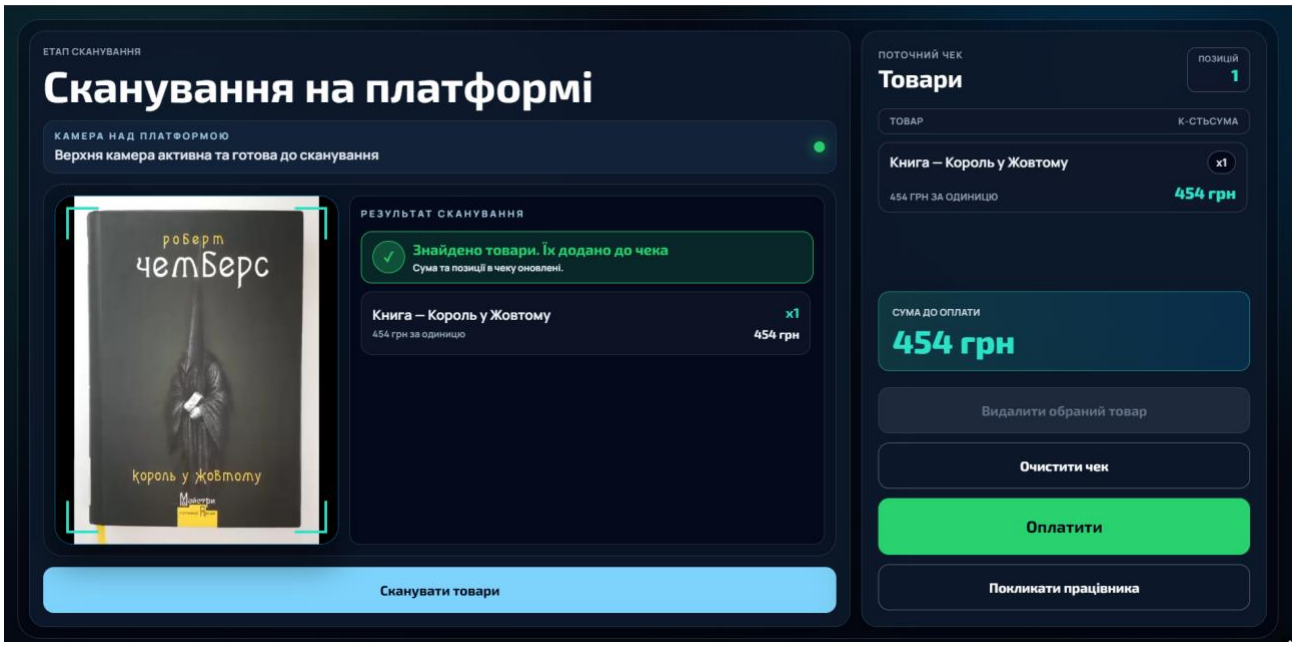


Рис. 4.2 Результат успішного розпізнавання та додавання товару до віртуального чека

У разі підтвердженого розпізнавання система показує назву розпізнаного товару, повідомлення про успішний результат і автоматично додає позицію до віртуального чека. Окрема дія “додати до чека” не виконується: якщо серверна частина повертає підтверджений результат, клієнтська частина самостійно оновлює стан чека, кількість позицій і загальну суму покупки.

Окремо перевірялися негативні та граничні сценарії. До них належать порожня область перед камерою, недостатньо якісний кадр, невідомий об’єкт, слабке освітлення або положення предмета, за якого система не може впевнено зіставити кадр з еталоном. У таких випадках товар не повинен додаватися до чека, а користувач має отримати повідомлення про порожній, непідтверджений або помилковий результат.

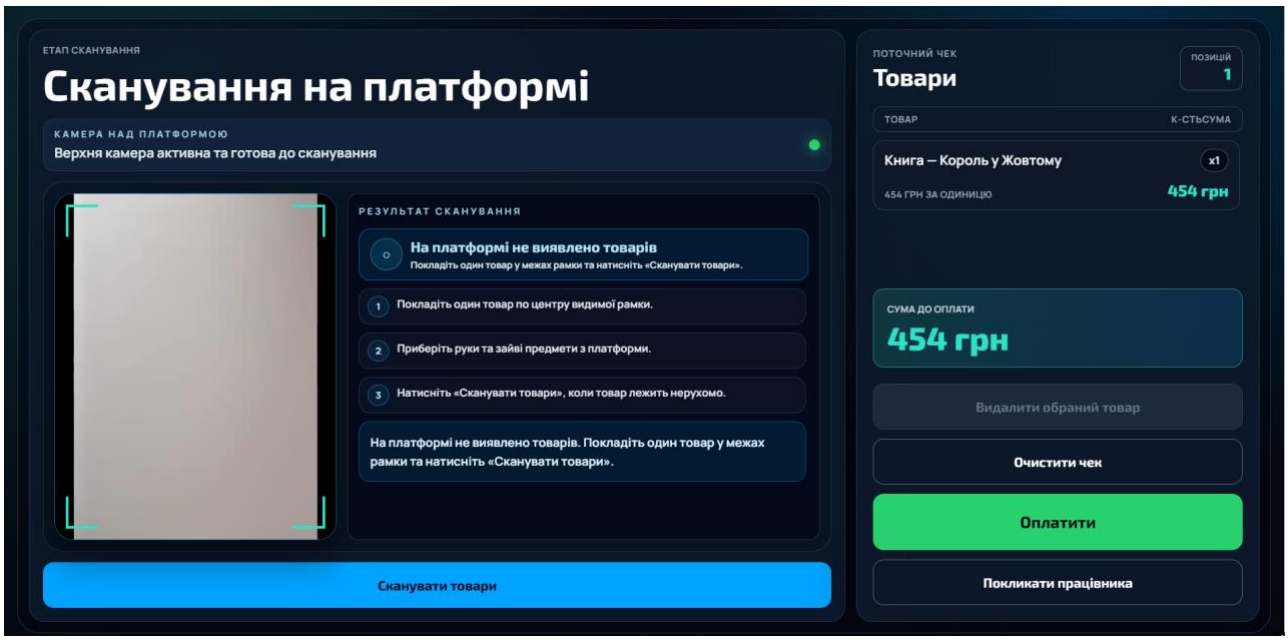


Рис. 4.3 Повідомлення системи у випадку порожньої платформи перед камерою

У цьому сценарії система не виявляє товар у зоні сканування та не додає позицію до віртуального чека. Користувач отримує інструкцію розмістити один товар у межах видимої рамки та повторити сканування. Такий сценарій є важливим, оскільки він перевіряє коректну поведінку системи за відсутності об’єкта в кадрі.

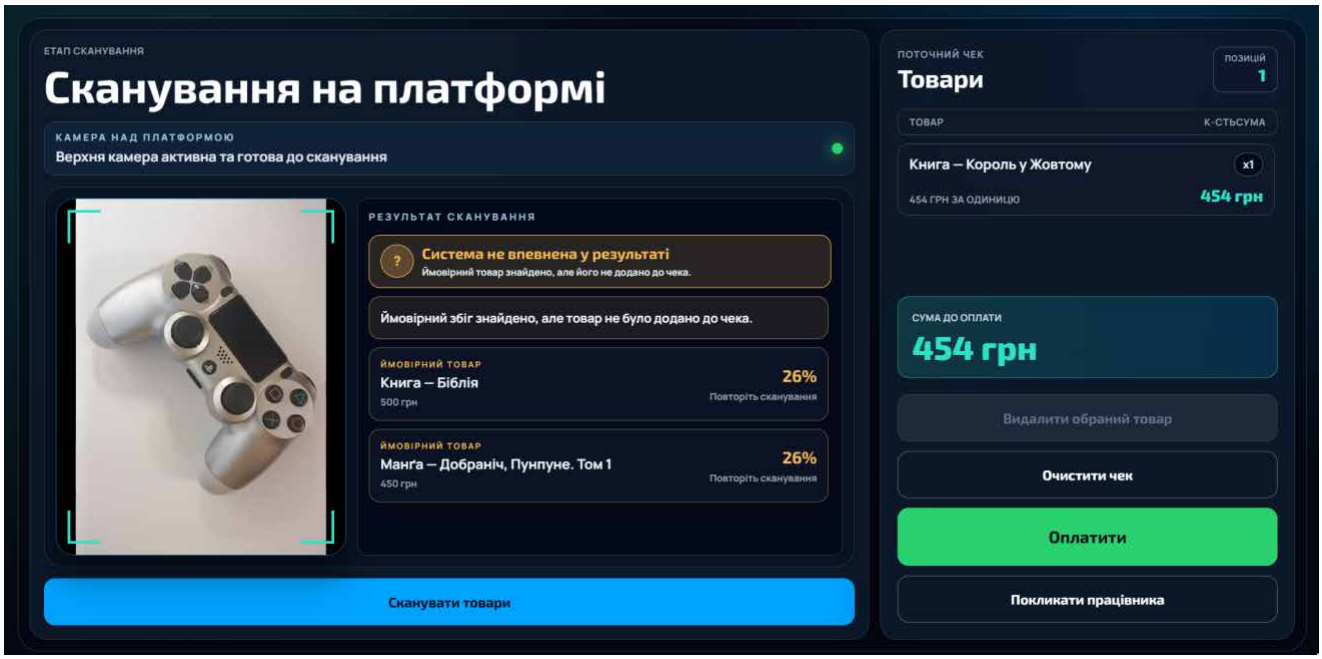


Рис. 4.4 Повідомлення системи у випадку непідтвердженого результату розпізнавання

У випадку непідтвердженого результату система може показати ймовірні збіги, однак не додає новий товар до чека автоматично. Це дозволяє уникнути помилкового додавання позиції, якщо рівень упевненості є недостатнім. Поточний стан чека при цьому залишається без змін, а користувач може повторити сканування за кращого положення об’єкта або освітлення.

Також було перевірено адміністративний режим. Він використовується не покупцем, а відповідальною особою для додавання нових об’єктів і формування еталонних зображень. Перед відкриттям адміністративного екрана система запитує пароль адміністративного доступу. Це дає змогу обмежити доступ до службових функцій, пов’язаних зі зміною каталогу та набору еталонних кадрів.

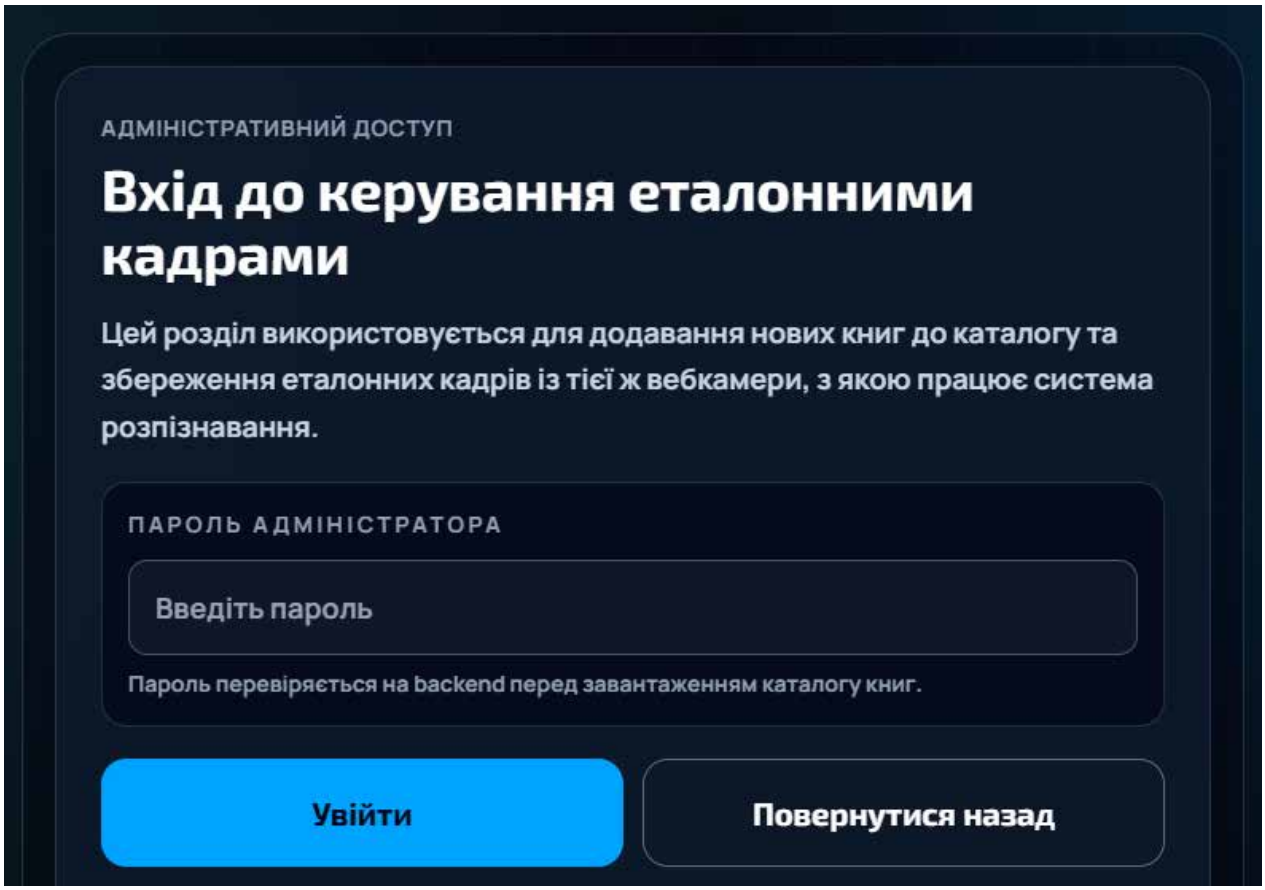


Рис. 4.5 Екран перевірки адміністративного доступу

Після успішного введення пароля адміністратор отримує доступ до екрана формування еталонних зображень. Адміністративний екран дозволяє переглянути активний каталог, додати нову книгу, вибрати наявний об’єкт, зазначити сторону або ракурс і зберегти поточний кадр з вебкамери як еталонне зображення. Після збереження система синхронізує інформацію про еталон із SQLite, а файл зображення зберігається у файловій структурі серверної частини.

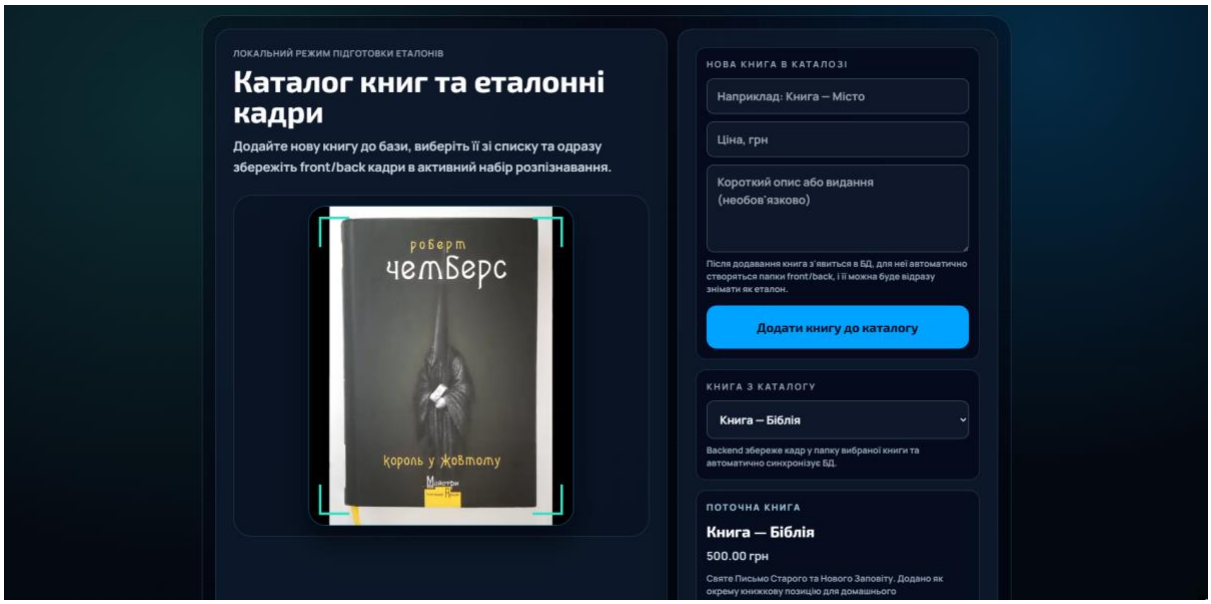


Рис. 4.6 Адміністративний екран формування еталонних зображень

Для узагальнення результатів тестування було складено набір тестових сценаріїв, наведений у таблиці в додатку П.

Крім ручного функціонального тестування, було перевірено технічну коректність збірки та запуску. Для клієнтської частини перевірялося виконання production-збірки командою npm run build, а для серверної частини — синтаксична коректність Python-модулів командою py -m compileall backend.

Окрему увагу було приділено перевірці хмарного розгортання. У цьому сценарії перевірялося, чи відкривається основний сайт, чи доступна адміністративна сторінка, чи правильно налаштована адреса серверного API, а також чи працює взаємодія між клієнтською та серверною частинами після розгортання. Якщо клієнтська частина відкривається без налаштованої адреси серверного API, система повинна показати зрозуміле повідомлення, а не виконувати некоректні запити.

Результати тестування показали, що розроблена система виконує основні функціональні вимоги: відображає інтерфейс каси самообслуговування, працює з вебкамерою, отримує кадр після натискання кнопки сканування, надсилає його на серверну частину, обробляє результат і оновлює віртуальний чек у разі підтвердженого розпізнавання. Також система коректно обробляє порожню платформу, непідтверджений результат і не додає товар до чека без достатньої

впевненості. Адміністративний режим дозволяє захистити доступ паролем, розширювати каталог об’єктів і формувати еталонні кадри без ручного редагування бази даних або файлової структури.

Водночас тестування підтвердило обмеження поточної реалізації: якість розпізнавання залежить від освітлення, положення книги, стабільності камери та якості еталонних зображень. Система найкраще працює в контрольованих умовах з одним об’єктом у кадрі, що відповідає її призначенню як демонстраційного дипломного прототипу.

4.2 Вимоги до апаратного та програмного забезпечення

Для коректної експлуатації розробленої системи необхідно визначити вимоги до апаратного та програмного забезпечення. Оскільки програмний продукт реалізовано як веборієнтовану клієнт-серверну систему, вимоги доцільно розглядати відповідно до основних вузлів її розміщення: клієнтського пристрою користувача, середовища розгортання клієнтської частини, середовища серверної частини та сховища даних.

Поточне розгортання програмного забезпечення виконано в середовищі Microsoft Azure. Клієнтська та серверна частини системи розгорнуті як окремі вебзастосунки в Azure App Service [16]. Такий підхід забезпечує логічне розділення frontend- і backend-рівнів та відповідає клієнт-серверній архітектурі системи. Було побудовано діаграму розміщення (рис. 4.7).

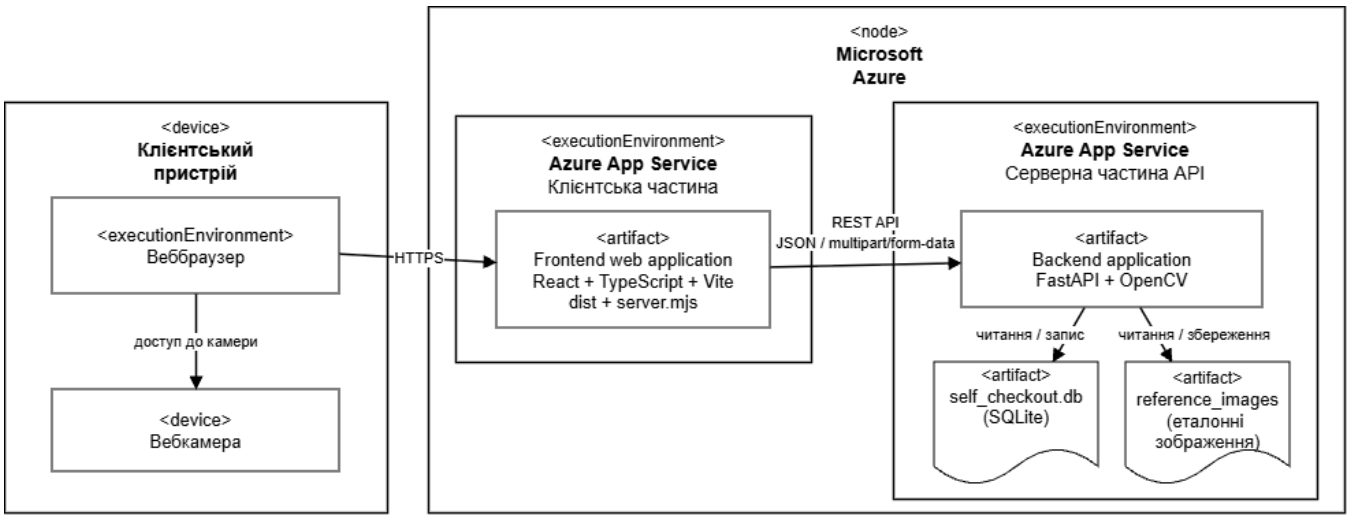


Рис. 4.7 Діаграма розміщення системи ідентифікації об’єктів за зображенням

На діаграмі показано клієнтський пристрій із браузером і вебкамерою, клієнтський застосунок в Azure App Service, серверний застосунок в Azure App Service, SQLite-базу даних і файлову структуру еталонних зображень на стороні серверної частини. Клієнтський пристрій використовується для відкриття інтерфейсу, роботи з вебкамерою та запуску сканування. Клієнтський застосунок відповідає за відображення інтерфейсу та надсилання запитів до серверного API. Серверний застосунок приймає кадри, виконує обробку зображення, звертається до даних каталогу та повертає результат розпізнавання.

Клієнтський пристрій повинен мати сучасний веббраузер із підтримкою доступу до вебкамери. Для роботи з камерою користувач має надати відповідний дозвіл у браузері. Також важливо, щоб застосунок відкривався через захищене з’єднання, оскільки браузери обмежують доступ до камери для незахищених сторінок [27]. Для стабільного розпізнавання бажано використовувати фіксоване положення камери, рівномірне освітлення, сталий фон та один об’єкт у кадрі.

Середовище розгортання клієнтської частини повинно забезпечувати віддавання зібраного вебзастосунку. У поточній реалізації для цього використовується Azure App Service з Node.js-середовищем. Production-збірка клієнтської частини формується за допомогою Vite, а зібрані файли розміщуються в каталозі dist. Для коректної взаємодії з серверною частиною має бути налаштована адреса API через змінну середовища VITE_API_BASE_URL.

Серверна частина потребує Python-середовища з установленими залежностями FastAPI, OpenCV та іншими бібліотеками, необхідними для обробки зображень. Серверний застосунок повинен мати доступ до SQLite-файлу та файлової структури еталонних зображень, оскільки в поточній реалізації структуровані дані й еталонні файли зберігаються на стороні серверного застосунку. Для адміністративного режиму також необхідно налаштувати пароль доступу через змінну середовища REFERENCE_ADMIN_PASSWORD.

Узагальнені вимоги до апаратного та програмного забезпечення наведено в табл. 4.2.1.

Таблиця 4.2

Вимоги до апаратного та програмного забезпечення системи

Компонент / середовище	**Основні вимоги**	**Призначення**
Клієнтський пристрій	Комп’ютер, ноутбук або термінал із сучасним веббраузером	Відкриття інтерфейсу каси самообслуговування
Вебкамера	Вбудована або зовнішня камера з достатньою якістю зображення	Отримання кадру об’єкта для розпізнавання
Браузер користувача	Підтримка доступу до камери, дозвіл на використання вебкамери, робота через захищене з’єднання	Відображення відеопотоку та передавання кадру до системи
Умови сканування	Стабільне освітлення, фіксований фон, один об’єкт у кадрі	Підвищення точності та стабільності розпізнавання
Клієнтська частина	Azure App Service / Node.js, production-збірка dist, налаштований VITE_API_BASE_URL	Віддавання вебінтерфейсу та зв’язок із backend API
Серверна частина	Python, FastAPI, OpenCV, залежності з requirements.txt	Приймання кадру, обробка зображення та формування відповіді
Сховище даних	SQLite-файл і файлова структура еталонних зображень	Зберігання каталогу, метаданих еталонів та історії сканувань
Хмарне розгортання	Окремі Azure App Service для frontend і backend	Розміщення системи у хмарному середовищі

Поточна реалізація орієнтована на демонстраційне single-instance розгортання. SQLite-файл і еталонні зображення зберігаються на стороні серверного застосунку, тому така організація є достатньою для дипломного проєкту та контрольованого сценарію роботи. Для промислового використання з масштабуванням на кілька серверних екземплярів доцільно винести структуровані дані до серверної СУБД, а еталонні зображення — до окремого хмарного файлового сховища.

Таким чином, для експлуатації системи необхідні клієнтський пристрій із браузером і вебкамерою, стабільне мережеве підключення, середовище розгортання клієнтської частини, Python-середовище для серверного застосунку та сховище для SQLite-файлу й еталонних зображень. Визначені вимоги відповідають поточній реалізації програмного забезпечення та враховують його призначення як дипломного вебзастосунку для ідентифікації одного об’єкта за зображенням у контрольованих умовах.

4.3 Склад інсталяційного пакету

Склад інсталяційного пакету визначає набір файлів, конфігурацій і допоміжних компонентів, необхідних для розгортання та запуску програмного забезпечення. Оскільки розроблена система є веборієнтованим клієнт-серверним застосунком, інсталяційний пакет не подається у вигляді одного класичного інсталятора типу .exe. У межах цієї роботи його доцільно розглядати як пакет розгортання, що складається з клієнтської частини, серверної частини, файлів інформаційного забезпечення, конфігураційних параметрів та засобів автоматизованого розгортання.

У поточній реалізації програмне забезпечення підготовлене до розгортання в Azure App Service [16]. Клієнтська та серверна частини розміщуються як окремі вебзастосунки, що відповідає клієнт-серверній структурі системи. Клієнтська частина після збірки пакується разом із файлом server.mjs, а серверна частина розгортається як Python/FastAPI-застосунок із власними залежностями та

командою запуску. Автоматизація розгортання виконується за допомогою GitHub Actions [17].

До складу клієнтської частини входять вихідні файли React-застосунку та конфігураційні файли Node.js-проєкту. Основними елементами є каталог src, файли package.json, package-lock.json, vite.config.ts і файл server.mjs. Каталог src містить компоненти, екрани, сервіси, типи та утиліти клієнтської частини. Після виконання production-збірки формується каталог dist, який містить готові статичні файли вебзастосунку. Для розгортання клієнтської частини в Azure App Service використовується зібраний каталог dist разом із файлом server.mjs, який забезпечує віддавання статичного застосунку та коректну роботу маршрутів.

До складу серверної частини входить каталог backend/app, у якому розміщено основні модулі FastAPI-застосунку: маршрути API, сервіси обробки зображень, модулі роботи з базою даних, репозиторії та схеми даних. Також до серверного пакету входить файл backend/requirements.txt, який містить перелік залежностей Python-проєкту. Під час розгортання ці залежності встановлюються в серверному середовищі, після чого запускається FastAPI-застосунок.

Окремою частиною пакету є файли інформаційного забезпечення. До них належать SQL-схема backend/app/db/schema.sql, модулі ініціалізації бази даних, seed-дані початкового каталогу об’єктів і модулі синхронізації еталонних зображень. SQL-схема використовується для створення таблиць SQLite-бази даних, а модулі ініціалізації забезпечують підготовку бази під час запуску серверної частини. Готовий SQLite-файл не є обов’язковою частиною пакету, оскільки база може бути створена автоматично на основі SQL-схеми та початкових даних.

До складу пакету також може входити початкова файлова структура еталонних зображень. Еталонні кадри зберігаються в каталозі backend/reference_images, а в базі даних фіксуються лише метадані та відносні шляхи до відповідних файлів. Якщо початковий набір еталонів наявний у пакеті, після запуску серверної частини виконується синхронізація файлової структури

з SQLite. Якщо еталони додаються вже після розгортання, це виконується через адміністративний екран /admin/reference-capture.

Конфігураційна частина пакету містить опис змінних середовища, необхідних для правильної роботи застосунку. Для клієнтської частини потрібно налаштувати VITE_API_BASE_URL, яка визначає адресу серверного API в хмарному середовищі. Для серверної частини потрібно налаштувати REFERENCE_ADMIN_PASSWORD, що використовується для захисту адміністративних дій. Значення секретів не включаються до інсталяційного пакету, а задаються окремо в середовищі розгортання або в налаштуваннях GitHub/Azure.

До deployment-частини входять workflow-файли GitHub Actions. Вони виконують встановлення залежностей, збірку клієнтської частини, підготовку серверного пакету, автентифікацію в Azure та розгортання відповідних частин у Azure App Service. Завдяки цьому оновлення програмного забезпечення може виконуватися автоматизовано після внесення змін до репозиторію.

Узагальнено інсталяційний пакет програмного забезпечення містить такі складові:

- клієнтський застосунок із вихідним кодом, конфігураційними файлами, production-збіркою dist та файлом server.mjs;
- серверний застосунок із каталогом backend/app, маршрутизаторами, сервісами, репозиторіями та схемами даних;
- файл залежностей backend/requirements.txt;
- SQL-схему та модулі ініціалізації SQLite-бази даних;
- seed-дані початкового каталогу об’єктів;
- файлову структуру еталонних зображень;
- workflow-файли GitHub Actions для розгортання клієнтської та серверної частин;
- опис необхідних змінних середовища без включення значень секретів до пакету;

- документацію або інструкції запуску для локального й хмарного середовища.

Таким чином, склад інсталяційного пакету відповідає клієнт-серверній природі розробленого вебзастосунку. Він охоплює вихідний код, production-збірку клієнтської частини, серверний пакет, SQL-схему, seed-дані, еталонні зображення, конфігураційні параметри та workflow-файли для автоматизованого розгортання. Такий склад дозволяє відтворити роботу системи як у локальному середовищі розробки, так і в хмарному середовищі Azure App Service.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено програмне забезпечення для ідентифікації об’єктів за зображенням у сценарії інтелектуальної каси самообслуговування. Розроблена система дає змогу користувачу виконати сканування одного об’єкта за допомогою вебкамери, передати отриманий кадр на серверну частину, отримати результат розпізнавання та відобразити його в інтерфейсі каси самообслуговування. У разі підтвердженого результату розпізнаний товар автоматично додається до віртуального чека.

У першому розділі було проаналізовано предметну область систем самообслуговування в роздрібній торгівлі. Розглянуто особливості використання кас самообслуговування, їхні переваги й обмеження, а також доцільність застосування технологій комп’ютерного зору для спрощення процесу ідентифікації товарів. Також було змодельовано предметну область, розглянуто існуючі аналогічні рішення та сформульовано постановку задачі для розроблення програмного продукту.

У другому розділі було спроєктовано інформаційне забезпечення системи. Визначено основні сутності, необхідні для роботи програмного забезпечення, побудовано логічну модель даних у вигляді ER-діаграми, обґрунтовано вибір SQLite як системи управління базою даних та описано фізичну структуру бази даних. Інформаційна база забезпечує зберігання каталогу об’єктів, метаданих еталонних зображень, запусків розпізнавання та підтверджених результатів роботи алгоритму. Еталонні зображення зберігаються у файловій структурі серверної частини, а база даних містить службові метадані та шляхи до відповідних файлів.

У третьому розділі було описано процес розроблення програмного забезпечення. Визначено основні абстракції предметної області, змодельовано структуру та взаємодію об’єктів, розглянуто організаційну й компонентну структуру системи, обґрунтовано вибір інструментарію та описано

алгоритмізацію основних програмних модулів. Програмне забезпечення реалізовано як клієнт-серверний вебзастосунок: клієнтська частина відповідає за взаємодію з користувачем і вебкамерою, а серверна частина - за обробку зображення, розпізнавання, роботу з еталонами та збереження службових результатів.

Для реалізації клієнтської частини було використано React, TypeScript, Vite і Tailwind CSS. Серверна частина реалізована мовою Python із використанням FastAPI та OpenCV. Розпізнавання об’єктів виконується за допомогою підходу OpenCV ORB feature matching, що дозволяє порівнювати кадр, отриманий із вебкамери, з еталонними зображеннями об’єктів. Також у системі реалізовано адміністративний режим, який дає змогу додавати нові об’єкти каталогу та формувати для них еталонні кадри без ручного редагування бази даних.

У четвертому розділі було розглянуто питання тестування, впровадження та експлуатації системи. Перевірено основні сценарії роботи програмного забезпечення: відкриття користувацького інтерфейсу, роботу з вебкамерою, сканування об’єкта, передавання кадру на серверну частину, отримання результату, автоматичне оновлення віртуального чека та роботу адміністративного режиму. Також було визначено вимоги до апаратного й програмного забезпечення та описано склад інсталяційного пакету.

Розроблений програмний продукт було розгорнуто у хмарному середовищі Microsoft Azure: клієнтська та серверна частини працюють як окремі вебзастосунки в Azure App Service. Це підтверджує можливість використання системи не лише в локальному середовищі розробки, а й як доступного вебзастосунку. Для демонстраційного сценарію використовується SQLite і файлова структура еталонних зображень, що відповідає поточному single-instance розгортанню.

Поставлену мету бакалаврської кваліфікаційної роботи було досягнуто. У межах роботи створено програмне забезпечення, яке реалізує повний цикл ідентифікації об’єкта за зображенням: отримання кадру з вебкамери, передавання його на серверну частину, порівняння з еталонними зображеннями,

формування результату розпізнавання, відображення результату користувачу та оновлення віртуального чека. Також реалізовано механізм адміністрування еталонів, що дозволяє розширювати каталог об’єктів.

Водночас система має певні обмеження. Поточна реалізація найкраще працює в контрольованих умовах: один об’єкт у кадрі, стабільне освітлення, фіксоване положення камери та якісні еталонні зображення. Використаний підхід OpenCV ORB не є повноцінною промисловою системою розпізнавання товарів у складному магазинному середовищі, однак він є достатнім для демонстрації принципу ідентифікації об’єктів за зображенням у межах дипломного проєкту.

Подальший розвиток системи може передбачати оптимізацію розпізнавання, кешування ознак еталонних зображень, розширення кількості підтримуваних об’єктів, використання хмарного сховища для еталонних кадрів, перехід на серверну СУБД для масштабованого розгортання, а також застосування нейромережевих моделей для більш надійного розпізнавання товарів у різних умовах зйомки.

Таким чином, виконана робота має практичне значення, оскільки демонструє можливість створення веборієнтованої системи ідентифікації об’єктів за зображенням із використанням сучасних засобів frontend- і backend-розробки, комп’ютерного зору, бази даних і хмарного розгортання. Розроблена система може бути використана як основа для подальшого розвитку інтелектуальних кас самообслуговування та інших програмних рішень, що потребують розпізнавання об’єктів за зображенням.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

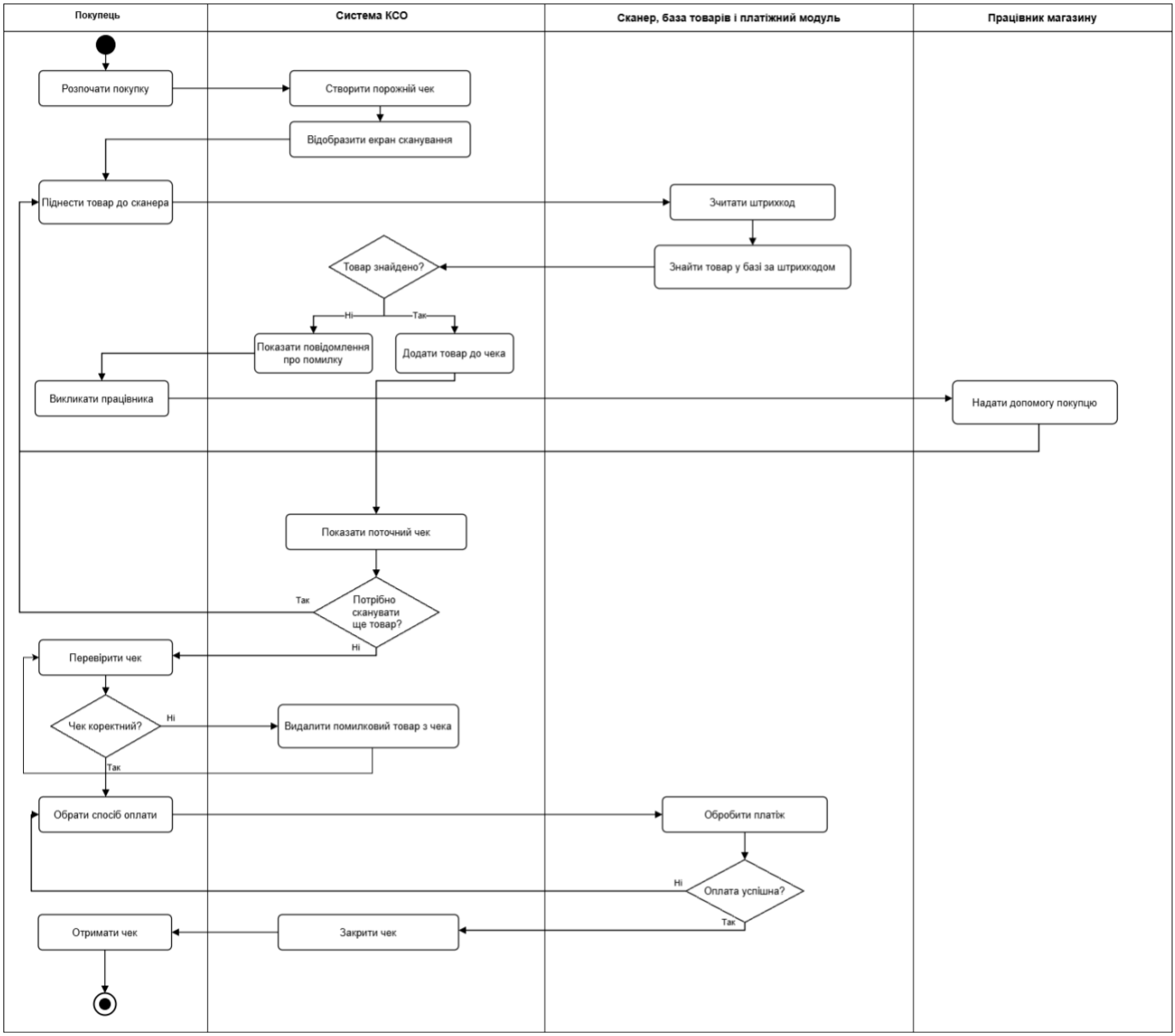
1. Self-Checkout Solutions for Retail Businesses: NCR Voyix. – [Електронний ресурс] – режим доступу: https://www.ncrvoyix.com/retail/self-checkout (дата звернення: 10.05.2026)
2. What is Object Detection?: IBM. – [Електронний ресурс] – режим доступу: https://www.ibm.com/think/topics/object-detection (дата звернення: 10.05.2026)
3. About the Unified Modeling Language Specification Version 2.5.1: Object Management Group. – [Електронний ресурс] – режим доступу: https://www.omg.org/spec/UML/2.5.1/About-UML (дата звернення: 11.05.2026)
4. GS1 Barcodes - Standards: GS1. – [Електронний ресурс] – режим доступу: https://www.gs1.org/standards/barcodes (дата звернення: 11.05.2026)
5. DN Series® EASY ONE: Diebold Nixdorf. – [Електронний ресурс] – режим доступу: https://www.dieboldnixdorf.com/en-us/retail/portfolio/systems/easy/easy-one/ (дата звернення: 21.05.2026)
6. Mashgin. The World's Fastest AI Checkout: Mashgin. – [Електронний ресурс] – режим доступу: https://www.mashgin.com/ (дата звернення: 12.05.2026)
7. MxP™ Self-Checkout 820: Toshiba Global Commerce Solutions. – [Електронний ресурс] – режим доступу: https://commerce.toshiba.com/wps/portal/marketing/?urile=wcm:path:/en-us/home/hardware/self-checkout/mxp-self-checkout (дата звернення: 12.05.2026)
8. What is an Entity Relationship Diagram?: IBM. – [Електронний ресурс] – режим доступу: https://www.ibm.com/think/topics/entity-relationship-diagram (дата звернення: 13.05.2026)
9. What Is Database Normalization?: IBM. – [Електронний ресурс] – режим доступу: https://www.ibm.com/think/topics/database-normalization (дата звернення: 13.05.2026)

10. About SQLite: SQLite. – [Електронний ресурс] – режим доступу: https://sqlite.org/about.html (дата звернення: 21.05.2026)
11. PostgreSQL: About: The PostgreSQL Global Development Group. – [Електронний ресурс] – режим доступу: https://www.postgresql.org/about/ (дата звернення: 14.05.2026)
12. MySQL 9.7 Reference Manual. What is MySQL?: Oracle. – [Електронний ресурс] – режим доступу: https://dev.mysql.com/doc/refman/9.7/en/what-is-mysql.html (дата звернення: 14.05.2026)
13. SQL Server technical documentation: Microsoft Learn. – [Електронний ресурс] – режим доступу: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver17 (дата звернення: 14.05.2026)
14. Datatypes In SQLite: SQLite. – [Електронний ресурс] – режим доступу: https://sqlite.org/datatype3.html (дата звернення: 15.05.2026)
15. SQLite Foreign Key Support: SQLite. – [Електронний ресурс] – режим доступу: https://sqlite.org/foreignkeys.html (дата звернення: 15.05.2026)
16. Azure App Service documentation: Microsoft Learn. – [Електронний ресурс] – режим доступу: https://learn.microsoft.com/en-us/azure/app-service/ (дата звернення: 16.05.2026)
17. Understanding GitHub Actions: GitHub Docs. – [Електронний ресурс] – режим доступу: https://docs.github.com/articles/getting-started-with-github-actions (дата звернення: 17.05.2026)
18. React: official documentation. – [Електронний ресурс] – режим доступу: https://react.dev/ (дата звернення: 17.05.2026)
19. TypeScript: JavaScript With Syntax For Types: Microsoft. – [Електронний ресурс] – режим доступу: https://www.typescriptlang.org/ (дата звернення: 18.05.2026)
20. Vite. Next Generation Frontend Tooling: Vite. – [Електронний ресурс] – режим доступу: https://vite.dev/ (дата звернення: 18.05.2026)
21. Tailwind CSS: official website. – [Електронний ресурс] – режим доступу: https://tailwindcss.com/ (дата звернення: 19.05.2026)

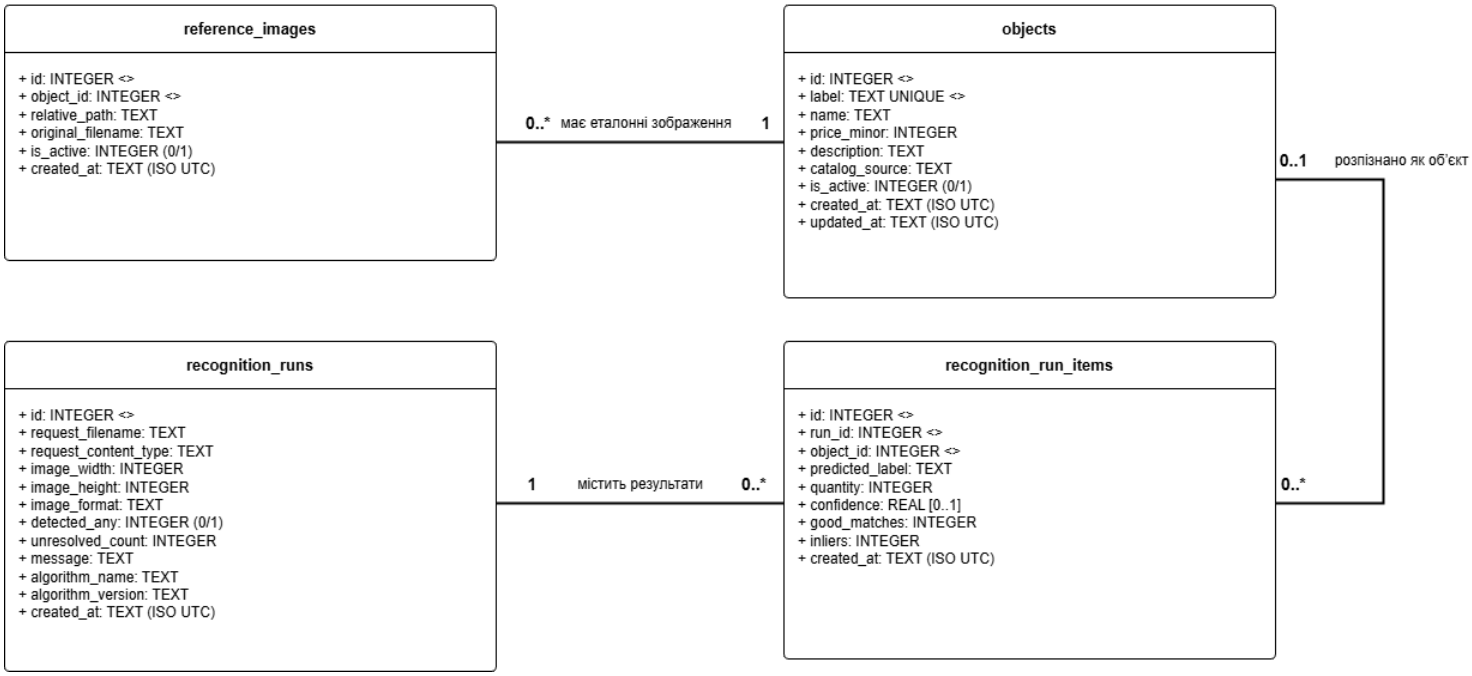
22. FastAPI: official documentation. – [Електронний ресурс] – режим доступу: https://fastapi.tiangolo.com/ (дата звернення: 19.05.2026)
23. OpenCV - Open Computer Vision Library: OpenCV. – [Електронний ресурс] – режим доступу: https://opencv.org/ (дата звернення: 20.05.2026)
24. OpenCV: cv::ORB Class Reference. – [Електронний ресурс] – режим доступу: https://docs.opencv.org/4.x/db/d95/classcv_1_1ORB.html (дата звернення: 20.05.2026)
25. OpenCV: Feature Matching. – [Електронний ресурс] – режим доступу: https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html (дата звернення: 21.05.2026)
26. What is Functional Testing?: IBM. – [Електронний ресурс] – режим доступу: https://www.ibm.com/think/topics/functional-testing (дата звернення: 21.05.2026)
27. MediaDevices: getUserMedia() method: MDN Web Docs. – [Електронний ресурс] – режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia (дата звернення: 21.05.2026)

ДОДАТКИ

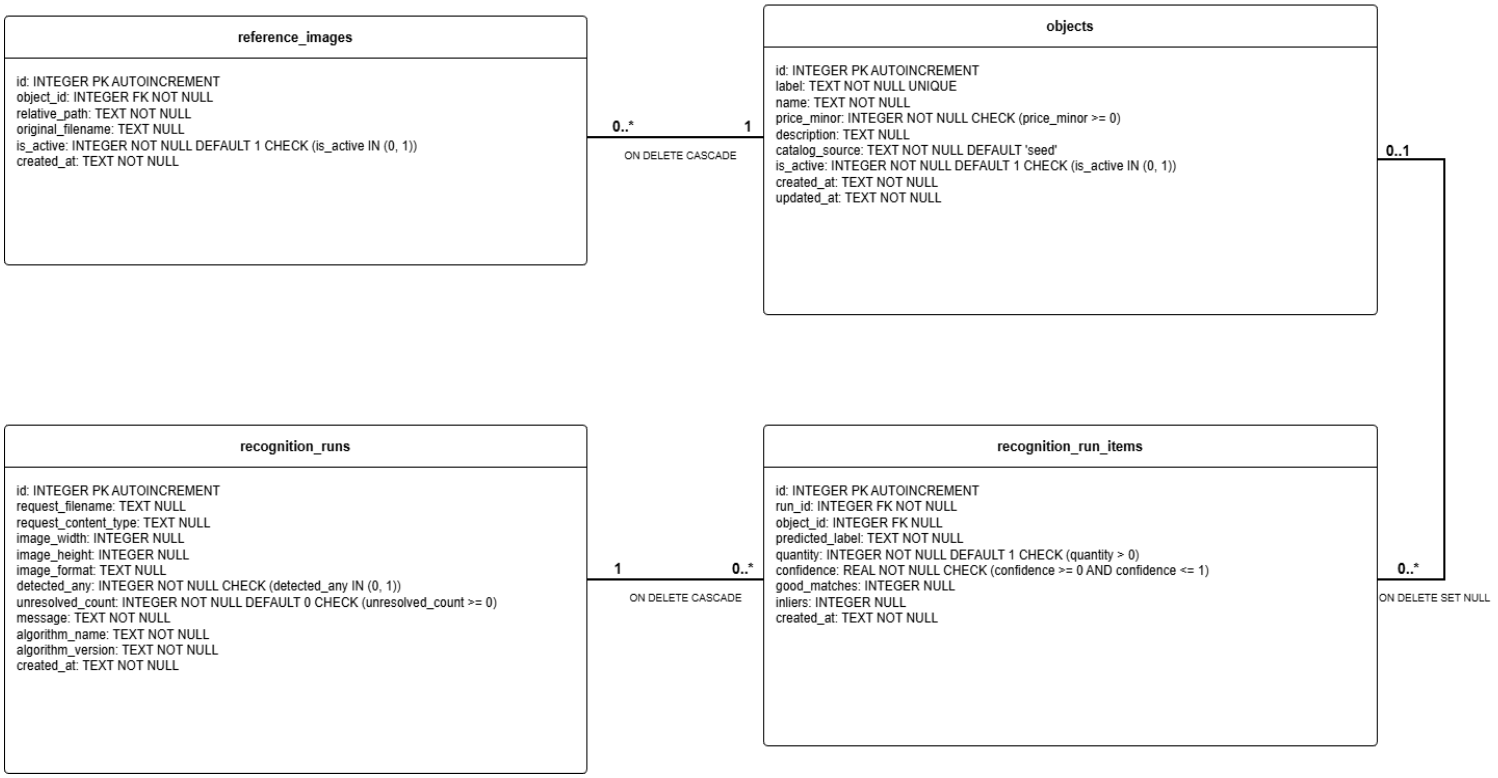
ДОДАТОК А



ДОДАТОК Б


reference_images
+ id: INTEGER <>
+ object_id: INTEGER <>
+ relative_path: TEXT
+ original_filename: TEXT
+ is_active: INTEGER (0/1)
+ created_at: TEXT (ISO UTC)
0..* має еталонні зображення 1
objects
+ id: INTEGER <>
+ label: TEXT UNIQUE <>
+ name: TEXT
+ price_minor: INTEGER
+ description: TEXT
+ catalog_source: TEXT
+ is_active: INTEGER (0/1)
+ created_at: TEXT (ISO UTC)
+ updated_at: TEXT (ISO UTC)
0..1 розпізнано як об'єкт
recognition_runs
+ id: INTEGER <>
+ request_filename: TEXT
+ request_content_type: TEXT
+ image_width: INTEGER
+ image_height: INTEGER
+ image_format: TEXT
+ detected_any: INTEGER (0/1)
+ unresolved_count: INTEGER
+ message: TEXT
+ algorithm_name: TEXT
+ algorithm_version: TEXT
+ created_at: TEXT (ISO UTC)
1 містить результати 0..*
recognition_run_items
+ id: INTEGER <>
+ run_id: INTEGER <>
+ object_id: INTEGER <>
+ predicted_label: TEXT
+ quantity: INTEGER
+ confidence: REAL [0..1]
+ good_matches: INTEGER
+ inliers: INTEGER
+ created_at: TEXT (ISO UTC)
0..*

ДОДАТОК В

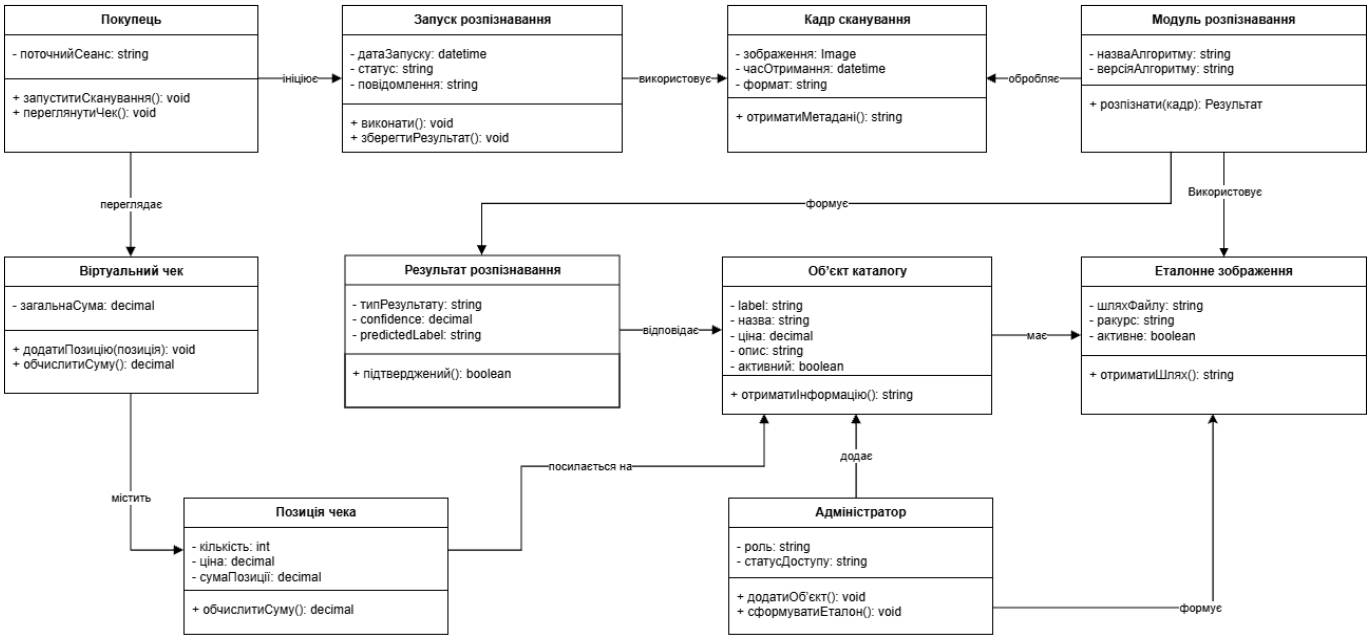

reference_images
id: INTEGER PK AUTOINCREMENT
object_id: INTEGER FK NOT NULL
relative_path: TEXT NOT NULL
original_filename: TEXT NULL
is_active: INTEGER NOT NULL DEFAULT 1 CHECK (is_active IN (0, 1))
created_at: TEXT NOT NULL
0..*
1
ON DELETE CASCADE
objects
id: INTEGER PK AUTOINCREMENT
label: TEXT NOT NULL UNIQUE
name: TEXT NOT NULL
price_minor: INTEGER NOT NULL CHECK (price_minor >= 0)
description: TEXT NULL
catalog_source: TEXT NOT NULL DEFAULT 'seed'
is_active: INTEGER NOT NULL DEFAULT 1 CHECK (is_active IN (0, 1))
created_at: TEXT NOT NULL
updated_at: TEXT NOT NULL
0..1
recognition_runs
id: INTEGER PK AUTOINCREMENT
request_filename: TEXT NULL
request_content_type: TEXT NULL
image_width: INTEGER NULL
image_height: INTEGER NULL
image_format: TEXT NULL
detected_any: INTEGER NOT NULL CHECK (detected_any IN (0, 1))
unresolved_count: INTEGER NOT NULL DEFAULT 0 CHECK (unresolved_count >= 0)
message: TEXT NOT NULL
algorithm_name: TEXT NOT NULL
algorithm_version: TEXT NOT NULL
created_at: TEXT NOT NULL
1
0..*
ON DELETE CASCADE
recognition_run_items
id: INTEGER PK AUTOINCREMENT
run_id: INTEGER FK NOT NULL
object_id: INTEGER FK NULL
predicted_label: TEXT NOT NULL
quantity: INTEGER NOT NULL DEFAULT 1 CHECK (quantity > 0)
confidence: REAL NOT NULL CHECK (confidence >= 0 AND confidence <= 1)
good_matches: INTEGER NULL
inliers: INTEGER NULL
created_at: TEXT NOT NULL
0..*
ON DELETE SET NULL

ДОДАТОК Д

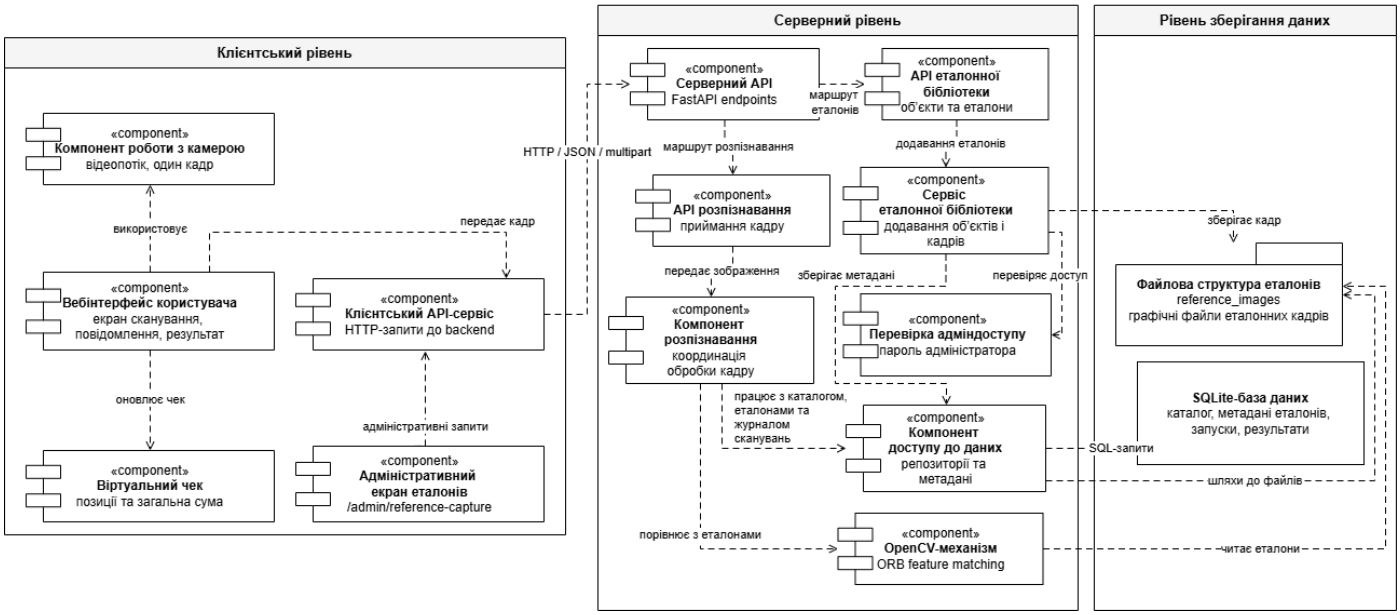
```
CREATE TABLE IF NOT EXISTS objects (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    label TEXT NOT NULL UNIQUE,
    name TEXT NOT NULL,
    price_minor INTEGER NOT NULL CHECK (price_minor >= 0),
    description TEXT,
    catalog_source TEXT NOT NULL DEFAULT 'seed',
    is_active INTEGER NOT NULL DEFAULT 1 CHECK (is_active IN (0, 1)),
    created_at TEXT NOT NULL,
    updated_at TEXT NOT NULL
);
CREATE TABLE IF NOT EXISTS reference_images (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    object_id INTEGER NOT NULL,
    relative_path TEXT NOT NULL,
    original_filename TEXT,
    is_active INTEGER NOT NULL DEFAULT 1 CHECK (is_active IN (0, 1)),
    created_at TEXT NOT NULL,
    FOREIGN KEY (object_id) REFERENCES objects(id) ON DELETE CASCADE
);
CREATE TABLE IF NOT EXISTS recognition_runs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    request_filename TEXT,
    request_content_type TEXT,
    image_width INTEGER,
    image_height INTEGER,
    image_format TEXT,
    detected_any INTEGER NOT NULL CHECK (detected_any IN (0, 1)),
    unresolved_count INTEGER NOT NULL DEFAULT 0 CHECK (unresolved_count >= 0),
    message TEXT NOT NULL,
    algorithm_name TEXT NOT NULL,
    algorithm_version TEXT NOT NULL,
    created_at TEXT NOT NULL
);
CREATE TABLE IF NOT EXISTS recognition_run_items (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    run_id INTEGER NOT NULL,
    object_id INTEGER,
    predicted_label TEXT NOT NULL,
    quantity INTEGER NOT NULL DEFAULT 1 CHECK (quantity > 0),
```

```
    confidence REAL NOT NULL CHECK (confidence >= 0 AND confidence <= 1),
    good_matches INTEGER,
    inliers INTEGER,
    created_at TEXT NOT NULL,
    FOREIGN KEY (run_id) REFERENCES recognition_runs(id) ON DELETE CASCADE,
    FOREIGN KEY (object_id) REFERENCES objects(id) ON DELETE SET NULL
);
CREATE INDEX IF NOT EXISTS idx_objects_label ON objects(label);
CREATE INDEX IF NOT EXISTS idx_reference_images_object_id ON reference_images(object_id);
CREATE INDEX IF NOT EXISTS idx_recognition_runs_created_at ON recognition_runs(created_at);
CREATE INDEX IF NOT EXISTS idx_recognition_run_items_run_id ON recognition_run_items(run_id);
CREATE INDEX IF NOT EXISTS idx_recognition_run_items_object_id ON recognition_run_items(object_id);
```

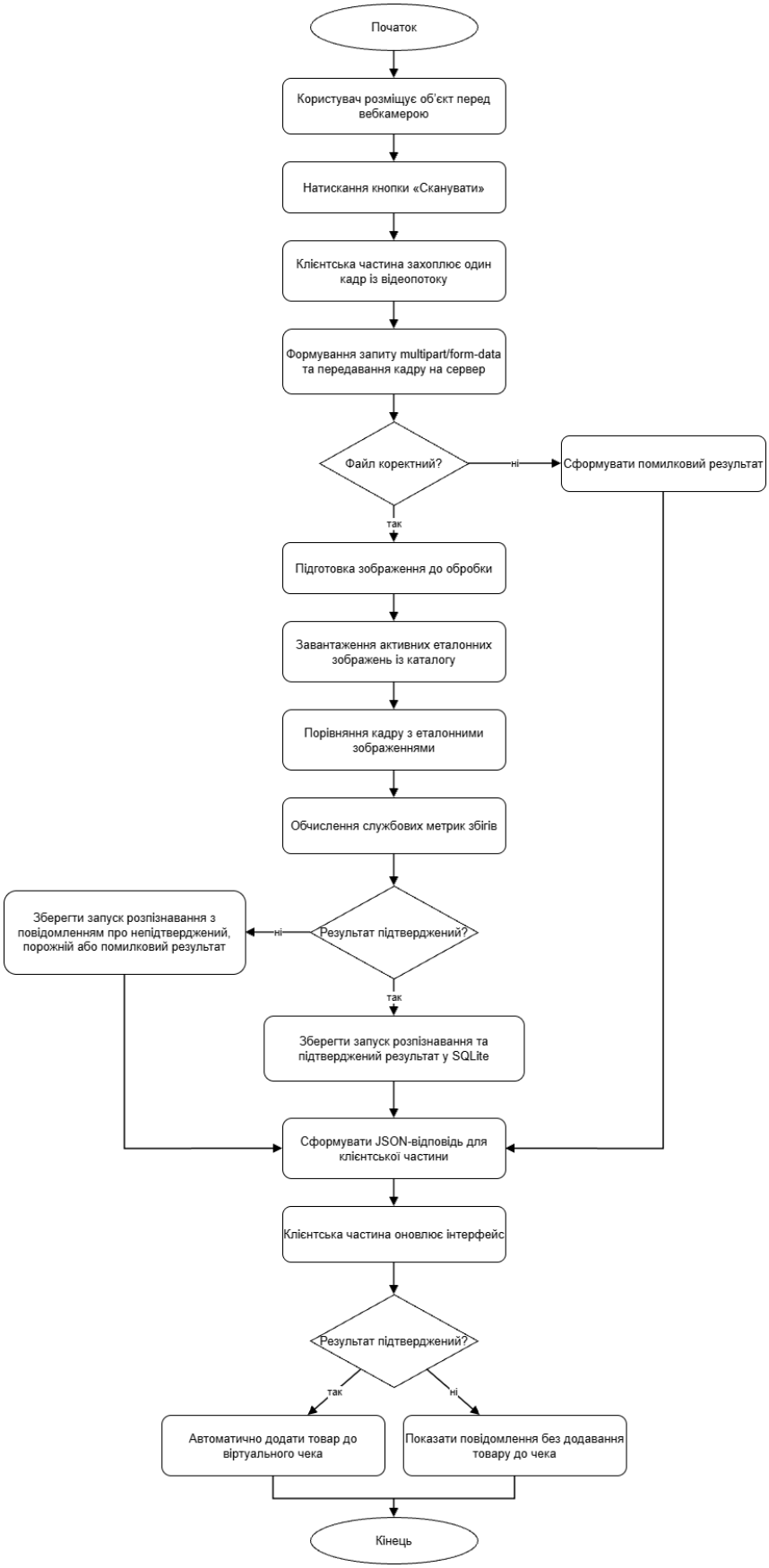
ДОДАТОК Е



ДОДАТОК Ж


Клієнтський рівень
«component»
Компонент роботи з камерою
відеопотік, один кадр
використовує
передає кадр
«component»
Вебінтерфейс користувача
екран сканування,
повідомлення, результат
«component»
Клієнтський API-сервіс
HTTP-запити до backend
оновлює чек
адміністративні запити
«component»
Віртуальний чек
позиції та загальна сума
«component»
Адміністративний
екран еталонів
/admin/reference-capture
HTTP / JSON / multipart
Серверний рівень
«component»
Серверний API
FastAPI endpoints
маршрут
еталонів
«component»
API еталонної
бібліотеки
об'єкти та еталони
маршрут розпізнавання
додавання еталонів
«component»
API розпізнавання
приймання кадру
«component»
Сервіс
еталонної бібліотеки
додавання об'єктів і
кадрів
передає зображення
зберігає метадані
перевіряє доступ
«component»
Компонент
розпізнавання
координація
обробки кадру
«component»
Перевірка адміндоступу
пароль адміністратора
працює з каталогом,
еталонами та
журналом
сканувань
«component»
Компонент
доступу до даних
репозиторії та
метадані
SQL-запити
порівнює з еталонами
«component»
OpenCV-механізм
ORB feature matching
Рівень зберігання даних
зберігає кадр
Файлова структура еталонів
reference_images
графічні файли еталонних кадрів
SQLite-база даних
каталог, метадані еталонів,
запуски, результати
шляхи до файлів
читає еталони

ДОДАТОК И


Початок
Користувач розміщує об'єкт перед вебкамерою
Натискання кнопки «Сканувати»
Клієнтська частина захоплює один кадр із відеопотоку
Формування запиту multipart/form-data та передавання кадру на сервер
Файл коректний?
ні
Сформувати помилковий результат
так
Підготовка зображення до обробки
Завантаження активних еталонних зображень із каталогу
Порівняння кадру з еталонними зображеннями
Обчислення службових метрик збігів
Результат підтверджений?
ні
Зберегти запуск розпізнавання з повідомленням про непідтверджений, порожній або помилковий результат
так
Зберегти запуск розпізнавання та підтверджений результат у SQLite
Сформувати JSON-відповідь для клієнтської частини
Клієнтська частина оновлює інтерфейс
Результат підтверджений?
так
ні
Автоматично додати товар до віртуального чека
Показати повідомлення без додавання товару до чека
Кінець

ДОДАТОК К

```
const canvas = document.createElement('canvas');
canvas.width = videoElement.videoWidth;
canvas.height = videoElement.videoHeight;

const context = canvas.getContext('2d');
if (!context) {
  throw new Error('Unable to capture the current video frame.');
}

context.drawImage(videoElement, 0, 0, canvas.width, canvas.height);

return await new Promise<Blob>((resolve, reject) => {
  canvas.toBlob(
    (blob) => {
      if (!blob) {
        reject(new Error('Unable to convert the captured frame to an image.'));
        return;
      }

      resolve(blob);
    },
    'image/jpeg',
    0.92,
  );
});
```

ДОДАТОК Л

```
import { buildApiUrl } from '../config/api';
import type { PredictionResponse } from '../types';

const predictionPath = '/api/inference/predict';

export const sendFrameForPrediction = async (imageBlob: Blob): Promise<PredictionResponse> => {
  const formData = new FormData();
  formData.append('image', imageBlob, 'checkout-frame.jpg');

  const response = await fetch(buildApiUrl(predictionPath), {
    method: 'POST',
    body: formData,
  });

  if (!response.ok) {
    let errorMessage = 'Сервер не зміг обробити запит.';

    try {
      const errorPayload = (await response.json()) as { detail?: string };
      if (errorPayload.detail) {
        errorMessage = errorPayload.detail;
      }
    } catch {
      // Ignore JSON parsing failures and keep the generic message.
    }

    throw new Error(errorMessage);
  }

  return (await response.json()) as PredictionResponse;
};
```

ДОДАТОК М

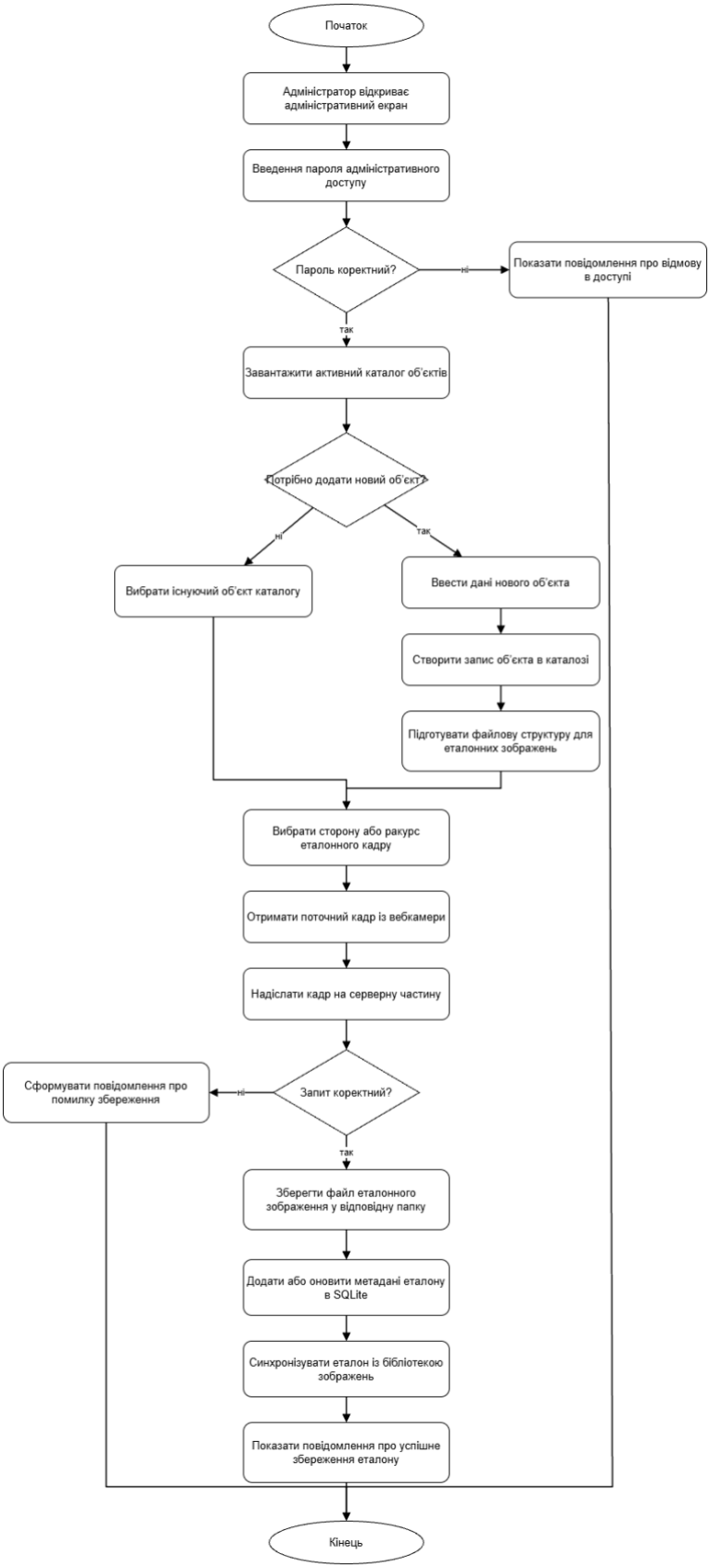
```
class OpenCVRecognitionEngine:
    algorithm_name = "opencv-orb-feature-matching"
    algorithm_version = "1.1.0"

    def __init__(self) -> None:
        self._orb = cv2.ORB_create(
            nfeatures=1500,
            scaleFactor=1.2,
            nlevels=8,
            edgeThreshold=15,
            patchSize=31,
            fastThreshold=10,
        )
        self._matcher = cv2.BFMatcher(cv2.NORM_HAMMING)
        self._ratio_threshold = 0.75
        self._min_scene_keypoints = 24
        self._min_reference_keypoints = 18
        self._min_good_matches_for_detection = 18
        self._min_inliers_for_detection = 12
        self._min_inlier_ratio_for_detection = 0.52
        self._min_confidence_for_detection = 0.5

    def recognize(
        self,
        *,
        image_bytes: bytes,
        reference_images: list[ReferenceImageRecord],
    ) -> OpenCVRecognitionResult:
        scene_image = self._decode_grayscale(image_bytes)
        if scene_image is None:
            return OpenCVRecognitionResult(
                detected_items=[],
                uncertain_items=[],
                unresolved_count=0,
                message="Не вдалося обробити кадр для розпізнавання.",
            )

        scene_keypoints, scene_descriptors = self._orb.detectAndCompute(scene_image, None)
        if scene_descriptors is None or len(scene_keypoints) < self._min_scene_keypoints:
            return OpenCVRecognitionResult(
```

```
                detected_items=[],
                uncertain_items=[],
                unresolved_count=0,
                message="На платформі не виявлено товарів. Покладіть один товар у межах рамки та натисніть «Сканувати товари».",
            )

    def _match_reference(
        self,
        *,
        reference_image: ReferenceImageRecord,
        scene_keypoints: list[cv2.KeyPoint],
        scene_descriptors: np.ndarray,
        scene_shape: tuple[int, ...],
    ) -> OpenCVMatchCandidate | None:
        reference_matrix = cv2.imread(str(reference_image.absolute_path), cv2.IMREAD_GRAYSCALE)
        if reference_matrix is None:
            return None

        best_candidate: OpenCVMatchCandidate | None = None

        for rotation_degrees, rotated_reference in self._generate_reference_variants(reference_matrix):
            reference_keypoints, reference_descriptors = self._orb.detectAndCompute(rotated_reference, None)
            if reference_descriptors is None or len(reference_keypoints) < self._min_reference_keypoints:
                continue

            knn_matches = self._matcher.knnMatch(reference_descriptors, scene_descriptors, k=2)
            good_matches = []

            for pair in knn_matches:
                if len(pair) < 2:
                    continue

                first_match, second_match = pair
                if first_match.distance < self._ratio_threshold * second_match.distance:
                    good_matches.append(first_match)

            good_matches_count = len(good_matches)
            if good_matches_count == 0:
                continue

            inliers = 0
```

```
inlier_ratio = 0.0
geometry_valid = False

if good_matches_count >= 4:
    reference_points = np.float32(
        [reference_keypoints[match.queryIdx].pt for match in good_matches]
    ).reshape(-1, 1, 2)
    scene_points = np.float32(
        [scene_keypoints[match.trainIdx].pt for match in good_matches]
    ).reshape(-1, 1, 2)

    homography, mask = cv2.findHomography(reference_points, scene_points, cv2.RANSAC, 5.0)
    if mask is not None:
        inliers = int(mask.ravel().sum())
        inlier_ratio = inliers / good_matches_count if good_matches_count else 0.0

    if homography is not None:
        area_ratio, geometry_valid = self._evaluate_projected_area(
            homography=homography,
            reference_shape=rotated_reference.shape,
            scene_shape=scene_shape,
        )

confidence = self._calculate_confidence(
    good_matches=good_matches_count,
    inliers=inliers,
    inlier_ratio=inlier_ratio,
    geometry_valid=geometry_valid,
)

candidate = OpenCVMatchCandidate(
    reference_image_id=reference_image.id,
    object_id=reference_image.object_id,
    label=reference_image.label,
    view_group=reference_image.view_group,
    name=reference_image.name,
    price=reference_image.price,
    quantity=1,
    confidence=confidence,
    reference_keypoints=len(reference_keypoints),
    good_matches=good_matches_count,
    inliers=inliers,
```

```
                inlier_ratio=round(inlier_ratio, 2),
                area_ratio=round(area_ratio, 4),
                geometry_valid=geometry_valid,
            )

            if best_candidate is None or self._candidate_sort_key(candidate) >
self._candidate_sort_key(best_candidate):
                best_candidate = candidate

        return best_candidate

    def _calculate_confidence(
        self,
        *,
        good_matches: int,
        inliers: int,
        inlier_ratio: float,
        geometry_valid: bool,
    ) -> float:
        good_match_component = min(good_matches / 55.0, 1.0)
        inlier_component = min(inliers / 30.0, 1.0)
        inlier_ratio_component = min(inlier_ratio / 0.75, 1.0)
        geometry_component = 1.0 if geometry_valid else 0.0
        confidence = (
            (good_match_component * 0.25)
            + (inlier_component * 0.35)
            + (inlier_ratio_component * 0.25)
            + (geometry_component * 0.15)
        )
        return round(confidence, 2)

    def _decode_grayscale(self, image_bytes: bytes) -> np.ndarray | None:
        byte_array = np.frombuffer(image_bytes, dtype=np.uint8)
        return cv2.imdecode(byte_array, cv2.IMREAD_GRAYSCALE)
```

ДОДАТОК Н


Початок
Адміністратор відкриває адміністративний екран
Введення пароля адміністративного доступу
Пароль коректний?
ні
Показати повідомлення про відмову в доступі
так
Завантажити активний каталог об'єктів
Потрібно додати новий об'єкт?
ні
так
Вибрати існуючий об'єкт каталогу
Ввести дані нового об'єкта
Створити запис об'єкта в каталозі
Підготувати файлову структуру для еталонних зображень
Вибрати сторону або ракурс еталонного кадру
Отримати поточний кадр із вебкамери
Надіслати кадр на серверну частину
Сформувати повідомлення про помилку збереження
ні
Запит коректний?
так
Зберегти файл еталонного зображення у відповідну папку
Додати або оновити метадані еталону в SQLite
Синхронізувати еталон із бібліотекою зображень
Показати повідомлення про успішне збереження еталону
Кінець

ДОДАТОК П

№	Тестовий сценарій	Очікуваний результат	Фактичний результат
1	Відкрити головний екран застосунку	Інтерфейс завантажується без помилок	Екран відкрився, основні елементи відображено
2	Перейти до екрана сканування	Відображається екран із камерою та віртуальним чеком	Екран сканування відкрився, камера й чек доступні
3	Надати доступ до вебкамери	У браузері відображається відеопотік	Дозвіл надано, відеопотік відображено
4	Натиснути кнопку сканування з книгою перед камерою	Кадр надсилається на сервер, повертається відповідь	Кадр передано, JSON-відповідь отримано
5	Сканувати книгу з еталонними зображеннями	Система розпізнає книгу та додає її до чека	Книгу розпізнано, позицію додано до чека
6	Сканувати порожню область перед камерою	Товар не додається, показується повідомлення	Товар не додано, повідомлення відображено
7	Сканувати невідомий або погано розміщений об’єкт	Товар не додається, показується непідтверджений результат	Товар не додано, показано непідтверджений результат
8	Перейти до демонстраційного екрана оплати	Відображається демонстраційний екран оплати	Екран оплати відкрився

9	Завершити демонстраційний сценарій покупки	Відображається екран успішного завершення	Екран успішного завершення відображено
10	Відкрити адміністративну сторінку	Система запитує пароль адміністратора	Сторінка відкрилася з перевіркою пароля
11	Увести пароль адміністративного доступу	Після правильного пароля відкривається екран еталонів	Адміністративний екран відкрито
12	Додати новий об’єкт каталогу	Об’єкт з’являється в каталозі	Новий об’єкт додано до каталогу
13	Зберегти еталонний кадр для об’єкта	Файл зберігається, метадані записуються в SQLite	Еталон збережено, метадані синхронізовано
14	Перевірити кінцеву точку працездатності сервера	Сервер повертає відповідь про працездатність	Відповідь про працездатність отримано

ДОДАТОК Р

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Філоненко Іван Олександрович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота (проєкт) на тему «Програмне забезпечення для ідентифікації об'єктів за зображенням» 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 ШІ ChatGPT, Gemini, DeepL були використані для:
        - перекладу іншомовних джерел;
        - стилістичного редагування та перевірки граматики;
        - допомоги у написанні/відлагодженні програмного коду;

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

«22» травня 2026 р. _____ **Іван ФІЛОНЕНКО**

(підпис)