Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

Кіберфізична система «Сканер товарів» на базі Raspberry Pi
Назва теми

Рівень вищої освіти *перший (бакалаврський)*

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Шифр КвРКІ 2301115.23.01.25 ПЗ

Виконав здобувач ІІІ курсу, група КІ2с-23-1 — Підпис — Владислав МЕЛЬНИК (Ініціали, прізвище)

Керівник — к.т.н., доцент (Науковий ступінь, учене звання) — Підпис — Катерина БЕРЕЗЬКА (Ініціали, прізвище)

Нормоконтролер — д.т.н., професор (Науковий ступінь, учене звання) — Підпис — Сергій ЛИСЕНКО (Ініціали, прізвище)

До захисту допускаю:
завідувач кафедри КІІС
«01» червня 2026 р.
дата — підпис — Ольга ПАВЛОВА (Ініціали, прізвище)

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Рівень вищої освіти ПЕРШИЙ (БАКАЛАВРСЬКИЙ)

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ
Завідувачка кафедри КІІС
_____ Ольга ПАВЛОВА

" 10 " 01 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Мельнику Владиславу Віталійовичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Кіберфізична система «Сканер товарів» на базі Raspberry Pi

Керівник проекту (роботи) Березька Катерина Миколаївна, к.т.н., доц.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 7

2. Термін подання здобувачем роботи на кафедру 01.06.2026 р.

3. Вихідні дані до роботи Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Кіберфізична система «Сканер товарів» на базі Raspberry Pi та постановка задачі щодо її удосконалення

Проєктування системи обробки інформації у кіберфізичній системі «Сканер товарів» на базі Raspberry Pi

Програмно-апаратна реалізація кіберфізичної системи «Сканер товарів» на базі Raspberry Pi

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Архітектура ПЗ проєкту

Архітектура ПЗ для кіберфізичної системи

Апаратне забезпечення проєкту

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «_10_»_____01_____2026 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2026	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2026	виконано
3	Робота над розділом 1 - дослідження предметної області та постановка задачі	01.03.2026	виконано
4	Робота над розділом 2 - вибір апаратних та програмних компонентів для реалізації кіберфізичної системи «Сканер товарів» на базі Raspberry Pi	01.04.2026	виконано
5	Робота над розділом 3 - проєктування та реалізація кіберфізичної системи сканування товарів із використанням RFID-технології	29.04.2026	виконано
6	Оформлення пояснювальної записки та графічної частини згідно вимог	25.05.2026	виконано
7	Попередній захист ВКР	26.05.2026	виконано
8	Захист ВКР на засіданні ЕК	Червень 2026 року	

Здобувач _____ Владислав МЕЛЬНИК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Катерина БЕРЕЗЬКА
Підпис Ім'я, ПРІЗВИЩЕ

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 2301115.23.01.25 ПЗ	Пояснювальна записка	61		
			Графічні матеріали			
2		КвРКІ 2301115.23.01.25 Е8	Архітектура ПЗ проєкту	1		
3		КвРКІ 2301115.23.01.25 Е8	Архітектура ПЗ для	1		
			кіберфізичної системи			
4		КвРКІ 2301115.23.01.25 Е8	IDEF0-модель	1		
			функціонування			
			кіберфізичної системи			

Зм	Арк	№ докум	Підпис	Дата	КвРКІ 2301115.23.01.25 ВП		
Розробив		Мельник		03.06	Відомість проекту	Літера	Аркуш	Аркушів
Перевір.		Березька		03.06		У	1	1
Н. контр.		Лисенко		03.06		ХНУ, КІ2с-23-1		
Затв.		Павлова		03.06				

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Кіберфізична система «Сканер товарів» на базі Raspberry Pi».

Автор роботи: Владислав МЕЛЬНИК.

Керівник роботи: Катерина БЕРЕЗЬКА.

Пояснювальна записка: 61 с., 30 рис., 12 табл., 3 дод., 50 джерел.

Графічна частина: 3 креслення.

АРХІТЕКТУРА, БАЗА ДАНИХ, КІБЕРФІЗИЧНА СИСТЕМА, RASPBERRY PI, СКАНЕР ТОВАРІВ.

Кваліфікаційна робота бакалавра присвячена розробці кіберфізичної системи сканування товарів на базі одноплатного комп'ютера Raspberry Pi та технології RFID. Актуальність теми зумовлена необхідністю автоматизації процесів обліку, ідентифікації та моніторингу товарів у складських, логістичних і торговельних системах. Використання кіберфізичних систем дає змогу підвищити швидкість оброблення інформації, зменшити кількість помилок та підвищити ефективність роботи систем управління.

Метою роботи є проєктування та реалізація програмно-апаратного комплексу для автоматизованого сканування й моніторингу товарів у реальному часі. У роботі проведено аналіз RFID-технологій та кіберфізичних систем, обрано платформу Raspberry Pi, розроблено структурну схему системи, архітектуру програмного забезпечення та базу даних. Реалізовано програмні модулі для взаємодії зі зчитувачами RFID-міток, оброблення та відображення даних користувачу.

У результаті виконання роботи створено прототип кіберфізичної системи «Сканер товарів», що забезпечує автоматичне зчитування RFID-міток, ведення журналу подій та облік товарів. Результати роботи можуть бути використані для розвитку автоматизованих систем обліку та моніторингу товарів у сфері логістики й торгівлі.

Підпис здобувача

30.05.2026

Дата

ЗМІСТ

ВСТУП

Протягом останніх років автоматизація інформаційних технологій впевнено посідає важливе місце у сфері торгівлі та обслуговування споживачів. Сучасні цифрові рішення активно впроваджуються в торговельні мережі, супермаркети та роздрібні магазини з метою підвищення ефективності роботи, оптимізації бізнес-процесів та покращення якості обслуговування клієнтів. Автоматизація дозволяє забезпечити швидкий доступ до інформації, мінімізувати людський фактор та скоротити час обслуговування покупців.

Одним із актуальних напрямів розвитку є створення кіберфізичних систем, які поєднують фізичні пристрої з програмним забезпеченням та обчислювальними ресурсами. Такі системи забезпечують взаємодію між апаратною частиною (сенсорами, сканерами, обчислювальними модулями) та цифровими сервісами, що обробляють і відображають інформацію в реальному часі.

Кіберфізична система «Сканер товарів» на базі Raspberry Pi є прикладом сучасного технічного рішення для автоматизації процесу отримання інформації про товар у магазині. За допомогою сканування штрихкоду покупець може оперативно отримати дані про продукт, зокрема його назву, зображення, актуальну ціну та ціну зі знижкою (за наявності). Це підвищує зручність здійснення покупок, забезпечує прозорість ціноутворення та сприяє покращенню клієнтського досвіду.

Актуальність розробки подібних систем зумовлена зростанням вимог до швидкості обслуговування, точності інформації та цифровізації торговельного середовища. Використання одноплатного комп’ютера Raspberry Pi дозволяє реалізувати функціональний, компактний та економічно доцільний пристрій, який може бути інтегрований у сучасну інфраструктуру магазину.

1 КІБЕРФІЗИЧНА СИСТЕМА СКАНУВАННЯ ТОВАРІВ НА БАЗІ RASPBERRY PI ТА ПОСТАНОВКА ЗАДАЧІ ЩОДО ЇЇ УДОСКОНАЛЕННЯ

1.1 Аналіз структурних і функціональних особливостей кіберфізичної системи сканування товарів

Кіберфізична система (КС) - це інтегрована інформаційно-технологічна система, яка поєднує фізичні компоненти з програмними засобами та обчислювальними ресурсами для забезпечення автоматизованої взаємодії між ними. Основною особливістю кіберфізичних систем є постійний обмін даними між фізичним середовищем та програмним забезпеченням у режимі реального часу. Завдяки цьому забезпечується автоматичний збір, обробка, передача та відображення інформації без безпосереднього втручання людини.

У сучасних умовах розвитку цифрових технологій кіберфізичні системи активно використовуються у промисловості, медицині, транспорті, логістиці та сфері торгівлі. Особливо актуальним є їх застосування у торговельних закладах, де важливими є швидкість обслуговування, точність обробки інформації та зручність взаємодії покупця із системою.

Одним із прикладів такої системи є кіберфізична система «Сканер товарів», призначена для автоматизованого отримання інформації про товари за допомогою технології штрихового кодування. Система забезпечує зчитування штрих-коду, обробку отриманих даних, пошук інформації у базі даних та відображення результату користувачу.

Основою функціонування системи є технологія автоматичної ідентифікації товарів за допомогою штрихових кодів. Штрих-код являє собою графічне представлення цифрової інформації у вигляді послідовності темних та світлих смуг різної товщини. Кожен код містить унікальний ідентифікатор товару, який використовується для пошуку інформації у базі даних.

Приклад сучасного штрихового кодування товару наведено на рисунку 1.1

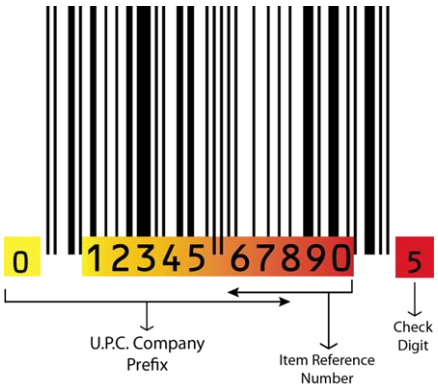


Рисунок 1.1 - Приклад сучасного штрихового кодування товару [1]

Сучасні штрихові коди містять інформацію про країну виробника, код компанії, ідентифікатор товару та контрольну цифру. Використання штрихового кодування дозволяє значно прискорити процес обліку товарів, зменшити кількість помилок під час введення даних та автоматизувати процеси взаємодії між обладнанням і програмним забезпеченням.

Кіберфізична система «Сканер товарів» складається із взаємопов’язаних апаратних та програмних компонентів, які забезпечують повний цикл обробки інформації від моменту сканування товару до відображення результату на дисплеї.

До складу системи входять такі основні елементи:

1. Сканер штрих-коду.
2. Одноплатний комп’ютер Raspberry Pi.
3. Дисплей для відображення інформації.
4. База даних товарів.
5. Користувацький інтерфейс.
6. Комунікаційні інтерфейси та периферійні пристрої.

Структурну схему кіберфізичної системи показано на рисунку 1.2.

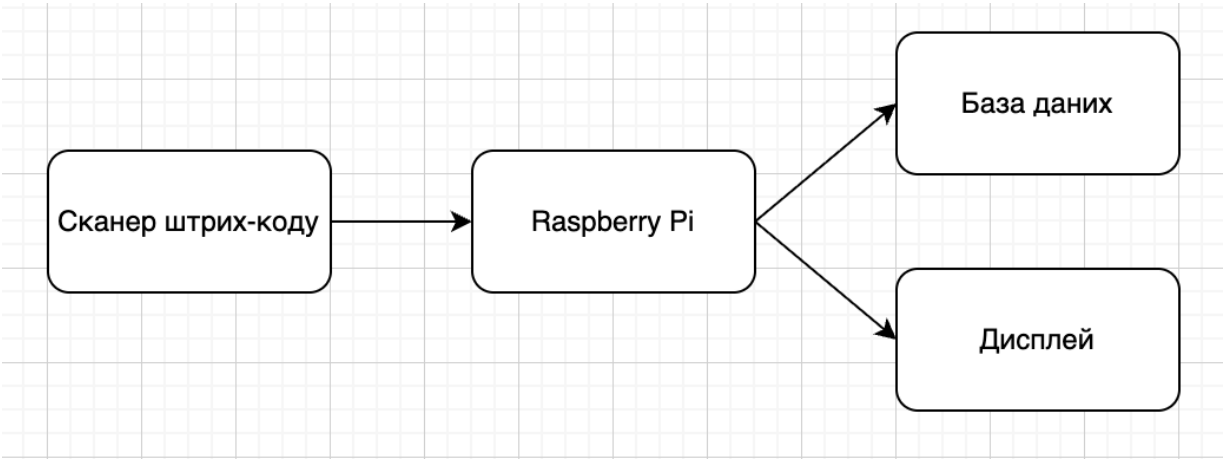


Рисунок 1.2 - Структурна схема кіберфізичної системи «Сканер товарів»

Як видно зі структурної схеми, центральним елементом системи є Raspberry Pi, який виконує функції обробки даних та координації роботи всіх компонентів. Після зчитування штрих-коду сканером інформація передається до Raspberry Pi, де виконується її обробка та пошук відповідного запису у базі даних. Після цього результати передаються на дисплей для відображення користувачу.

Основні компоненти кіберфізичної системи «Сканер товарів» наведено у таблиці 1.1.

Таблиця 1.1 - Основні компоненти кіберфізичної системи «Сканер товарів»

Компонент	Призначення
Сканер штрих-коду	Зчитування та декодування штрих-коду товару
Raspberry Pi	Обробка даних та керування системою
База даних	Зберігання інформації про товари
Дисплей	Відображення інформації користувачу
Інтерфейс користувача	Забезпечення взаємодії із системою
Комунікаційні інтерфейси	Передача даних між компонентами

Сканер штрих-коду є одним із ключових елементів системи, оскільки саме він забезпечує отримання первинної інформації про товар. Залежно від типу пристрою можуть використовуватися лазерні, світлодіодні або CMOS-сканери. У більшості випадків підключення до Raspberry Pi виконується через USB-інтерфейс, що значно спрощує інтеграцію обладнання.

Одноплатний комп'ютер Raspberry Pi виконує роль центрального обчислювального вузла системи. Він забезпечує:

- приймання даних від сканера;
- обробку отриманої інформації;
- виконання пошуку у базі даних;
- керування відображенням інформації;
- взаємодію між програмними модулями.

Кіберфізична система «Сканер товарів» виконує ряд функцій, спрямованих на автоматизацію процесу отримання інформації про товари. Основні функції наведені у таблиці 1.2.

Таблиця 1.2 - Функції кіберфізичної системи «Сканер товарів»

№	Функція	Опис
1	Зчитування штрих-кодів	Отримання ідентифікатора товару
2	Обробка даних	Передача та аналіз отриманих даних
3	Пошук у базі даних	Визначення товару за кодом
4	Відображення інформації	Виведення назви, ціни та зображення
5	Оновлення інформації	Можливість зміни даних у базі
6	Контроль помилок	Обробка некоректних або відсутніх даних

Важливою особливістю системи є можливість функціонування у режимі реального часу. Це означає, що користувач отримує інформацію практично миттєво після сканування товару. Такий підхід дозволяє підвищити швидкість обслуговування покупців та зменшити навантаження на персонал магазину.

Принцип роботи кіберфізичної системи базується на послідовному виконанні операцій обробки інформації:

1. Користувач підносить товар до сканера.
2. Сканер зчитує штрих-код.
3. Дані передаються до Raspberry Pi.
4. Виконується пошук у базі даних.
5. Результат відображається на дисплеї.

Схему алгоритму роботи системи наведено на рисунку 1.3.

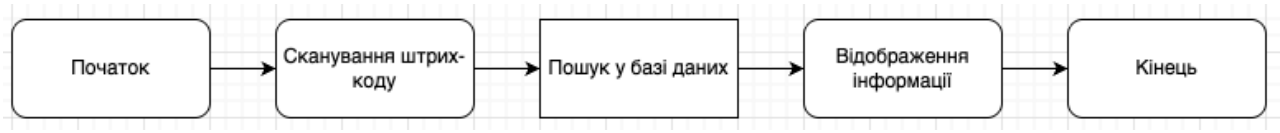


Рисунок 1.3 - Алгоритм роботи системи сканування товарів

Алгоритм роботи системи є достатньо простим, проте забезпечує ефективну взаємодію між усіма компонентами. Після сканування товару система автоматично виконує перевірку наявності запису у базі даних. У разі успішного пошуку користувачу відображаються назва товару, ціна, акційна пропозиція та інша додаткова інформація. Якщо товар відсутній у базі, система формує відповідне повідомлення про помилку.

До апаратної частини системи належать:

1. Raspberry Pi як центральний обчислювальний модуль.
2. Сканер штрих-коду.
3. Дисплей для відображення інформації.
4. Джерело живлення.
5. Засоби комунікації та периферійні пристрої.

Використання Raspberry Pi є доцільним завдяки його компактності, низькому енергоспоживанню та підтримці великої кількості інтерфейсів підключення. Плата підтримує USB, HDMI, GPIO, Wi-Fi та Bluetooth, що дозволяє легко інтегрувати додаткові пристрої у систему.

Зовнішній вигляд Raspberry Pi наведено на рисунку 1.4.

Рисунок 1.4 - Зовнішній вигляд Raspberry Pi

Однією з переваг Raspberry Pi є можливість використання операційної системи Android та широкого набору програмних бібліотек. Це дозволяє реалізувати функції роботи зі сканером, базами даних, графічним інтерфейсом та мережевими сервісами без необхідності використання дорогого спеціалізованого обладнання.

Програмна частина системи забезпечує:

- обробку сигналів зі сканера;
- декодування штрих-кодів;
- взаємодію з базою даних;
- відображення інформації користувачу;
- обробку помилок;
- керування інтерфейсом користувача.

Для реалізації програмного забезпечення можуть використовуватися мови програмування Python, Kotlin або Java, а також відповідні бібліотеки для роботи з графічним інтерфейсом та мережевими протоколами. База даних використовується для централізованого зберігання інформації про товари, зокрема:

- назви товару;

- ціни;
- акційної вартості;
- штрих-коду;
- зображення товару;
- додаткових характеристик.

Незважаючи на широке використання автоматизованих систем у сфері торгівлі, існує ряд проблем, які залишаються актуальними:

- відсутність можливості швидкої перевірки ціни товару у торговельному залі;
- необхідність звернення до персоналу;
- розбіжність між цінами на цінниках і в касовій системі;
- недостатній рівень інформативності для покупця;
- черги біля інформаційних терміналів та кас.

Упровадження кіберфізичної системи «Сканер товарів» дозволяє вирішити значну частину зазначених проблем шляхом автоматизації процесу отримання інформації про товари. Покупець отримує можливість самостійно перевіряти актуальні дані про товар без допомоги працівників магазину, що підвищує рівень комфорту та швидкість обслуговування.

У результаті проведеного аналізу встановлено, що кіберфізична система «Сканер товарів» є інтегрованим програмно-апаратним комплексом, який забезпечує автоматизоване отримання інформації про товар у режимі реального часу. Визначено основні структурні компоненти системи, їх функціональне призначення та принцип взаємодії між ними. Також проаналізовано принцип роботи системи, її програмні та апаратні складові, а також обґрунтовано доцільність використання Raspberry Pi як центрального елемента обчислювальної частини.

Отримані результати створюють основу для подальшого аналізу існуючих рішень, вибору програмно-апаратних компонентів та проєктування кіберфізичної системи «Сканер товарів».

1.2 Аналіз програмно-апаратного забезпечення обробки інформації в кіберфізичній системі «Сканер товарів» на базі Raspberry Pi

У сучасних кіберфізичних системах процес обробки інформації є одним із ключових етапів функціонування, оскільки саме він визначає швидкодію, надійність та ефективність роботи всієї системи. Для кіберфізичної системи «Сканер товарів» особливого значення набуває оперативна взаємодія між апаратними компонентами, програмним забезпеченням та базою даних товарів.

Система повинна забезпечувати автоматизоване зчитування штрих-кодів, обробку отриманих даних, формування запитів до бази даних та відображення результатів користувачу у режимі реального часу. Для реалізації таких функцій необхідне поєднання сучасних апаратних засобів та програмних технологій.

На відміну від складних промислових кіберфізичних систем, системи у сфері роздрібної торгівлі орієнтовані насамперед на швидкість взаємодії з користувачем, простоту експлуатації та економічну доцільність. Більшість існуючих рішень реалізуються у вигляді окремих програмних модулів або POS-систем, які виконують лише частину необхідних функцій. У результаті виникає потреба у створенні комплексної програмно-апаратної системи, що об’єднує всі компоненти в єдине інформаційне середовище.

Кіберфізична система «Сканер товарів» побудована за принципом взаємодії між фізичними пристроями та програмними сервісами. Основною апаратною платформою є Raspberry Pi, який виконує функції центрального обчислювального модуля та координує роботу інших компонентів системи.

Одноплатний комп’ютер Raspberry Pi показано на рисунку 1.5.

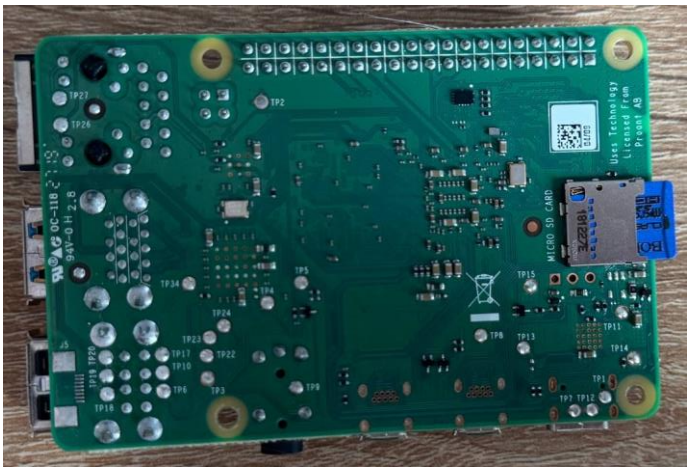


Рисунок 1.5 - Одноплатний комп’ютер Raspberry Pi

Raspberry Pi є компактним одноплатним комп’ютером, що поєднує достатню обчислювальну потужність, низьке енергоспоживання та підтримку великої кількості інтерфейсів підключення. Завдяки цьому платформа широко використовується у робототехніці, системах автоматизації, IoT-рішеннях та кіберфізичних системах.

До основних переваг Raspberry Pi належать:

- компактні розміри;
- доступна вартість;
- підтримка Linux та Android;
- наявність USB, GPIO, HDMI, Wi-Fi та Bluetooth;
- можливість інтеграції периферійних пристроїв;
- низьке енергоспоживання;
- підтримка мережевих технологій.

У складі кіберфізичної системи Raspberry Pi забезпечує:

- приймання даних зі сканера штрих-коду;
- обробку інформації;
- формування HTTP-запитів;
- взаємодію з сервером та базою даних;
- відображення інформації на дисплеї;

- виконання логіки роботи користувацького інтерфейсу.

До складу апаратного забезпечення системи входять:

1. Raspberry Pi.
2. Сканер штрих-коду.
3. Дисплей (LCD або HDMI).
4. Блок живлення.
5. Інтерфейси підключення.
6. Комунікаційні модулі.

Основні апаратні компоненти наведені у таблиці 1.3.

Таблиця 1.3 - Основні апаратні компоненти системи

Компонент	Характеристика	Призначення
Raspberry Pi	Одноплатний комп’ютер	Обробка даних
Сканер штрих-коду	Лазерний або CCD	Зчитування інформації
Дисплей	LCD/HDMI	Виведення інформації
Блок живлення	5V	Живлення системи
Інтерфейси	USB, GPIO	Передача даних

Сканер штрих-коду забезпечує отримання цифрового ідентифікатора товару. У більшості випадків використовується HID-режим роботи, у якому сканер функціонує як стандартна клавіатура та передає код у систему через емуляцію натискань клавіш. Такий підхід дозволяє спростити інтеграцію обладнання та уникнути необхідності використання спеціалізованих драйверів.

Дисплей використовується для відображення інформації про товар. Після сканування користувач може отримати:

- назву товару;
- поточну ціну;
- акційну пропозицію;
- фотографію товару;

- службові повідомлення системи.

Важливу роль у роботі системи відіграє програмне забезпечення, яке забезпечує логіку функціонування всіх компонентів та організовує інформаційні потоки між ними. Програмна складова виконує:

- отримання даних зі сканера;
- декодування штрих-кодів;
- формування запитів до серверного API;
- обробку JSON-відповідей;
- взаємодію з базою даних;
- відображення інформації користувачу.

Для реалізації клієнтської частини системи доцільно використовувати мову програмування Kotlin та платформу Android. Kotlin є сучасною мовою програмування, яка забезпечує високу стабільність, зручність підтримки коду та сумісність із Android SDK.

Основні програмні компоненти системи наведені у таблиці 1.4.

Таблиця 1.4 - Основні програмні компоненти системи

Компонент	Технологія/Інструмент	Призначення
Мова програмування	Kotlin	Реалізація клієнтського застосунку
Платформа	Android SDK	Робота застосунку на Raspberry Pi
Мережевий модуль	OkHttp	HTTP-запити до серверного API
Обробка зображень	Glide	Завантаження та кешування зображень
UI бібліотеки	Material Components, ConstraintLayout	Побудова інтерфейсу користувача

Кінець таблиці 1.4

База даних	PostgreSQL	Зберігання інформації про товари
Сервер API	REST API	Надання даних клієнтському застосунку
Формат даних	JSON	Передача даних між компонентами
Система введення	HID	Зчитування штрих-коду

Для взаємодії між клієнтською частиною та сервером використовується REST API. Передача даних виконується через HTTP-запити із використанням бібліотеки OkHttp. Серверна частина обробляє запити, виконує пошук інформації у базі PostgreSQL та повертає результат у форматі JSON.

Схему обробки інформації у системі наведено на рисунку 1.6.

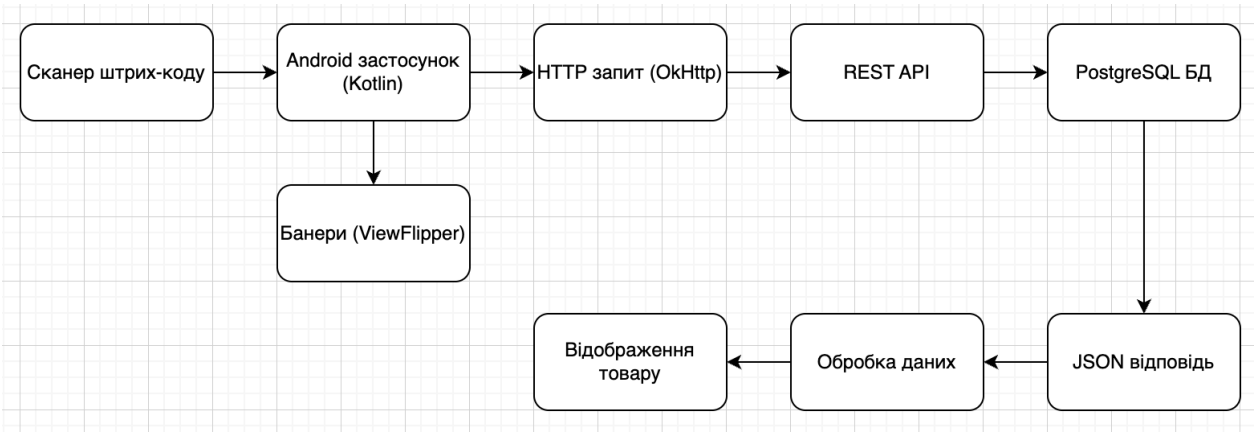


Рисунок 1.6 - Схема обробки інформації в системі

Як видно зі схеми, після сканування штрих-коду Android-застосунок отримує код товару та формує HTTP-запит до REST API. Сервер виконує звернення до бази даних PostgreSQL, після чого формує JSON-відповідь із даними про товар. Отримана інформація обробляється клієнтським застосунком та відображається користувачу у графічному інтерфейсі.

Важливим компонентом інтерфейсу є система банерів, реалізована за допомогою ViewFlipper. Вона дозволяє відображати рекламні матеріали, акційні пропозиції або службову інформацію у режимі автоматичної зміни слайдів.

Процес обробки інформації в системі включає декілька основних етапів:

1. Зчитування штрих-коду товару.
2. Передача даних до Raspberry Pi.
3. Формування HTTP-запиту.
4. Передача запиту до REST API.
5. Пошук інформації у базі PostgreSQL.
6. Формування JSON-відповіді.
7. Обробка результату застосунком.
8. Відображення інформації користувачу.

У сучасних умовах існує декілька підходів до реалізації систем сканування товарів. Основні з них наведено у таблиці 1.5.

Таблиця 1.5 - Порівняння підходів до реалізації систем сканування

Тип системи	Переваги	Недоліки
Касові POS-системи	Висока точність та інтеграція	Висока вартість
Мобільні застосунки	Доступність та мобільність	Залежність від смартфона
DIY системи на Raspberry Pi	Низька вартість, гнучкість	Потреба в налаштуванні

Порівняльний аналіз показує, що використання Raspberry Pi дозволяє створити недороге та гнучке рішення, яке можна адаптувати до потреб конкретного торговельного закладу. Крім того, така система може бути модернізована шляхом додавання нових функцій або інтеграції додаткових пристроїв.

До основних переваг використання Raspberry Pi у складі кіберфізичної системи належать:

- можливість швидкої розробки прототипу;
- підтримка сучасних програмних технологій;
- простота інтеграції зі сканерами та дисплеями;
- підтримка мережевих сервісів;
- можливість роботи з базами даних;
- масштабованість системи.

Разом із тим аналіз існуючих рішень дозволив виділити ряд проблем:

- відсутність універсальних інтегрованих систем;
- обмежені можливості бюджетних рішень;
- складність впровадження промислових систем;
- висока вартість професійного обладнання;
- потреба в адаптації програмного забезпечення під конкретні задачі.

Таким чином, проведений аналіз показав, що використання Raspberry Pi у поєднанні із сучасними програмними технологіями дозволяє створити ефективну кіберфізичну систему сканування товарів. Визначено основні апаратні та програмні компоненти системи, проаналізовано принципи їх взаємодії та особливості обробки інформації. Отримані результати є основою для подальшого проєктування архітектури системи та реалізації програмно-апаратного комплексу «Сканер товарів».

1.3 Постановка задачі дослідження кіберфізичної системи «Сканер товарів»

У результаті проведеного аналізу предметної області, структурних і функціональних особливостей кіберфізичних систем, а також сучасного програмно-апаратного забезпечення встановлено, що існуючі рішення у сфері

автоматизації торгівлі не повністю задовольняють потреби користувачів щодо швидкого, зручного та автономного отримання інформації про товари.

Як було розглянуто у попередніх підрозділах, більшість сучасних систем сканування товарів реалізуються у вигляді касових терміналів самообслуговування або мобільних застосунків. Такі рішення мають низку обмежень. Касові комплекси орієнтовані переважно на завершальний етап покупки та не забезпечують покупцю можливості оперативно отримати інформацію про товар безпосередньо у торговельному залі. Мобільні застосунки, у свою чергу, потребують встановлення програмного забезпечення на смартфон користувача, стабільного доступу до мережі Інтернет та достатнього рівня технічної підготовки користувача.

Крім того, у багатьох торговельних закладах покупці стикаються з такими проблемами:

- відсутність актуальної інформації про ціну товару;
- невідповідність цінників даним у касовій системі;
- неможливість швидко перевірити акційні пропозиції;
- складність пошуку характеристик товару;
- перевантаження працівників торговельного залу через постійні запити покупців.

Усе це знижує ефективність обслуговування клієнтів та негативно впливає на рівень автоматизації торговельного процесу. Саме тому виникає необхідність створення окремої кіберфізичної системи, яка забезпечуватиме автоматизоване зчитування інформації про товар без участі персоналу магазину.

Кіберфізична система «Сканер товарів» повинна поєднувати апаратні та програмні компоненти в єдине інтегроване середовище. Основною особливістю такої системи є взаємодія фізичних пристроїв зі програмними сервісами у режимі реального часу. До фізичної складової належать Raspberry Pi, сканер штрих-кодів, дисплей та периферійні пристрої, а до програмної модулі обробки даних, база даних товарів та інтерфейс користувача.

Розроблювана система повинна забезпечувати швидке отримання інформації після сканування штрих-коду товару, що дозволить підвищити рівень обслуговування покупців та зменшити навантаження на персонал торговельного закладу. Також важливою перевагою системи є її автономність та можливість інтеграції у вже існуючу інфраструктуру магазину.

Метою даної роботи є розробка та дослідження кіберфізичної системи «Сканер товарів» на базі Raspberry Pi, яка забезпечує автоматизоване зчитування штрих-кодів та відображення актуальної інформації про товар із використанням сучасних інформаційних технологій.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати існуючі рішення у сфері автоматизації торгівлі.
2. Дослідити принципи побудови кіберфізичних систем та особливості їх застосування у торговельній сфері.
3. Визначити основні функціональні вимоги до системи сканування товарів.
4. Обґрунтувати вибір апаратної платформи Raspberry Pi для реалізації системи.
5. Визначити необхідні апаратні та програмні компоненти системи.
6. Розробити архітектуру кіберфізичної системи.
7. Реалізувати механізм зчитування та обробки штрих-кодів.
8. Забезпечити взаємодію системи з базою даних товарів.
9. Розробити інтерфейс користувача для відображення інформації.
10. Провести тестування системи та оцінити ефективність її роботи.

У процесі розробки система повинна забезпечувати виконання таких основних функцій:

- зчитування штрих-коду товару за допомогою сканера;
- передачу отриманих даних до обчислювального модуля;
- обробку та аналіз отриманої інформації;
- пошук відповідного товару у базі даних;

- відображення інформації про товар на дисплеї;
- виведення повідомлень про помилки або відсутність товару;
- підтримку безперервної роботи у торговельному залі.

Окрім функціональних вимог, важливе значення мають нефункціональні характеристики системи. Вони визначають стабільність, продуктивність та зручність використання програмно-апаратного комплексу.

Інформація, що повинна відображатися користувачу після сканування товару, наведена у таблиці 1.6.

Таблиця 1.6 - Інформація, що відображається системою

Параметр	Опис
Назва товару	Повна назва товару
Ціна	Поточна вартість товару
Акційна ціна	Ціна зі знижкою за наявності акції
Зображення товару	Фотографія або графічне представлення
Статус наявності	Інформація про наявність товару в базі

До нефункціональних вимог системи належать:

- висока швидкодія обробки запитів;
- надійність роботи у режимі безперервного функціонування;
- зручність користування для покупців різного рівня підготовки;
- простота масштабування системи;
- енергоефективність апаратної платформи;
- сумісність з Raspberry Pi;
- можливість модернізації програмного забезпечення;
- стабільна робота при підключенні периферійних пристроїв.

Під час реалізації системи необхідно також враховувати особливості роботи кіберфізичних систем, зокрема:

- взаємодію фізичних пристроїв із програмним забезпеченням;
- обмін даними у режимі реального часу;
- обробку подій від апаратних компонентів;
- забезпечення стабільності роботи при тривалому навантаженні.

Очікується, що розроблена система дозволить підвищити рівень автоматизації торговельного процесу та покращити взаємодію покупців із торговельним обладнанням. Використання Raspberry Pi як центрального обчислювального модуля забезпечить компактність, доступність та енергоефективність системи, а застосування сучасних програмних засобів дозволить реалізувати зручний користувацький інтерфейс.

У результаті виконання роботи планується отримати:

- функціонуючу кіберфізичну систему сканування товарів;
- інтегроване програмно-апаратне рішення;
- систему, придатну для використання у реальних умовах торговельних закладів;
- програмне забезпечення для автоматизованого пошуку та відображення інформації про товари;
- результати тестування ефективності та стабільності роботи системи.

1.4 Висновки до першого розділу

У першому розділі кваліфікаційної роботи проведено комплексний аналіз теоретичних та практичних аспектів побудови кіберфізичної системи «Сканер товарів» на базі Raspberry Pi. Розглянуто сучасні підходи до створення кіберфізичних систем, визначено їх основні структурні компоненти, принципи функціонування та особливості інтеграції програмних і апаратних засобів у єдине інформаційне середовище.

У процесі дослідження встановлено, що кіберфізичні системи є одним із перспективних напрямів розвитку сучасних інформаційних технологій, оскільки

забезпечують тісну взаємодію між фізичними пристроями, програмним забезпеченням та користувачем у режимі реального часу. Особливу увагу приділено застосуванню таких систем у сфері роздрібної торгівлі, де автоматизація процесів обслуговування дозволяє підвищити швидкість обробки інформації, покращити якість обслуговування покупців та оптимізувати роботу персоналу.

У межах розділу також було проведено аналіз існуючих рішень для сканування та обробки товарної інформації. Встановлено, що більшість сучасних систем орієнтовані або на використання касових терміналів, або на мобільні програмні застосунки. Такі рішення мають певні переваги, проте характеризуються низкою недоліків, серед яких залежність від додаткового обладнання, обмежена доступність для користувачів, складність інтеграції та недостатній рівень автономності.

Проведений аналіз дозволив визначити основні проблеми, які виникають під час отримання покупцями інформації про товари у торговельному залі. До таких проблем належать:

- затримки під час пошуку актуальної ціни товару;
- відсутність швидкого доступу до акційної інформації;
- необхідність звернення до працівників магазину;
- перевантаження касових зон;

На основі отриманих результатів було обґрунтовано доцільність створення власної кіберфізичної системи «Сканер товарів», яка забезпечуватиме автоматизоване зчитування штрих-кодів та оперативне отримання інформації про товар безпосередньо у торговельному залі.

Окрему увагу приділено аналізу програмно-апаратного забезпечення систем обробки інформації. У результаті дослідження визначено доцільність використання:

- мови програмування Kotlin для створення сучасного та стабільного програмного забезпечення;

- платформи Android для реалізації інтерфейсу користувача;
- мережевих технологій OkHttp та REST API для організації обміну даними між компонентами системи;
- системи керування базами даних PostgreSQL для зберігання та обробки інформації про товари;
- апаратної платформи Raspberry Pi як центрального обчислювального модуля системи.

Використання Raspberry Pi дозволяє реалізувати компактне, енергоефективне та доступне рішення, яке може бути інтегроване у реальні умови функціонування торговельних закладів. Крім того, платформа забезпечує достатній рівень продуктивності для виконання операцій зі зчитування, обробки та передачі даних у режимі реального часу.

У результаті проведеного аналізу сформульовано постановку задачі розробки кіберфізичної системи «Сканер товарів». Визначено основну мету роботи, перелік ключових завдань, а також функціональні й нефункціональні вимоги до системи. Зокрема, встановлено, що система повинна забезпечувати:

- автоматизоване зчитування штрих-кодів;
- обробку отриманих даних;
- взаємодію з базою даних товарів;
- відображення інформації про товар у зручному для користувача вигляді;

Таким чином, результати першого розділу створюють теоретичну та практичну основу для подальшого проєктування й реалізації кіберфізичної системи «Сканер товарів». Отримані висновки та визначені вимоги будуть використані у наступних розділах роботи під час розробки архітектури системи, вибору апаратних компонентів, створення програмного забезпечення та проведення експериментального дослідження ефективності функціонування системи.

2 ПРОЄКТУВАННЯ КІБЕРФІЗИЧНОЇ СИСТЕМИ «СКАНЕР ТОВАРІВ»

2.1 Визначення апаратних і програмних підсистем програмно-технічного засобу

Аналіз, проведений у першому розділі, дозволяє визначити основні підсистеми кіберфізичної системи «Сканер товарів» на базі Raspberry Pi. Дана система є програмно-технічним комплексом, що поєднує апаратні засоби, програмне забезпечення та інформаційні ресурси для забезпечення автоматизованого отримання інформації про товар.

На відміну від складних розподілених систем, розглянута кіберфізична система має чітко визначену архітектуру, яка складається з трьох основних підсистем:

- апаратна підсистема;
- програмна підсистема клієнтського рівня;
- серверна підсистема обробки даних.

Структуру кіберфізичної системи «Сканер товарів» надано на рисунку 2.1.

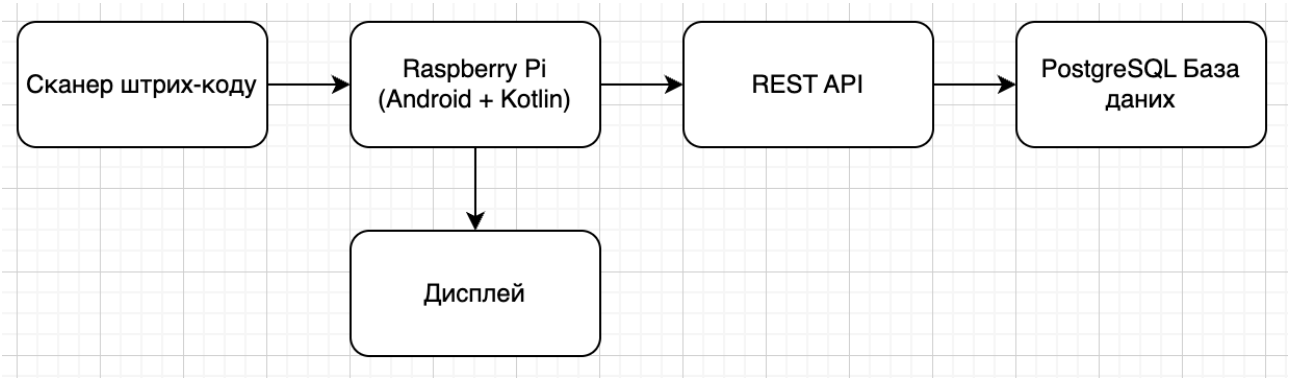


Рисунок 2.1 - Структура кіберфізичної системи «Сканер товарів»

Апаратна підсистема забезпечує фізичну взаємодію користувача з системою та реалізує процес зчитування інформації.

До її складу входять:

1. Одноплатний комп’ютер Raspberry Pi.
2. Сканер штрих-кодів (USB HID).
3. Дисплей для відображення інформації.
4. Джерело живлення.
5. Комунікаційні інтерфейси (USB, HDMI, GPIO).

Основною функцією апаратної підсистеми є зчитування штрих-коду та передача даних до програмної частини системи.

Клієнтська частина системи реалізована у вигляді Android-застосунку, розробленого на мові програмування Kotlin. Вона забезпечує обробку вхідних даних, взаємодію з сервером та відображення інформації користувачу.

Основні компоненти програмної підсистеми:

- модуль обробки введення (зчитування штрих-коду через HID);
- модуль мережевої взаємодії (HTTP-запити через OkHttp);
- модуль обробки даних (парсинг JSON);
- модуль відображення (інтерфейс користувача);
- модуль відображення банерів.

Згідно з архітектурою застосунку, центральним елементом є компонент MainActivity, який координує всі процеси взаємодії між модулями системи.

Серверна підсистема відповідає за зберігання та обробку інформації про товари.

До її складу входять:

- REST API сервер;
- база даних PostgreSQL;
- система обробки запитів.

Сервер забезпечує обробку запитів клієнтського застосунку та формує відповідь у форматі JSON, яка містить необхідну інформацію про товар.

Взаємодія між підсистемами здійснюється за наступним принципом:

1. Сканер зчитує штрих-код.
2. Дані передаються до клієнтського застосунку;.

3. Формується HTTP-запит до серверного API.
4. Сервер звертається до бази даних PostgreSQL.
5. Формується відповідь у форматі JSON.
6. Дані обробляються клієнтським застосунком.
7. Інформація відображається на дисплеї.

Основні підсистеми кіберфізичної системи наведені у таблиці 2.1.

Таблиця 2.1 - Основні підсистеми кіберфізичної системи

Підсистема	Склад	Функції
Апаратна	Raspberry Pi, сканер, дисплей	Зчитування та передача даних
Клієнтська	Android, Kotlin	Обробка та відображення
Серверна	API, PostgreSQL	Зберігання та обробка

У результаті проєктування визначено основні підсистеми кіберфізичної системи «Сканер товарів» та встановлено їх функціональне призначення. Виділення апаратної, клієнтської та серверної підсистем дозволяє чітко структуризувати процес обробки інформації та забезпечити ефективну взаємодію компонентів системи.

Отримані результати є основою для подальшого проєктування архітектури системи та реалізації її окремих модулів.

2.2 Розробка архітектури кіберфізичної системи «Сканер товарів»

Розробка архітектури кіберфізичної системи «Сканер товарів» є одним із ключових етапів проєктування, оскільки саме архітектура визначає структуру системи, принципи взаємодії її компонентів та ефективність обробки інформації. На основі аналізу, проведеного у першому розділі, встановлено, що доцільним є

використання клієнт-серверної моделі, яка дозволяє розділити функції між пристроєм сканування та серверною частиною.

У загальному вигляді архітектура системи складається з трьох логічних рівнів: рівня представлення, рівня обробки даних та рівня зберігання інформації.

Такий підхід забезпечує гнучкість системи, можливість масштабування та спрощує її модернізацію.

Загальну архітектуру кіберфізичної системи «Сканер товарів» висвітлено на рисунку 2.2.

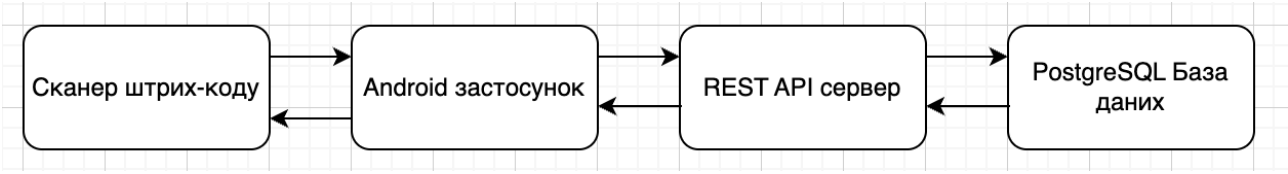


Рисунок 2.2 - Загальна архітектура кіберфізичної системи «Сканер товарів»

Рівень представлення реалізований у вигляді клієнтського застосунку, що працює на пристрої під керуванням операційної системи Android. Центральним елементом цього рівня є компонент MainActivity, який виконує функцію координації взаємодії між користувачем, сканером та іншими програмними модулями системи. Відображення інформації про товар здійснюється за допомогою фрагмента ProductFragment, який формує інтерфейс користувача з урахуванням отриманих даних. Окрім цього, у системі реалізовано механізм відображення рекламного контенту у вигляді банерів, що циклічно змінюються на екрані.

Згідно з технічним описом архітектури застосунку, управління станом системи зосереджене у межах одного Activity, що спрощує реалізацію та забезпечує достатню продуктивність для задач даного типу.

Рівень обробки даних відповідає за реалізацію бізнес-логіки системи. Саме на цьому рівні відбувається прийом вхідних даних від сканера, їх обробка та формування запитів до серверної частини. Сканер штрих-кодів функціонує у

режимі емуляції клавіатури (HID), що дозволяє передавати дані без використання спеціалізованих драйверів. Після завершення введення штрих-коду формується HTTP-запит до серверного API, який виконується за допомогою бібліотеки OKHTTP.

Отримані від сервера дані мають формат JSON і містять інформацію про товар, включаючи назву, ціну, акційну ціну та зображення. Подальша обробка передбачає парсинг цих даних та їх підготовку до відображення у користувацькому інтерфейсі.

Логіка обробки даних у клієнтському застосунку наведена на рисунку 2.3.

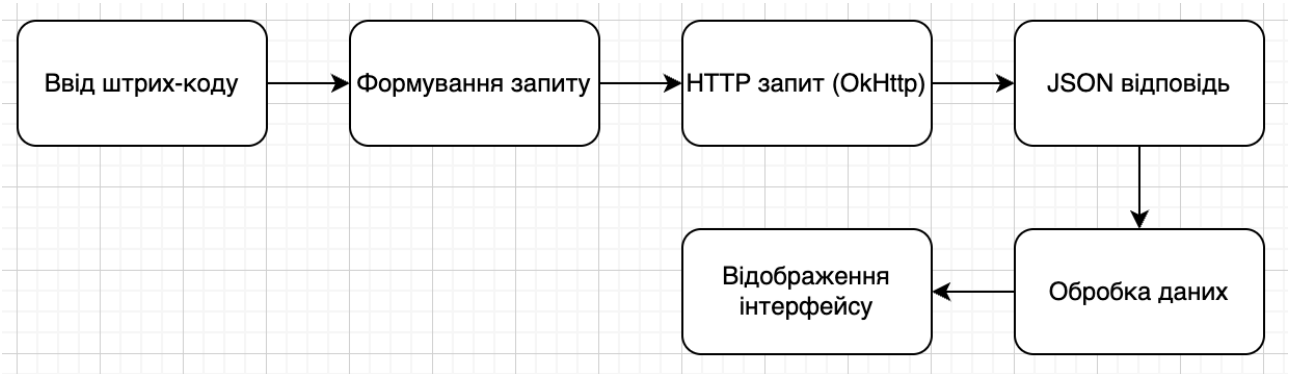


Рисунок 2.3 - Логіка обробки даних у клієнтському застосунку

Рівень зберігання даних реалізований на серверній стороні та включає REST API і базу даних PostgreSQL. Серверна частина забезпечує прийом запитів від клієнтського застосунку, їх обробку та взаємодію з базою даних. Основною функцією цього рівня є зберігання актуальної інформації про товари, включаючи ціни, акційні пропозиції та мультимедійні ресурси.

Звернення до серверної частини здійснюється за допомогою REST API, яке забезпечує доступ до даних через HTTP-запити. Зокрема, для отримання інформації про товар використовується запит виду:

```
/api/v2/products/{barcode}
```

а для отримання рекламного контенту запит:

```
/api/v3/blog/articles
```

що дозволяє динамічно оновлювати інформацію без необхідності зміни клієнтського застосунку.

Взаємодія між рівнями архітектури відбувається послідовно: після зчитування штрих-коду клієнтський застосунок формує запит до серверної частини, яка, у свою чергу, звертається до бази даних. Після обробки запиту сервер повертає результат у форматі JSON, який обробляється клієнтською частиною та відображається користувачу.

Розподіл функцій між рівнями архітектури наведено у таблиці 2.2.

Таблиця 2.2 - Розподіл функцій між рівнями архітектури

Рівень	Основні функції
Представлення	Взаємодія з користувачем, відображення інформації
Обробка	Формування запитів, обробка даних
Дані	Зберігання та надання інформації

Обрана архітектура характеризується простотою реалізації та достатньою ефективністю для задач обслуговування користувачів у торговельному середовищі. Використання клієнт-серверної моделі дозволяє централізувати дані та забезпечити їх актуальність, а також спрощує оновлення системи без внесення змін у клієнтську частину.

Разом з тим, така архітектура має певні обмеження, пов’язані із залежністю від мережевого з’єднання та відсутністю локального кешування даних. Проте для задач, що розглядаються у даній роботі, ці обмеження не є критичними.

У результаті виконаного проєктування визначено архітектуру кіберфізичної системи «Сканер товарів», яка базується на клієнт-серверній моделі. Виділено основні рівні системи, встановлено їх функціональне призначення та описано взаємодію між компонентами.

Запропонована архітектура забезпечує ефективну обробку інформації, простоту реалізації та можливість подальшого розвитку системи.

2.3 Проєктування бази даних кіберфізичної системи «Сканер товарів»

Одним із ключових елементів кіберфізичної системи «Сканер товарів» є база даних, оскільки саме вона забезпечує зберігання актуальної інформації про товари, ціни, акційні пропозиції, зображення та додаткові характеристики. У розроблюваній системі база даних використовується серверною частиною, яка отримує запит від Android-застосунку, виконує пошук товару за штрих-кодом і повертає результат у форматі JSON.

Для реалізації зберігання даних доцільно використати PostgreSQL, оскільки ця система керування базами даних підтримує роботу зі структурованими даними, індексацію, обмеження цілісності та тип jsonb, що є корисним для зберігання додаткових штрих-кодів товару. У межах даної системи основною таблицею є shop_products, яка містить усі необхідні поля для ідентифікації товару, відображення його ціни та перевірки наявності акційної пропозиції.

ER-діаграма таблиці товарів shop_products наведена на рисунку 2.4.

Рисунок 2.4 - ER-діаграма таблиці товарів shop_products

Основна логіка роботи бази даних полягає у тому, що кожний товар має унікальний ідентифікатор id, внутрішній артикул sku та основний штрих-код barcode. Саме поле barcode використовується під час пошуку товару після сканування. Якщо користувач підносить товар до сканера, застосунок отримує штрих-код, передає його на сервер, а серверна частина виконує запит до таблиці shop_products.

Для збереження базової інформації про товар використовуються поля title, category, price, description, status та image. Поле title містить назву товару, яка відображається на екрані користувача. Поле price зберігає основну ціну, а image містить шлях або URL до зображення товару. Статус товару зберігається у полі status, що дозволяє контролювати доступність товару для відображення у системі.

Для підтримки акційних пропозицій у таблиці передбачено групу полів, які описують акцію: promo_title, promo_start_at, promo_end_at, promo_discount_percent, promo_discount, promo_promo_price, promo_type та promo_price_type. Завдяки цьому система може відображати не лише основну ціну товару, а й акційну ціну, якщо вона діє на момент сканування.

Структура основної таблиці бази даних shop_products описана у таблиці 2.3.

Таблиця 2.3 - Структура основної таблиці бази даних shop_products

Поле	Тип даних	Призначення
id	uuid	Унікальний ідентифікатор товару
sku	varchar (255)	Артикул товару
barcode	varchar (255)	Основний штрих-код товару
title	varchar (255)	Назва товару
category	varchar (255)	Категорія товару
price	double	Основна ціна товару

Кінець таблиці 2.3

created_at	timestamp (0)	Дата створення запису
promo_title	varchar (255)	Назва акційної пропозиції
promo_start_at	timestamp (0)	Дата початку акції
promo_end_at	timestamp (0)	Дата завершення акції
promo_discount_percent	integer	Відсоток знижки
promo_discount	double	Розмір знижки
promo_promo_price	double	Акційна ціна
promo_type	varchar (255)	Тип акції
promo_price_type	varchar (255)	Тип формування акційної ціни
description	text	Опис товару
status	varchar (255)	Статус товару
image	varchar (255)	Посилання на зображення
price_mobile	double	Ціна для мобільного відображення
barcodes	jsonb	Додаткові штрих-коди товару

Окрему увагу потрібно приділити полю barcodes, яке має тип jsonb. Це поле використовується для зберігання масиву додаткових штрих-кодів, які можуть бути пов’язані з одним товаром. Такий підхід є зручним у випадках, коли один товар має кілька варіантів маркування або постачається різними партіями. Використання jsonb дозволяє зберігати гнучку структуру даних без створення додаткової таблиці.

Для забезпечення швидкого пошуку та унікальності даних у таблиці створено індекси. Поля sku та barcode мають унікальні індекси, що запобігає дублюванню товарів за артикулом або основним штрих-кодом. Для поля barcodes створено GIN-індекс, який оптимізує пошук у структурі JSONB.

Схему пошуку товару в базі даних за штрих-кодом проілюстровано на рисунку 2.5.

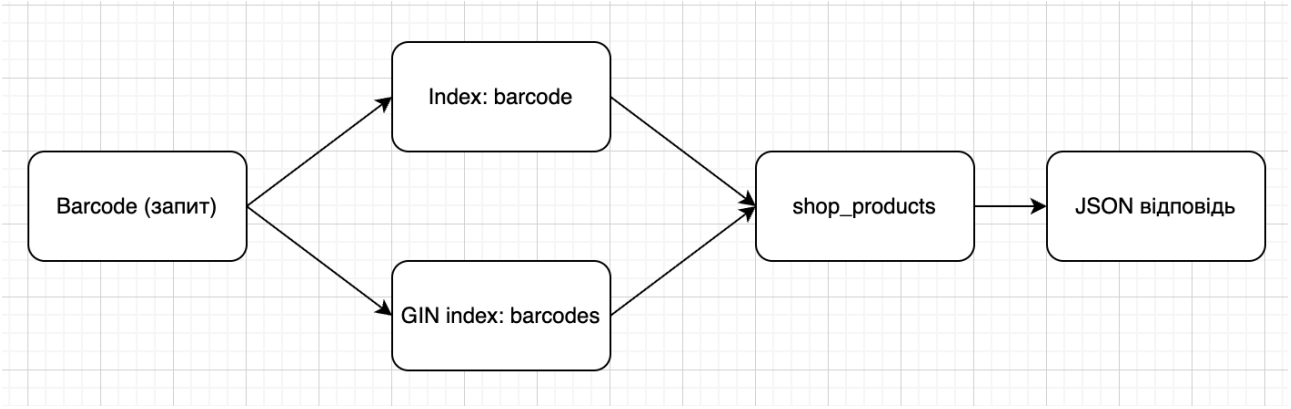


Рисунок 2.5 - Схема пошуку товару в базі даних за штрих-кодом

Індекси таблиці shop_products описані в таблиці 2.4.

Таблиця 2.4 - Індекси таблиці shop_products

Назва індексу	Поле	Тип	Призначення
uniq_6802a0ccf9038c4	sku	Unique index	Забезпечення унікальності артикула
uniq_6802a0cc97ae0266	barcode	Unique index	Забезпечення унікальності основного штрих-коду
idx_barcodes_jsonb	barcodes	GIN index	Прискорення пошуку серед додаткових штрих-кодів

З точки зору роботи кіберфізичної системи найбільш важливим є швидкий пошук товару за штрих-кодом. Саме тому поле barcode є обов’язковим та унікальним. У разі успішного пошуку сервер формує відповідь, яка містить назву товару, артикул, ціну, акційну ціну та зображення. Якщо товар не знайдено, клієнтський застосунок відображає повідомлення про відсутність товару.

Типовий запит до бази даних може мати такий вигляд:

```
SELECT id, sku, barcode, title, price, promo_promo_price, image
FROM shop_products
WHERE barcode = :barcode
   OR barcodes @> :barcode_json;
```

де :barcode - штрих-код, отриманий від сканера;

:barcode_json - значення штрих-коду у форматі JSONB для пошуку в полі додаткових штрих-кодів.

Загальна структура бази даних у межах даного підрозділу є достатньою для реалізації основних функцій системи. Вона забезпечує зберігання базових відомостей про товар, підтримку акційних пропозицій, роботу із зображеннями та можливість пошуку за кількома варіантами штрих-кодів.

У подальшому структуру бази даних можна розширити, додавши окремі таблиці для категорій, історії зміни цін, залишків товарів на складах та статистики сканування. Однак для реалізації базової кіберфізичної системи «Сканер товарів» достатньо використання таблиці shop_products, оскільки вона охоплює основні дані, необхідні для відображення інформації покупцю.

У підрозділі виконано проєктування структури бази даних кіберфізичної системи «Сканер товарів». Визначено основну таблицю shop_products, описано її поля, призначення основних атрибутів та механізм пошуку товару за штрих-кодом. Також розглянуто індекси, які забезпечують унікальність даних і підвищують швидкодію запитів.

Запропонована структура бази даних дозволяє зберігати інформацію про товари, ціни, акційні пропозиції та зображення, а також забезпечує ефективну взаємодію між серверною частиною та клієнтським застосунком.

2.4 Висновки до другого розділу

У другому розділі кваліфікаційної роботи виконано проєктування кіберфізичної системи «Сканер товарів» на базі Raspberry Pi та визначено

основні принципи її функціонування. На основі результатів аналізу предметної області сформовано структуру програмно-технічного засобу, що включає апаратну, клієнтську та серверну підсистеми.

У процесі проєктування визначено склад апаратних компонентів системи, до яких належать сканер штрих-кодів, одноплатний комп’ютер Raspberry Pi, дисплей та засоби комунікації між пристроями. Встановлено, що використання Raspberry Pi як центрального обчислювального модуля дозволяє забезпечити достатню продуктивність системи при невисоких апаратних витратах.

Також у межах другого розділу розроблено архітектуру системи, яка базується на клієнт-серверному підході. Клієнтська частина реалізована у вигляді Android-застосунку на мові програмування Kotlin, а серверна частина забезпечує обробку запитів та взаємодію з базою даних PostgreSQL. Описано принципи обміну інформацією між компонентами системи за допомогою REST API та формату JSON.

Окрему увагу приділено проєктуванню бази даних системи. Визначено структуру таблиці shop_products, яка забезпечує зберігання інформації про товари, ціни, акційні пропозиції, зображення та додаткові штрих-коди. Розглянуто механізми індексації даних, що дозволяють підвищити швидкодію пошуку товарів за штрих-кодом.

У результаті проведеного проєктування сформовано логічну та функціональну структуру кіберфізичної системи «Сканер товарів», визначено взаємодію між її компонентами та підготовлено основу для подальшої програмно-апаратної реалізації системи, яка буде розглянута у третьому розділі.

3 ПРОГРАМНО-АПАРАТНА РЕАЛІЗАЦІЯ КІБЕРФІЗИЧНОЇ СИСТЕМИ «СКАНЕР ТОВАРІВ»

3.1 Опис реалізації модулів апаратного та програмного забезпечення програмно-технічного засобу

Програмно-апаратна реалізація кіберфізичної системи «Сканер товарів» передбачає інтеграцію апаратних компонентів із програмним забезпеченням для забезпечення автоматизованого отримання та відображення інформації про товар у режимі реального часу. Основною метою реалізації є створення стабільної системи, яка забезпечує швидке сканування штрих-кодів, обробку даних та зручну взаємодію з користувачем у торговельному середовищі.

На відміну від класичних касових систем, запропонована система орієнтована на самообслуговування покупців, тому особлива увага приділяється простоті інтерфейсу, швидкості роботи та стабільності функціонування.

Апаратна реалізація системи базується на використанні одноплатного комп’ютера Raspberry Pi, який виконує функцію центрального обчислювального модуля. До Raspberry Pi підключено сканер штрих-кодів, дисплей та периферійні пристрої, необхідні для забезпечення роботи системи.

Для реалізації кіберфізичної системи використано одноплатний комп’ютер Raspberry Pi 4 Model B, який забезпечує обробку даних, взаємодію з периферійними пристроями та роботу Android-застосунку. Raspberry Pi має достатню обчислювальну потужність для виконання задач сканування, обробки HTTP-запитів та відображення інформації на дисплеї у режимі реального часу.

Сканер штрих-кодів підключається через USB-інтерфейс та функціонує у режимі HID-клавіатури. Такий підхід дозволяє спростити інтеграцію обладнання, оскільки система сприймає сканер як звичайний пристрій введення. Після зчитування штрих-коду сканер передає послідовність символів до клієнтського застосунку.

Для зчитування інформації про товари використовується USB-сканер штрих-кодів, який функціонує у режимі HID-клавіатури. Після сканування штрих-коду пристрій автоматично передає послідовність символів до Android-застосунку без необхідності встановлення додаткових драйверів.

Одноплатний комп’ютер Raspberry Pi 4 Model B зображено на рисунках 3.1-3.2.

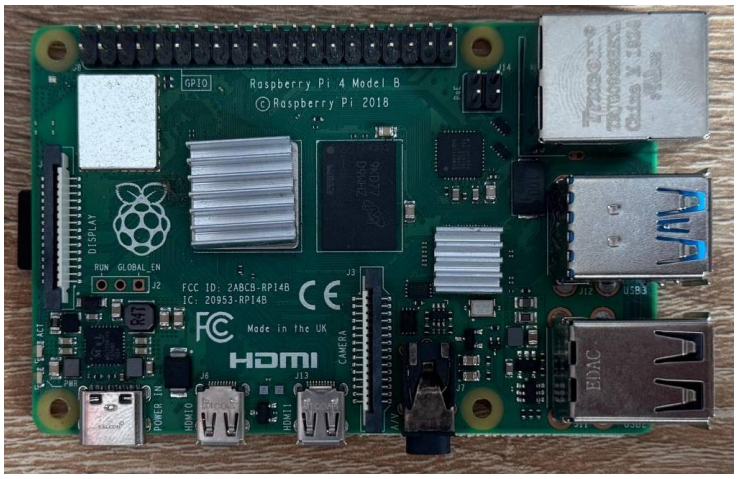


Рисунок 3.1 - Одноплатний комп’ютер Raspberry Pi 4 Model B

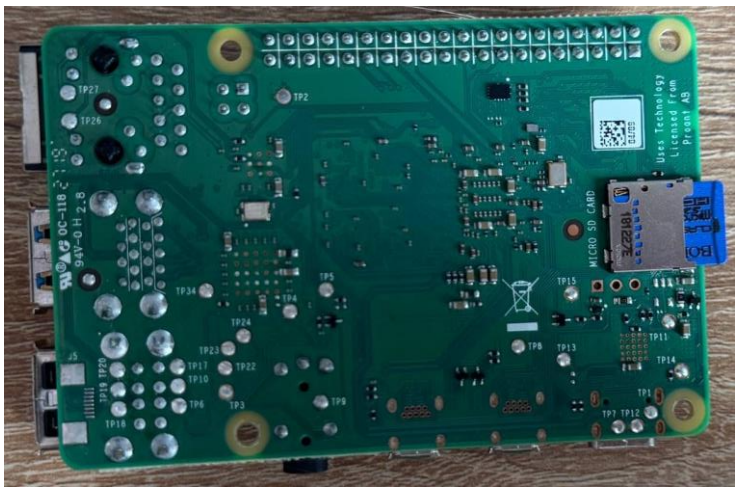


Рисунок 3.2 - Одноплатний комп’ютер Raspberry Pi 4 Model B

Для відображення інформації використовується дисплей, підключений через HDMI. На екрані відображається інформація про товар, включаючи назву,

основну ціну, акційну ціну та зображення товару. У режимі очікування система демонструє рекламні банери.

USB-сканер штрих-кодів показано на рисунках 3.3-3.4.

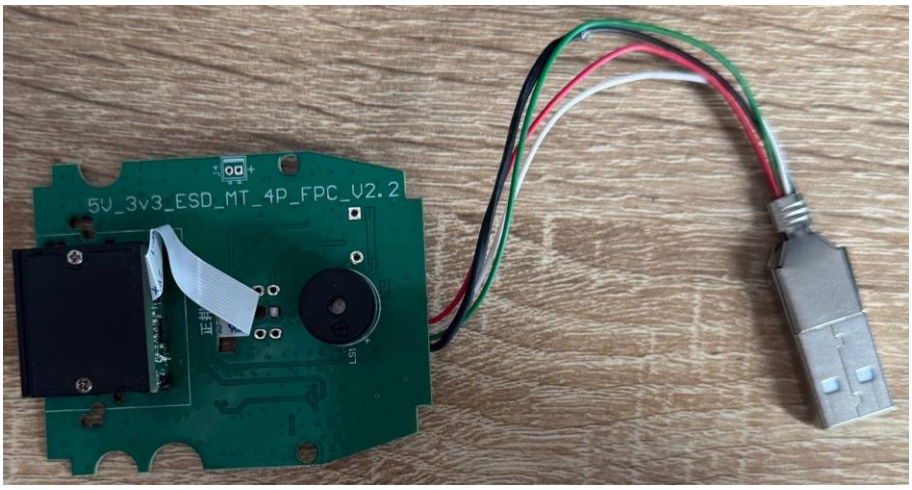


Рисунок 3.3 - USB-сканер штрих-кодів

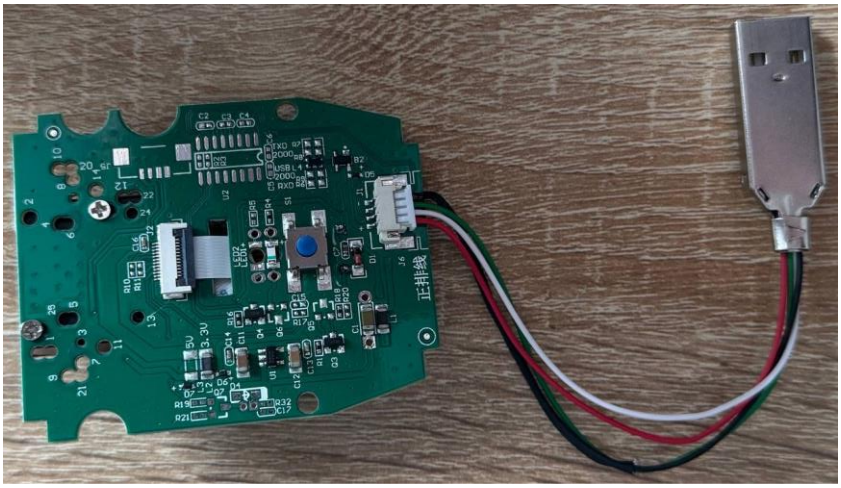


Рисунок 3.4 - USB-сканер штрих-кодів

У якості засобу відображення інформації використовується 7-дюймовий LCD-дисплей із підтримкою сенсорного керування. Дисплей забезпечує відображення інформації про товар, рекламних банерів та взаємодію користувача із системою.

LCD-дисплей системи сканування товарів наведено на рисунку 3.5.

Рисунок 3.5 - 7-дюймовий LCD-дисплей системи сканування товарів

Для забезпечення роботи дисплея використовується окремий контролер із HDMI-інтерфейсом, який забезпечує передачу відеосигналу від Raspberry Pi до LCD-панелі та підтримку сенсорного введення.

Контролер LCD-дисплея з HDMI-інтерфейсом показано на рисунку 3.6.

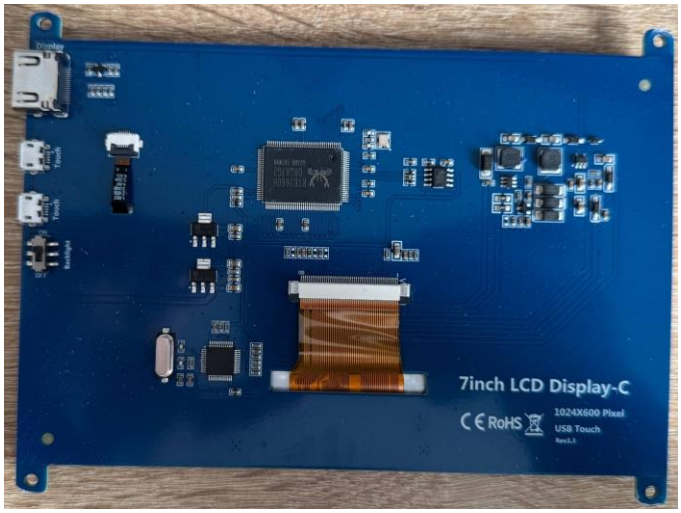


Рисунок 3.6 - Контролер LCD-дисплея з HDMI-інтерфейсом

Схему підключення апаратних компонентів системи проілюстровано на рисунку 3.7.

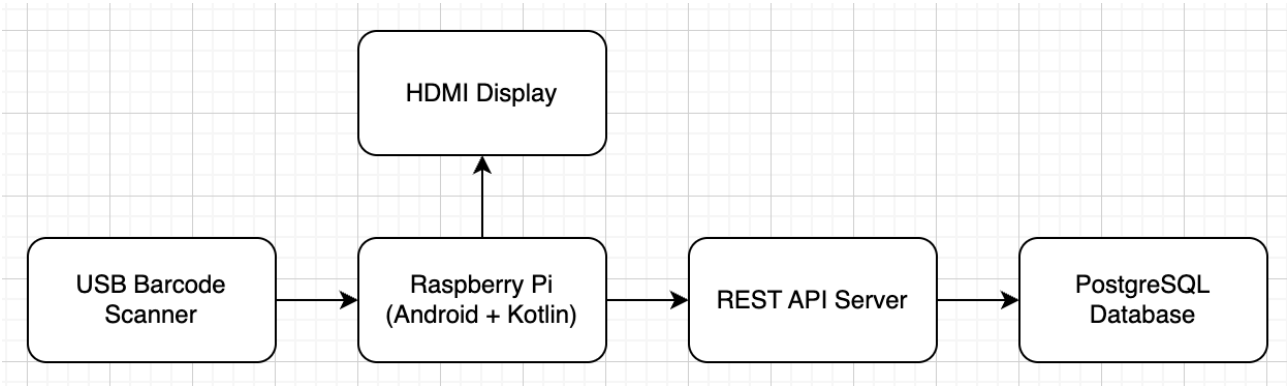


Рисунок 3.7 - Схема підключення апаратних компонентів системи

Для забезпечення стабільної роботи системи використовується блок живлення з напругою 5 В, який забезпечує живлення Raspberry Pi та підключених пристроїв. Підключення до серверної частини здійснюється через мережу Wi-Fi або Ethernet.

Програмна частина системи реалізована у вигляді Android-застосунку, розробленого на мові програмування Kotlin. Вибір Kotlin обумовлений підтримкою сучасних Android-технологій, високою продуктивністю та зручністю розробки.

Архітектура застосунку побудована таким чином, щоб забезпечити швидку обробку даних та мінімізувати затримки при скануванні товарів. Центральним компонентом застосунку є MainActivity, який відповідає за:

- отримання даних від сканера;
- формування HTTP-запитів;
- обробку відповідей сервера;
- керування відображенням інтерфейсу.

Згідно з технічною архітектурою проєкту, застосунок реалізований за схемою «один Activity - один основний Fragment», що дозволяє спростити структуру програми та зменшити навантаження на систему [ОВ].

Після отримання штрих-коду застосунок формує HTTP-запит до серверного API за допомогою бібліотеки OkHttp. Сервер повертає дані у форматі

JSON, після чого система виконує їх обробку та відображення у користувацькому інтерфейсі.

Типовий процес обробки інформації включає:

1. Зчитування штрих-коду.
2. Формування API-запиту.
3. Отримання відповіді сервера.
4. Парсинг JSON-даних.
5. Відображення інформації на екрані.

Алгоритм роботи програмної частини системи показано на рисунку 3.8.

Рисунок 3.8 - Алгоритм роботи програмної частини системи

Інтерфейс користувача розроблено з урахуванням використання системи у торговельному залі. Основною вимогою до інтерфейсу є забезпечення швидкого сприйняття інформації покупцем.

Після сканування товару на екрані відображаються:

- назва товару;
- основна ціна;
- акційна ціна (за наявності);
- зображення товару;
- артикул або штрих-код.

У випадку, якщо товар не знайдено у базі даних, система відображає повідомлення «Товар не знайдено». Такий підхід дозволяє забезпечити однозначну реакцію системи на помилки або відсутність інформації.

Для відображення зображень та рекламних банерів використовується бібліотека Glide, яка забезпечує кешування та швидке завантаження графічних ресурсів.

Інтерфейс користувача кіберфізичної системи «Сканер товарів» у режимі рекламного банера та показу відсканованого товару представлено на рисунках 3.9-3.10.

Рисунок 3.9 - Інтерфейс користувача кіберфізичної системи «Сканер товарів». Режим рекламного банера

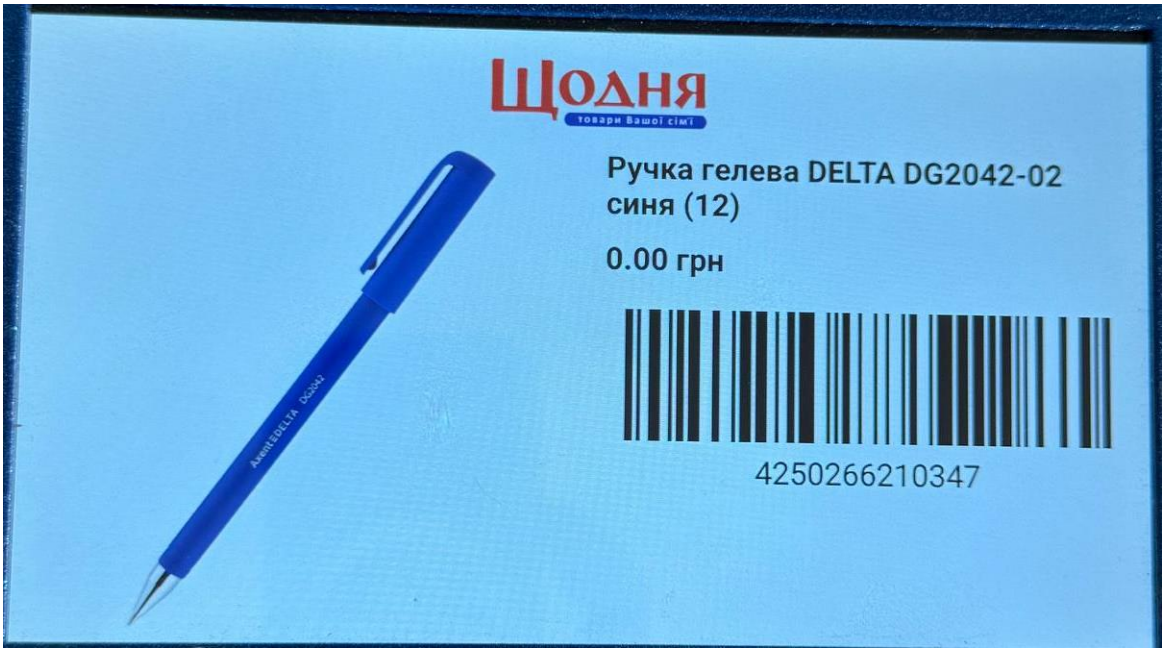


Рисунок 3.10 - Інтерфейс користувача кіберфізичної системи «Сканер товарів». Режим показу відсканованого товару

Реалізована система має ряд особливостей, які забезпечують її ефективність у реальних умовах використання:

- підтримка роботи у повноекранному режимі;
- автоматичне оновлення рекламних банерів;
- підтримка акційних цін;
- швидке отримання інформації про товар;
- можливість роботи у режимі кіоску.

У межах підрозділу виконано опис програмно-апаратної реалізації кіберфізичної системи «Сканер товарів». Розглянуто структуру апаратної частини, принципи взаємодії компонентів системи та особливості реалізації Android-застосунку на мові програмування Kotlin.

Також описано механізм обробки штрих-кодів, взаємодію з REST API та відображення інформації у користувацькому інтерфейсі. Реалізована система забезпечує швидке отримання інформації про товар та може використовуватись у реальних умовах торговельного середовища.

3.2 Опис процесу створення баз даних

Одним із ключових етапів реалізації кіберфізичної системи «Сканер товарів» є створення бази даних, яка забезпечує централізоване зберігання інформації про товари, ціни, акційні пропозиції та додаткові характеристики. Саме база даних є основним джерелом інформації для клієнтського застосунку, який отримує дані через REST API після сканування штрих-коду товару.

Процес проєктування бази даних розпочинається зі створення логічної та фізичної моделі даних. Для побудови структури бази даних доцільно використовувати CASE-засоби моделювання, які дозволяють візуалізувати сутності, їх атрибути та зв’язки між таблицями. У межах даної роботи фізична

модель бази даних побудована на основі структури таблиці shop_products, яка використовується серверною частиною системи.

Фізичну модель бази даних кіберфізичної системи «Сканер товарів» проілюстровано на рисунку 3.11.

Рисунок 3.11 - Фізична модель бази даних кіберфізичної системи «Сканер товарів»

На відміну від складних багатотабличних систем, у даній роботі використовується спрощена структура бази даних, оскільки основна задача полягає у швидкому отриманні інформації про товар за штрих-кодом. Основною сутністю є таблиця shop_products, яка містить усі необхідні дані для функціонування системи.

До структури таблиці входять:

- унікальний ідентифікатор товару (id);
- артикул (sku);
- штрих-код (barcode);

- назва товару (title);
- ціна (price);
- акційна ціна (promo_promo_price);
- опис товару (description);
- зображення (image);
- додаткові штрих-коди (barcodes).

Така структура дозволяє забезпечити швидкий доступ до інформації без необхідності виконання складних SQL-запитів або об’єднання великої кількості таблиць.

Після створення фізичної моделі здійснюється генерація структури бази даних у PostgreSQL. Для цього формується SQL-код створення таблиці shop_products, який містить опис полів, типів даних, первинного ключа та індексів.

Приклад створення таблиці:

```
CREATE TABLE shop_products
(
    id uuid PRIMARY KEY,
    sku varchar(255) NOT NULL,
    barcode varchar(255) NOT NULL,
    title varchar(255) NOT NULL,
    category varchar(255) NOT NULL,
    price double precision NOT NULL,
    description text,
    image varchar(255),
    status varchar(255) NOT NULL,
    promo_promo_price double precision,
    barcodes jsonb NOT NULL DEFAULT '[]'::jsonb
);
```

Після виконання SQL-коду у системі PostgreSQL створюється фізична структура бази даних, яка надалі використовується серверною частиною застосунку.

Для забезпечення швидкого пошуку товарів у таблиці створюються індекси. Поля sku та barcode мають унікальні індекси, що виключає можливість дублювання товарів. Окремо створюється GIN-індекс для поля barcodes, яке має тип jsonb.

Структуру таблиці shop_products та індексів бази даних представлено на рисунку 3.12.

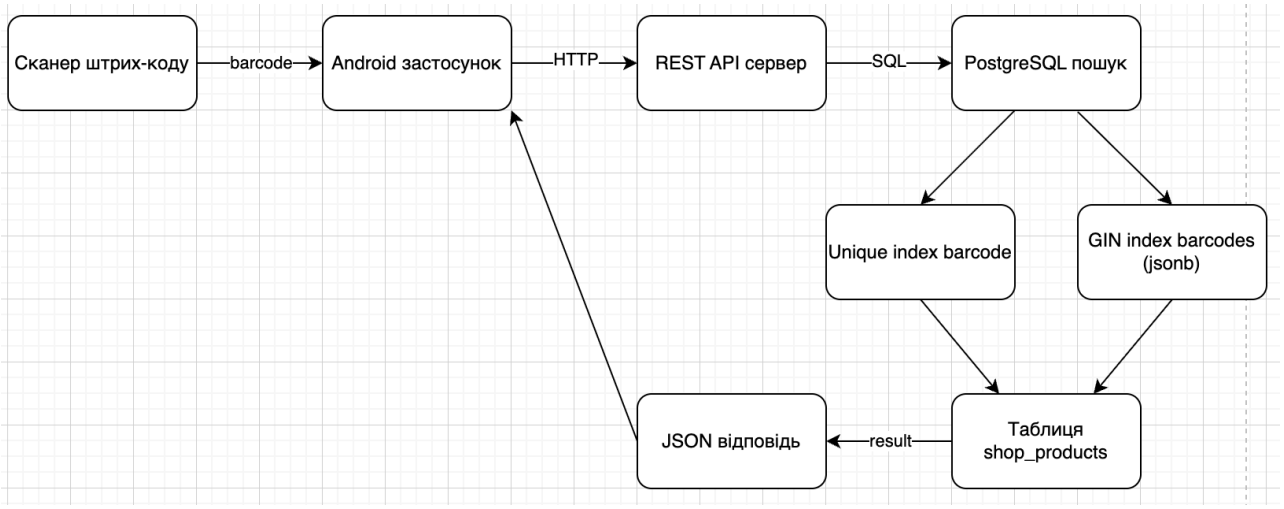


Рисунок 3.12 - Структура таблиці shop_products та індексів бази даних

Після створення структури бази даних виконується наповнення таблиці інформацією про товари. Дані можуть імпортуватися з зовнішніх інформаційних систем або додаватися вручну через адміністративну панель. Під час імпорту зберігаються назва товару, ціна, акційні параметри та посилання на зображення.

Важливою особливістю реалізованої бази даних є підтримка акційних пропозицій. Для цього використовуються поля:

- promo_start_at;
- promo_end_at;
- promo_discount_percent;
- promo_promo_price.

Серверна частина системи перевіряє актуальність акції та повертає клієнтському застосунку відповідну ціну товару.

Для перевірки працездатності бази даних виконуються тестові SQL-запити. Пошук товару здійснюється за штрих-кодом, який надходить від сканера через Android-застосунок.

Типовий запит має вигляд:

```
SELECT title, price, promo_promo_price, image
FROM shop_products
WHERE barcode = :barcode;
```

де :barcode - значення штрих-коду, отримане після сканування товару.

Результат виконання запиту використовується серверним API для формування JSON-відповіді, яка передається клієнтському застосунку.

Таким чином, створена база даних забезпечує:

- централізоване зберігання інформації про товари;
- підтримку акційних цін;
- швидкий пошук товарів за штрих-кодом;
- взаємодію з REST API;
- можливість подальшого масштабування системи.

У межах підрозділу розглянуто процес створення бази даних кіберфізичної системи «Сканер товарів». Побудовано фізичну модель даних, визначено структуру таблиці shop_products та реалізовано механізм індексації для забезпечення швидкого пошуку товарів.

Також описано процес генерації структури бази даних у PostgreSQL, створення SQL-запитів та механізм взаємодії бази даних із серверною частиною системи. Реалізована структура забезпечує ефективне зберігання та обробку інформації про товари у межах кіберфізичної системи.

3.3 Опис роботи кіберфізичної системи «Сканер товарів»

Робота кіберфізичної системи «Сканер товарів» базується на взаємодії апаратних і програмних компонентів, які забезпечують автоматизоване отримання інформації про товар після зчитування штрих-коду. Основною

задачею системи є швидке надання користувачу актуальної інформації про товар без необхідності звернення до персоналу магазину.

Функціонування системи починається з моменту піднесення товару до сканера штрих-кодів. Сканер виконує зчитування коду та передає отримані дані до Android-застосунку, який працює на Raspberry Pi. Оскільки сканер функціонує у режимі HID-клавіатури, передача даних відбувається у вигляді емуляції натискань клавіш, що спрощує інтеграцію обладнання та не потребує використання спеціалізованих драйверів.

Після отримання штрих-коду клієнтський застосунок виконує перевірку коректності введених даних та формує HTTP-запит до серверного API. Запит містить значення штрих-коду, яке використовується для пошуку товару у базі даних PostgreSQL.

Загальну схема роботи кіберфізичної системи «Сканер товарів» описано на рисунку 3.13.

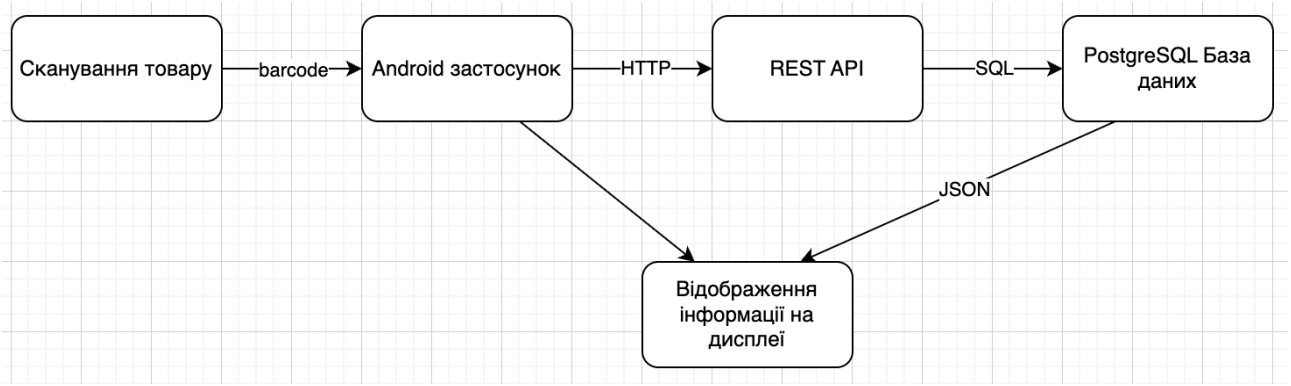


Рисунок 3.13 - Загальна схема роботи кіберфізичної системи «Сканер товарів»

На серверній стороні виконується обробка запиту та пошук відповідного товару у таблиці shop_products. Пошук здійснюється за основним полем barcode, а також за додатковими штрих-кодами, що зберігаються у полі barcodes(jsonb). Для прискорення роботи використовуються індекси бази даних, завдяки чому

система забезпечує швидке отримання результату навіть при великій кількості товарів.

Після успішного пошуку сервер формує JSON-відповідь, яка містить:

- назву товару;
- основну ціну;
- акційну ціну;
- опис товару;
- посилання на зображення.

Отримані дані передаються клієнтському застосунку, де виконується їх обробка та відображення у користувацькому інтерфейсі.

Важливою частиною роботи системи є механізм підтримки акційних пропозицій. Якщо для товару у базі даних визначено акційну ціну та період дії акції, система автоматично відображає оновлену ціну на екрані. Це дозволяє користувачу одразу бачити актуальну вартість товару без необхідності перевірки на касі.

У випадку, якщо товар відсутній у базі даних, застосунок відображає повідомлення «Товар не знайдено».

Відображення інформації про товар після сканування наведено на рисунку 3.14.

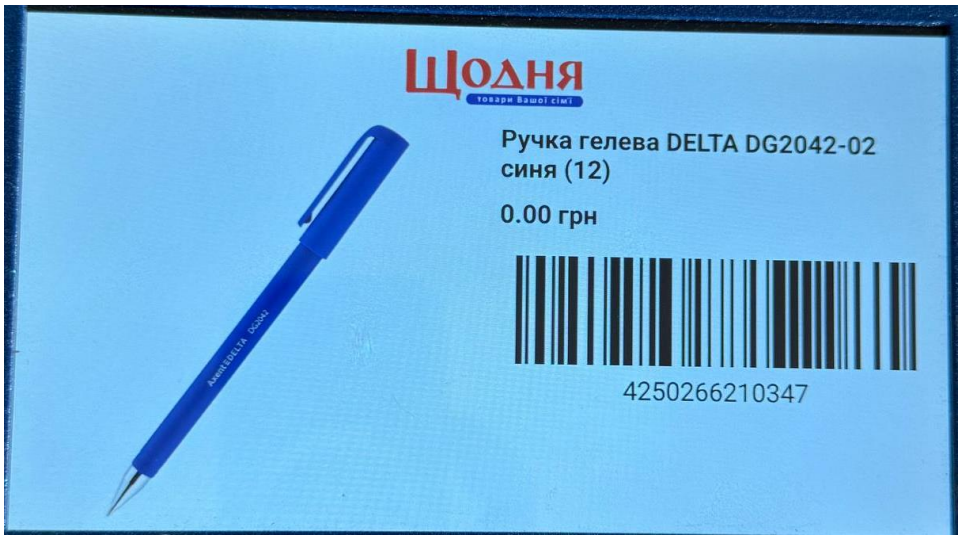


Рисунок 3.14 - Відображення інформації про товар після сканування

У режимі очікування система автоматично переходить до демонстрації рекламних банерів. Завантаження банерів здійснюється через окремий API-запит, а їх відображення реалізовано за допомогою компонента ViewFlipper. Це дозволяє використовувати систему не лише як інформаційний термінал, а і як засіб демонстрації маркетингового контенту.

Оновлення банерів та інформації про товари відбувається без необхідності перевстановлення застосунку, оскільки всі дані завантажуються із серверної частини. Такий підхід забезпечує централізоване керування контентом і спрощує адміністрування системи.

Для забезпечення стабільності роботи застосунку реалізовано механізм обробки помилок мережевого з’єднання. Якщо сервер недоступний або відсутнє підключення до мережі, система відображає повідомлення про помилку та повторює спробу підключення через певний проміжок часу.

Окрім цього, у системі реалізовано:

- автоматичне очищення екрана після завершення сеансу сканування;
- підтримку повноекранного режиму;
- автоматичне повернення до демонстрації банерів;
- оновлення застосунку через Google Play In-App Updates.

Основні етапи роботи системи перелічені у таблиці 3.1.

Таблиця 3.1 - Основні етапи роботи системи

Етап	Опис
Сканування	Зчитування штрих-коду товару
Обробка	Формування HTTP-запиту
Пошук	Пошук товару у PostgreSQL
Відповідь	Формування JSON-відповіді
Відображення	Показ інформації на екрані

Таким чином, реалізована кіберфізична система забезпечує автоматизовану обробку інформації про товари у режимі реального часу. Використання клієнт-серверної архітектури, REST API та PostgreSQL дозволяє забезпечити стабільну роботу системи, швидкий пошук товарів та централізоване оновлення даних.

У межах підрозділу розглянуто принцип роботи кіберфізичної системи «Сканер товарів» та описано процес взаємодії між її компонентами. Визначено основні етапи обробки інформації від сканування штрих-коду до відображення результату користувачу.

Також описано механізм роботи із серверною частиною, базою даних PostgreSQL та REST API. Реалізована система забезпечує швидке отримання актуальної інформації про товар та може використовуватись у реальних умовах торговельного середовища.

3.4 Тестування та аналіз роботи кіберфізичної системи «Сканер товарів»

Після завершення програмно-апаратної реалізації кіберфізичної системи «Сканер товарів» було проведено тестування її функціональних можливостей, стабільності роботи та швидкодії основних модулів. Основною метою тестування є перевірка коректності взаємодії між апаратною та програмною частинами системи, а також оцінка ефективності роботи системи в умовах, наближених до реального використання у торговельному середовищі.

Тестування проводилося поетапно та охоплювало перевірку:

- роботи сканера штрих-кодів;
- коректності передачі даних;
- взаємодії з REST API;
- швидкодії бази даних PostgreSQL;
- відображення інформації у користувацькому інтерфейсі;
- обробки помилок та нестандартних ситуацій.

Перед початком тестування система була розгорнута на Raspberry Pi та підключена до серверної частини через локальну мережу. У базу даних було додано тестовий набір товарів із різними категоріями, цінами та акційними пропозиціями.

На першому етапі перевірялася робота сканера штрих-кодів та коректність передачі даних до Android-застосунку. Під час тестування використовувалися реальні штрих-коди товарів різних форматів.

У результаті тестування встановлено, що система стабільно зчитує штрих-коди та передає їх до програмної частини без втрати символів. Середній час зчитування та обробки штрих-коду становив менше однієї секунди.

Результати тестування процесу сканування товару наведено на рисунку 3.15.

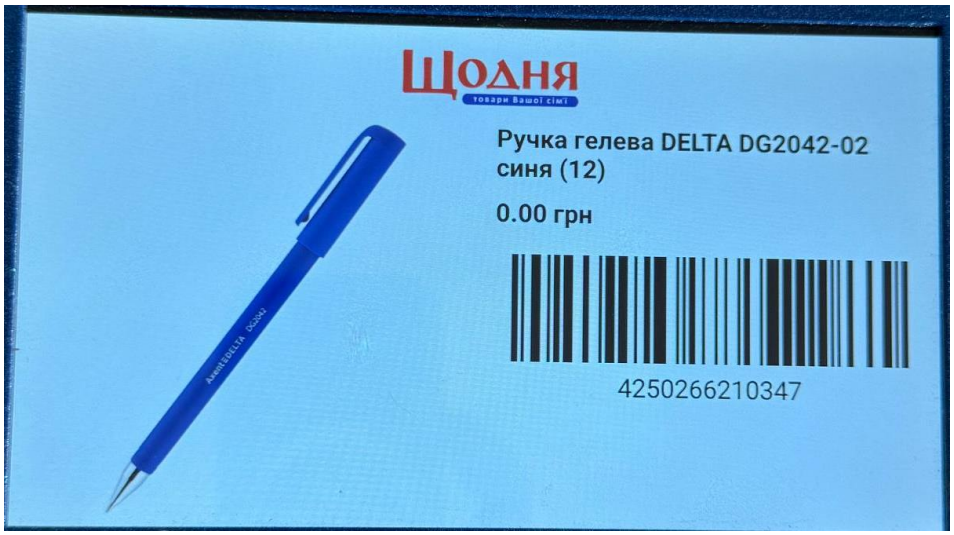


Рисунок 3.15 - Тестування процесу сканування товару

Під час тестування також було перевірено роботу системи при повторному скануванні товарів та при швидкому послідовному скануванні декількох позицій. Система успішно обробляла запити без зависань та помилок інтерфейсу.

Наступним етапом було тестування серверної частини та механізму пошуку товарів у базі даних PostgreSQL. Особлива увага приділялася швидкодії SQL-запитів та роботі індексів.

Пошук товару виконувався:

- за основним полем barcode;
- через GIN-індекс поля barcodes(jsonb);
- із використанням акційних параметрів товару.

У процесі тестування встановлено, що використання індексів дозволяє значно скоротити час пошуку товарів у базі даних навіть при збільшенні кількості записів.

Результати тестування швидкодії системи перелічено у таблиці 3.2.

Таблиця 3.2 - Результати тестування швидкодії системи

Операція	Середній час виконання
Зчитування штрих-коду	0.3-0.5 с
Формування HTTP-запиту	0.1 с
Пошук товару у PostgreSQL	0.05-0.2 с
Формування JSON-відповіді	0.05 с
Відображення результату	0.2-0.4 с

Отримані результати свідчать про те, що система забезпечує швидке отримання інформації про товар та може використовуватись у режимі реального часу без помітних затримок для користувача.

Результат пошуку товару у базі даних PostgreSQL показано на рисунку 3.16.

Рисунок 3.16 - Результат пошуку товару у базі даних PostgreSQL

Окремо проводилося тестування інтерфейсу користувача. Основною задачею було перевірити зручність сприйняття інформації на екрані та швидкість оновлення даних після сканування товару.

У результаті тестування встановлено, що:

- назва товару відображається коректно;
- ціна та акційна ціна мають достатній розмір шрифту;
- зображення товару завантажується без затримок;
- система автоматично повертається до банерного режиму після завершення роботи з товаром.

Також перевірялася реакція системи на помилки. У випадку відсутності товару у базі даних застосунок коректно відображав повідомлення «Товар не знайдено».

Для оцінки стабільності роботи система тестувалася протягом тривалого часу у режимі безперервної роботи. Під час тестування перевірялися:

- стабільність мережевого підключення;
- використання оперативної пам’яті Raspberry Pi;
- навантаження на процесор;
- стабільність роботи Android-застосунку.

У результаті тестування критичних помилок або аварійного завершення роботи застосунку не виявлено. Система стабільно функціонувала при тривалому навантаженні та забезпечувала коректну взаємодію між усіма компонентами.

Взаємодію компонентів системи під час тестування проілюстровано на рисунку 3.17.

У результаті проведеного тестування підтверджено працездатність кіберфізичної системи «Сканер товарів» та ефективність обраної архітектури. Використання Raspberry Pi, Android-застосунку на Kotlin та PostgreSQL

дозволило реалізувати стабільну систему для автоматизованого отримання інформації про товари.

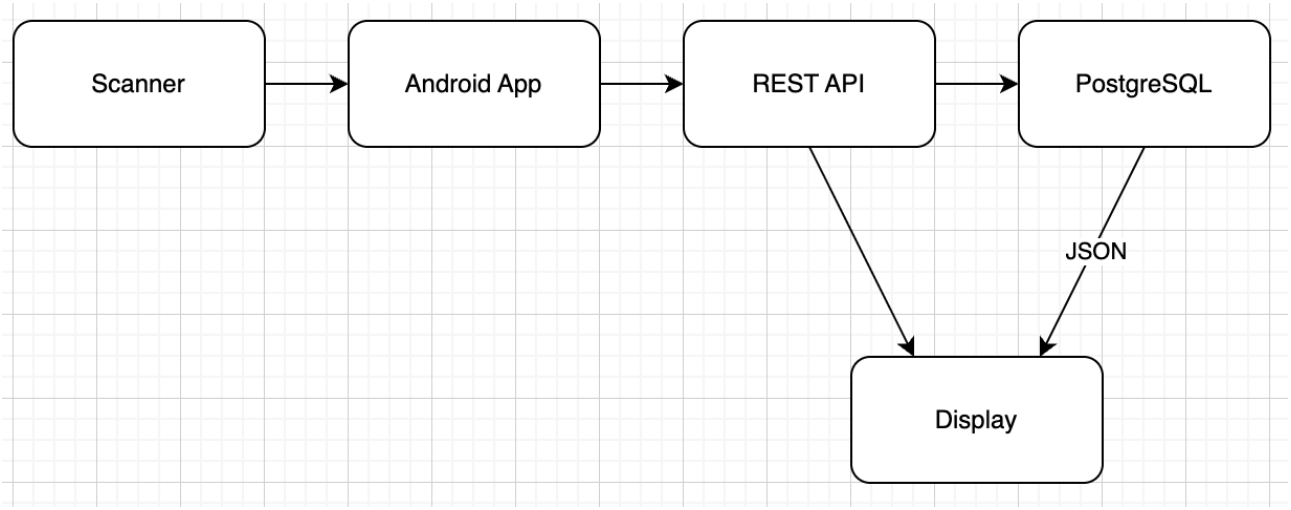


Рисунок 3.17 - Взаємодія компонентів системи під час тестування

У межах підрозділу проведено тестування програмно-апаратної реалізації кіберфізичної системи «Сканер товарів». Перевірено роботу сканера штрих-кодів, серверної частини, бази даних PostgreSQL та користувацького інтерфейсу.

Результати тестування підтвердили коректність роботи системи, швидке отримання інформації про товари та стабільність функціонування всіх компонентів у режимі реального часу.

3.5 Аналіз ефективності та перспектив розвитку кіберфізичної системи «Сканер товарів»

Розроблена кіберфізична система «Сканер товарів» на базі Raspberry Pi є прикладом інтеграції програмного та апаратного забезпечення для автоматизації процесів отримання інформації про товари у торговельному середовищі. Реалізована система забезпечує швидкий доступ покупців до актуальної інформації про продукцію, що дозволяє підвищити рівень обслуговування та зменшити навантаження на персонал магазину.

У процесі тестування та експлуатації системи було встановлено, що використання Raspberry Pi як центрального обчислювального модуля є ефективним рішенням для реалізації подібних кіберфізичних систем. Одноплатний комп’ютер забезпечує достатню продуктивність для роботи Android-застосунку, взаємодії з REST API та відображення інформації на дисплеї у режимі реального часу.

Важливою перевагою системи є використання клієнт-серверної архітектури. Такий підхід дозволяє відокремити логіку обробки даних від інтерфейсу користувача та забезпечує централізоване керування інформацією про товари. Оновлення цін, акцій та описів товарів виконується без необхідності втручання у роботу клієнтського застосунку, оскільки всі дані зберігаються у PostgreSQL та передаються через REST API.

Аналіз ефективності роботи кіберфізичної системи наведено на рисунку 3.18.

Рисунок 3.18 - Аналіз ефективності роботи кіберфізичної системи

Під час аналізу роботи системи було визначено основні переваги реалізованого рішення:

- швидке отримання інформації про товар;
- підтримка акційних цін та банерної системи;
- можливість централізованого оновлення даних;

- низьке енергоспоживання Raspberry Pi;
- простота інтеграції зі сканерами штрих-кодів;
- підтримка декількох штрих-кодів для одного товару.

Особливу роль у забезпеченні продуктивності системи відіграє використання індексів PostgreSQL. Пошук товарів за основним штрих-кодом та через поле barcodes(jsonb) дозволяє ефективно працювати навіть при значному обсязі даних. Використання GIN-індексації суттєво зменшує час виконання SQL-запитів.

Під час експлуатації було встановлено, що система забезпечує стабільну роботу при безперервному функціонуванні та великій кількості послідовних запитів. Час від моменту сканування товару до відображення результату на дисплеї є мінімальним та практично непомітним для користувача.

Для оцінки продуктивності враховувалися:

- швидкість сканування;
- час відповіді REST API;
- швидкодія PostgreSQL;
- стабільність Android-застосунку;
- навантаження на Raspberry Pi.

У результаті аналізу встановлено, що навіть при одночасній роботі банерного режиму та процесу сканування система не перевищує допустимі показники використання оперативної пам’яті та процесора.

Основні показники ефективності системи зібрано у таблиці 3.3.

Таблиця 3.3 - Основні показники ефективності системи

Показник	Значення
Середній час пошуку товару	0.05-0.2 с
Час повного відображення результату	до 1 с

Кінець таблиці 3.3

Середнє навантаження CPU Raspberry Pi	15-30 %

Середнє використання RAM	400-700 МБ
Підтримка роботи 24/7	Так

Окрім функціональних переваг, система має і практичну економічну ефективність. Використання Raspberry Pi значно зменшує вартість розробки порівняно з використанням повноцінних комп'ютерних систем або спеціалізованих POS-терміналів. При цьому забезпечується достатня продуктивність та можливість масштабування системи.

Незважаючи на реалізований функціонал, система може бути розширена та вдосконалена. Одним із перспективних напрямів є інтеграція з торговельними ERP-системами та сервісами автоматичного оновлення цін у режимі реального часу.

Також перспективними напрямами розвитку є:

- підтримка QR-кодів;
- інтеграція з RFID-мітками;
- реалізація голосового пошуку товарів;
- підтримка мультимовного інтерфейсу;
- використання штучного інтелекту для рекомендацій товарів;
- збір статистики взаємодії користувачів із системою.

Додатково система може бути адаптована для використання:

- у складських приміщеннях;
- у логістичних центрах;
- у системах самообслуговування;
- у медичних та фармацевтичних закладах.

Перспективи розвитку кіберфізичної системи «Сканер товарів» наведені на рисунку 3.19.

Таким чином, розроблена кіберфізична система «Сканер товарів» демонструє ефективність використання сучасних програмно-апаратних засобів для автоматизації процесів у сфері роздрібної торгівлі. Система забезпечує

швидкий доступ до інформації про товари, підтримує централізовану обробку даних та має значний потенціал для подальшого розвитку.

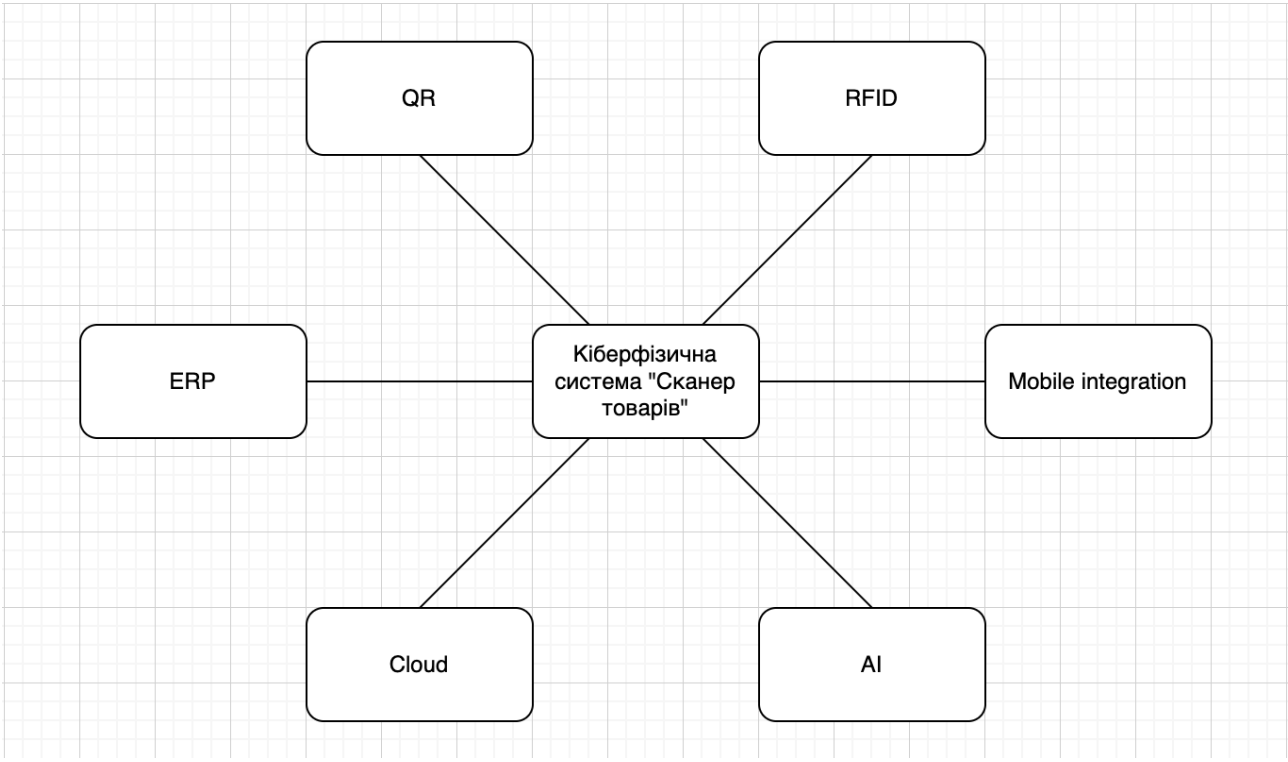


Рисунок 3.19 - Перспективи розвитку кіберфізичної системи «Сканер товарів»

У межах підрозділу проведено аналіз ефективності роботи кіберфізичної системи «Сканер товарів» та визначено основні переваги реалізованого рішення. Оцінено продуктивність системи, стабільність її роботи та ефективність використання Raspberry Pi як центрального обчислювального модуля.

Також визначено перспективні напрями розвитку системи, що передбачають інтеграцію з сучасними інформаційними технологіями та розширення функціональних можливостей програмно-апаратного комплексу.

3.6. Висновки до третього розділу

У межах третього розділу виконано програмно-апаратну реалізацію кіберфізичної системи «Сканер товарів» на базі Raspberry Pi. У процесі роботи

було реалізовано взаємодію між апаратними компонентами системи, Android-застосунком, серверною частиною та базою даних PostgreSQL.

Під час розробки програмного забезпечення реалізовано механізм автоматизованого зчитування штрих-кодів, передачі даних до REST API та отримання інформації про товари у режимі реального часу. Для створення клієнтської частини використано мову програмування Kotlin, що дозволило забезпечити стабільну роботу Android-застосунку та зручний користувацький інтерфейс.

У ході реалізації системи розроблено структуру бази даних PostgreSQL, яка забезпечує централізоване зберігання інформації про товари, ціни, акційні пропозиції та додаткові штрих-коди. Для оптимізації пошуку товарів використано механізми індексації, зокрема unique index та GIN index для поля barcodes(jsonb).

Також у межах розділу виконано:

- реалізацію механізму пошуку товарів за штрих-кодом;
- інтеграцію Android-застосунку з REST API;
- реалізацію режиму відображення рекламних банерів;
- підтримку акційних цін;
- тестування швидкодії та стабільності роботи системи.

У результаті проведеного тестування підтверджено працездатність усіх компонентів кіберфізичної системи. Встановлено, що система забезпечує швидке отримання інформації про товар, стабільну роботу при тривалому навантаженні та коректну взаємодію між програмними й апаратними модулями.

Проведений аналіз ефективності показав, що використання Raspberry Pi є доцільним для реалізації подібних кіберфізичних систем завдяки низькому енергоспоживанню, компактності та достатній продуктивності. Використання клієнт-серверної архітектури та PostgreSQL дозволяє забезпечити масштабованість системи та централізоване керування даними.

Окрім цього, визначено перспективні напрями подальшого розвитку системи, серед яких:

- підтримка QR-кодів та RFID;
- інтеграція з ERP-системами;
- використання хмарних сервісів;
- реалізація мобільної інтеграції;
- застосування елементів штучного інтелекту для аналізу товарів та рекомендацій користувачам.

Таким чином, у третьому розділі реалізовано та протестовано кіберфізичну систему «Сканер товарів», яка забезпечує автоматизоване отримання інформації про товари у торговельному середовищі та може бути використана як основа для подальшого розвитку систем самообслуговування у сфері роздрібної торгівлі.

ВИСНОВКИ

У кваліфікаційній роботі за результатами проведених теоретичних і практичних досліджень було розроблено кіберфізичну систему «Сканер товарів» на базі Raspberry Pi, призначену для автоматизованого отримання інформації про товари у сфері роздрібної торгівлі. Розроблена система забезпечує зчитування штрих-кодів, обробку запитів через серверну частину та відображення актуальної інформації про товар у режимі реального часу.

Актуальність виконаної роботи обумовлена активним розвитком цифрових технологій у сфері торгівлі та необхідністю підвищення рівня автоматизації процесів обслуговування покупців. Використання кіберфізичних систем у торговельному середовищі дозволяє забезпечити швидкий доступ до інформації про товари, зменшити навантаження на персонал та підвищити зручність взаємодії покупця з торговельною інфраструктурою.

У першому розділі проведено аналіз структурних і функціональних особливостей кіберфізичних систем у сфері автоматизованої обробки інформації. Розглянуто сучасні підходи до побудови систем автоматичної ідентифікації товарів, проаналізовано принципи роботи штрихового кодування та визначено основні проблеми, що виникають у процесі отримання інформації про товари у торговельних закладах. Окрему увагу приділено аналізу програмно-апаратного забезпечення, яке може використовуватись для реалізації подібних систем. У результаті аналізу обґрунтовано доцільність використання Raspberry Pi як центрального обчислювального модуля кіберфізичної системи.

У другому розділі виконано проєктування системи обробки інформації та визначено структуру програмних і апаратних підсистем програмно-технічного засобу. Було розроблено архітектуру взаємодії між Android-застосунком, REST API та базою даних PostgreSQL. Також визначено функціональне призначення основних модулів системи, побудовано структуру бази даних та реалізовано механізм пошуку товарів за штрих-кодом. У межах розділу розглянуто

особливості використання JSONB-полів та індексації PostgreSQL для забезпечення швидкого пошуку інформації про товари.

У третьому розділі виконано програмно-апаратну реалізацію кіберфізичної системи «Сканер товарів». Реалізовано Android-застосунок мовою програмування Kotlin, механізм взаємодії зі сканером штрих-кодів та REST API, а також систему відображення інформації про товар і рекламних банерів. Створено структуру бази даних PostgreSQL та реалізовано SQL-запити для пошуку товарів за основним і додатковими штрих-кодами. Проведено тестування роботи системи, у результаті якого підтверджено стабільність функціонування, коректність взаємодії між компонентами та достатню швидкодію системи у режимі реального часу.

Проведений аналіз показав, що використання Raspberry Pi є ефективним рішенням для реалізації подібних кіберфізичних систем завдяки компактності, низькому енергоспоживанню та достатній продуктивності. Використання клієнт-серверної архітектури забезпечує централізоване керування інформацією про товари та можливість подальшого масштабування системи.

Практична цінність отриманих результатів полягає у можливості використання розробленої системи у реальних умовах торговельного середовища для підвищення якості обслуговування покупців та автоматизації процесів отримання інформації про товари. Розроблена система також може бути адаптована для використання у складських комплексах, логістичних системах та системах самообслуговування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Barcode Graphics. GS1 Barcode Service FAQs. URL: https://www.barcode.graphics/gs1-barcode-service-faqs/.

2. Duo W., Zhou M., Abusorrah A. A survey of cyber attacks on cyber physical systems: Recent advances and challenges. *IEEE/CAA Journal of Automatica Sinica*. 2022. Vol.9(5). P. 784-800.

3. Lee E. A. Cyber Physical Systems: Design Challenges. *11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC).* Orlando, USA, 2008. P. 363-369.

4. Rajkumar R., Lee I., Sha L., Stankovic J. Cyber-Physical Systems: The Next Computing Revolution. *Design Automation Conference*. Anaheim, USA, 2010. P. 731-736.

5. Baheti R., Gill H. Cyber-physical Systems. *The Impact of Control Technology.* New York : IEEE Control Systems Society, 2011. P. 161-166.

6. Monostori L. Cyber-physical Production Systems: Roots, Expectations and R&D Challenges. *Procedia CIRP*. 2014. Vol. 17. P. 9-13.

7. Wan J., Tang S., Shu Z., Li D., Wang S., Imran M., Vasilakos A. Software-Defined Industrial Internet of Things in the Context of Industry 4.0. *IEEE Sensors Journal*. 2016. Vol. 16(20). P. 7373-7380.

8. Kagermann H., Wahlster W., Helbig J. Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0. Frankfurt : Acatech, 2013. 82 p.

9. Xu L. D., He W., Li S. Internet of Things in Industries: A Survey. *IEEE Transactions on Industrial Informatics*. 2014. Vol. 10(4). P. 2233-2243.

10. Atzori L., Iera A., Morabito G. The Internet of Things: A Survey. *Computer Networks.* 2010. Vol. 54(15). P. 2787-2805.

11. Gubbi J., Buyya R., Marusic S., Palaniswami M. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Generation Computer Systems*. 2013. Vol. 29(7). P. 1645-1660.

12. Al-Fuqaha A., Guizani M., Mohammadi M., Aledhari M., Ayyash M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials*. 2015. Vol. 17(4). P. 2347-2376.

13. Ashton K. That 'Internet of Things' Thing. *RFID Journal*. 2009. Vol. 22(7). P. 97-114.

14. Want R. An Introduction to RFID Technology. *IEEE Pervasive Computing*. 2006. Vol. 5(1). P. 25-33.

15. Finkenzeller K. RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication. 3rd ed. Chichester : Wiley, 2010. 462 p.

16. Dobkin D. M. The RF in RFID: Passive UHF RFID in Practice. Burlington : Newnes, 2012. 520 p.

17. Landt J. The History of RFID. *IEEE Potentials*. 2005. Vol. 24(4). P. 8-11.

18. Zhang Y., Liu S., Liu Y., Yang H. Research on Warehouse Management System Based on RFID Technology. *International Conference on Intelligent Computation Technology and Automation*. Changsha, China, 2010. Vol. 3. P. 990-993.

19. Kelepouris T., Pramatari K., Doukidis G. RFID-enabled Traceability in the Food Supply Chain. *Industrial Management & Data Systems*. 2007. Vol. 107(2). P. 183-200.

20. Angeles R. RFID Technologies: Supply-Chain Applications and Implementation Issues. *Information Systems Management*. 2005. Vol. 22(1). P. 51-65.

21. Ngai E., Moon K., Riggins F., Yi C. RFID Research: An Academic Literature Review (1995-2005) and Future Research Directions. *International Journal of Production Economics*. 2008. Vol. 112(2). P. 510-520.

22. Zhou W., Piramuthu S. Security/privacy of RFID applications in supply chains. *Journal of Organizational and End User Computing*. 2010. Vol. 22(2). P. 1-20.

23. He W., Yan G., Xu L. Developing Vehicular Data Cloud Services in the IoT Environment. *IEEE Transactions on Industrial Informatics*. 2014. Vol. 10(2). P. 1587-1595.

24. Sicari S., Rizzardi A., Grieco L., Coen-Porisini A. Security, Privacy and Trust in Internet of Things: The Road Ahead. *Computer Networks*. 2015. Vol. 76. P. 146-164.

25. Roman R., Zhou J., Lopez J. On the Features and Challenges of Security and Privacy in Distributed Internet of Things. *Computer Networks*. 2013. Vol. 57(10). P. 2266-2279.

26. Humayed A., Lin J., Li F., Luo B. Cyber-Physical Systems Security - A Survey. *IEEE Internet of Things Journal*. 2017. Vol. 4(6). P. 1802-1831.

27. Alguliyev R., Imamverdiyev Y., Sukhostat L. Cyber-Physical Systems and Their Security Issues. *Computers in Industry*. 2018. Vol. 100. P. 212-223.

28. Wolf W. Computers as Components: Principles of Embedded Computing System Design. 3rd ed. Burlington : Morgan Kaufmann, 2012. 784 p.

29. Marwedel P. Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems. 3rd ed. Cham : Springer, 2021. 458 p.

30. Stallings W. Wireless Communications and Networks. 2nd ed. Boston : Pearson, 2005. 584 p.

31. Tanenbaum A., Wetherall D. Computer Networks. 5th ed. Boston : Pearson, 2011. 960 p.

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

33. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.

34. Silberschatz A., Galvin P., Gagne G. Operating System Concepts. 10th ed. Hoboken : Wiley, 2018. 976 p.

35. Date C. An Introduction to Database Systems. 8th ed. Boston : Pearson, 2003. 1024 p.

36. Elmasri R., Navathe S. Fundamentals of Database Systems. 7th ed. Boston : Pearson, 2016. 1272 p.

37. Oracle Corporation. Java Platform, Standard Edition Documentation. URL: https://docs.oracle.com/javase/8/docs.

38. Oracle Corporation. JDBC Basics. URL: https://docs.oracle.com/javase/tutorial/jdbc/basics/.

39. MySQL Documentation Team. MySQL 8.0 Reference Manual. URL: https://dev.mysql.com/doc/.

40. Oracle Corporation. MySQL Connector/J Developer Guide. URL: https://dev.mysql.com/doc/connector-j/en/.

41. Eclipse Foundation. Jakarta Servlet Specification. URL: https://jakarta.ee/specifications/servlet/.

42. ISO/IEC 27001:2022 Information Security, Cybersecurity and Privacy Protection - Information Security Management Systems - Requirements. Geneva : ISO, 2022. 30 p.

43. NIST. Framework for Improving Critical Infrastructure Cybersecurity. Version 2.0. Gaithersburg : National Institute of Standards and Technology, 2024. 106 p.

44. IEC 62443-3-3:2013 Industrial Communication Networks - Network and System Security. Geneva : IEC, 2013. 84 p.

45. Stallings W. Cryptography and Network Security: Principles and Practice. 7th ed. Boston : Pearson, 2017. 768 p.

46. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge : MIT Press, 2016. 800 p.

47. Bishop C. Pattern Recognition and Machine Learning. New York : Springer, 2006. 738 p.

48. Cisco Systems. What Is the Internet of Things (IoT)? URL: https://www.cisco.com/c/en/us/products/se/internet-of-things/what-is-iot.html.

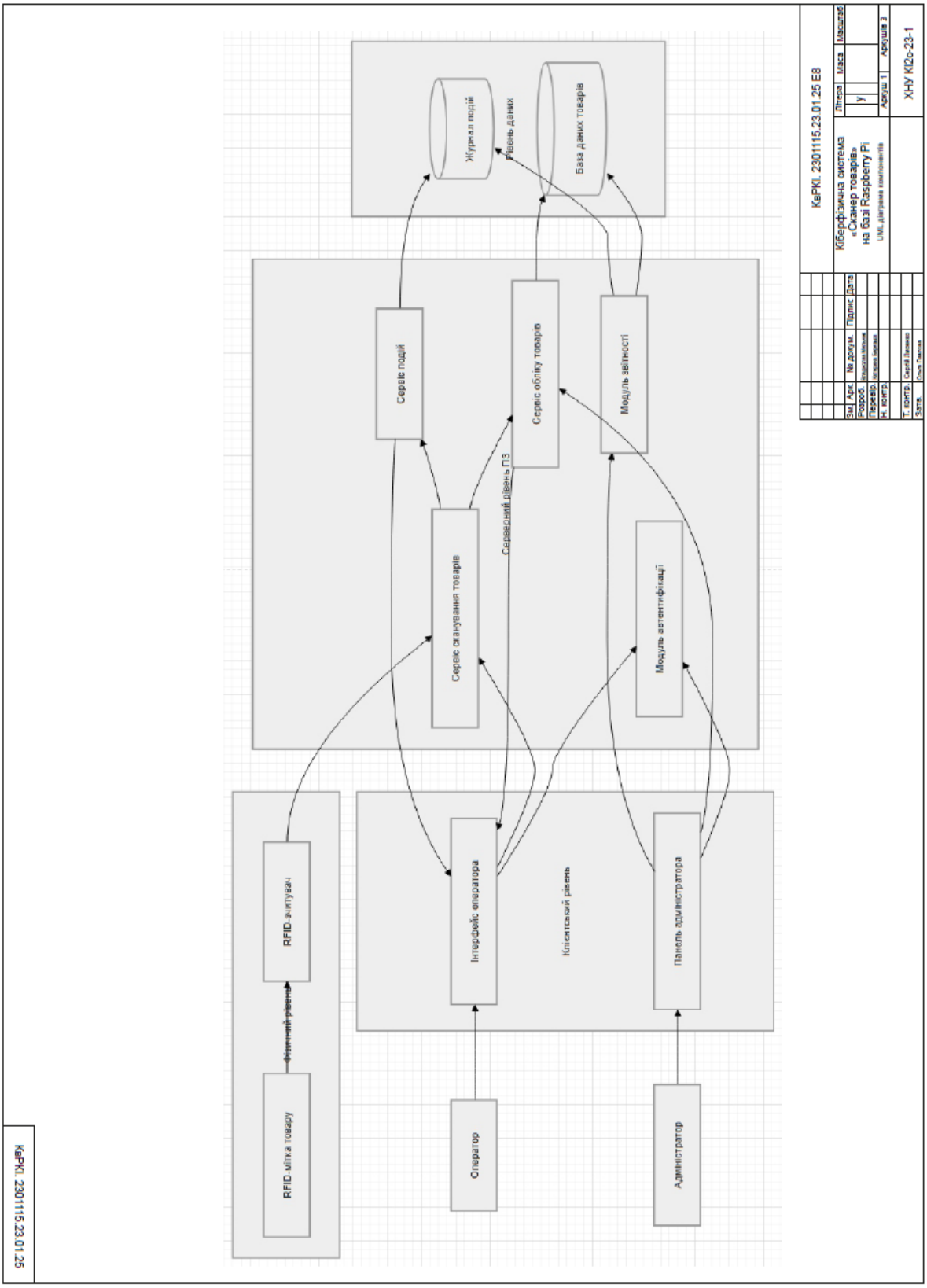
49. IBM Corporation. What are Cyber-Physical Systems? URL: https://www.ibm.com/think/topics/cyber-physical-systems.

50. Microsoft Corporation. Internet of Things (IoT). URL: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-iot/.

ДОДАТОК А

(обов’язковий)

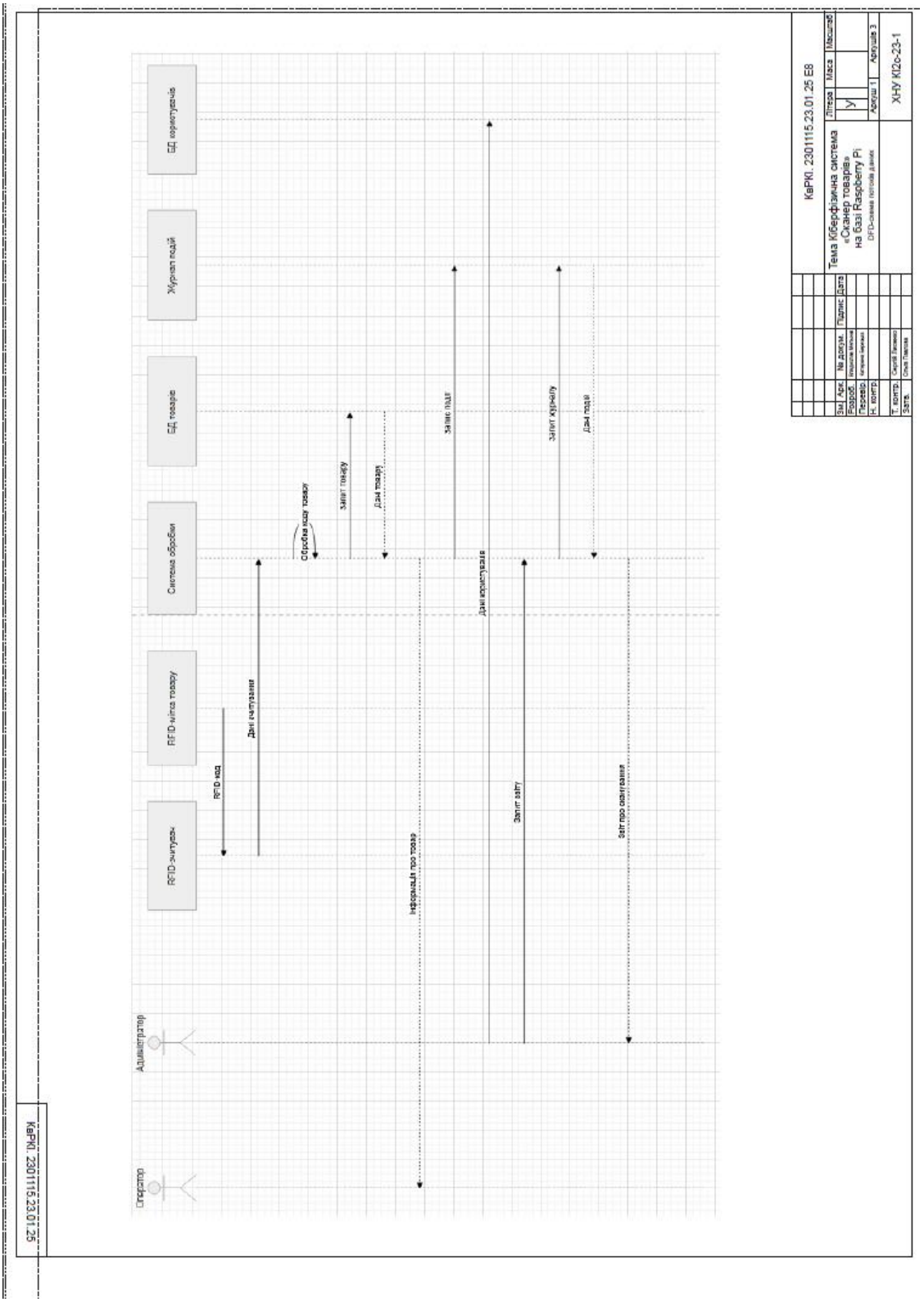
Копія креслення «Діаграма компонентів»



ДОДАТОК Б

(обов’язковий)

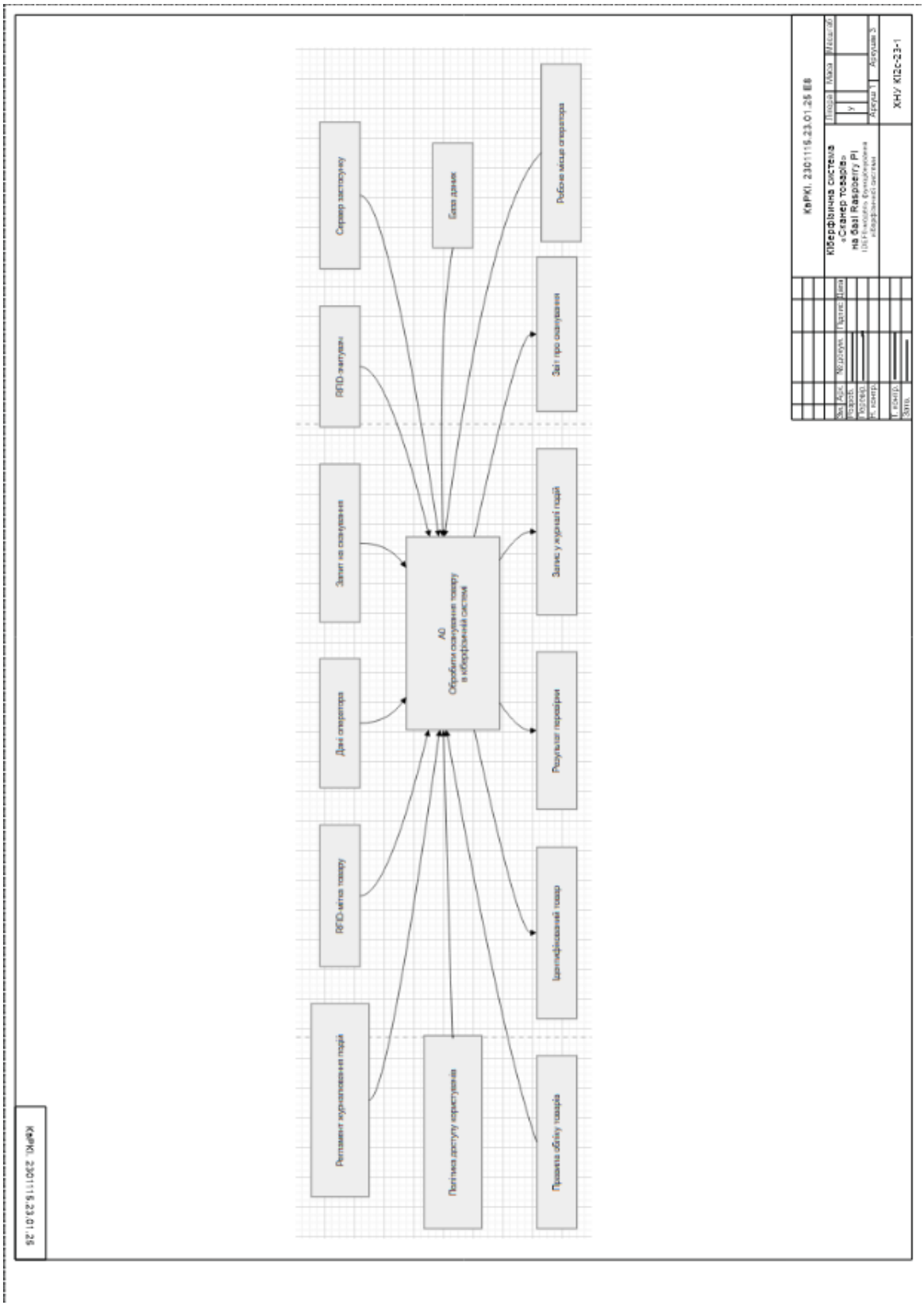
Копія креслення «Схема потоків даних»



ДОДАТОК В

(обов’язковий)

Копія креслення «IDEF0-модель функціонування кіберфізичної системи»



ДОДАТОК Г

Код програми

app/src/main/AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.
com/apk/res/android"

xmlns:tools="http://schemas.android.co
m/tools">
    <uses-permission
android:name="android.permission.INTER
NET" />
    <application
        android:allowBackup="true"

android:dataExtractionRules="@xml/data
_extraction_rules"

android:fullBackupContent="@xml/backup
_rules"

android:icon="@mipmap/ic_launcher"

android:label="@string/app_name"

android:roundIcon="@mipmap/ic_launcher
_round"
        android:supportsRtl="true"

android:theme="@style/Theme.KotlinScan
ner">
        <activity

android:name=".MainActivity"

android:configChanges="keyboardHidden|
orientation|screenSize"

android:windowSoftInputMode="stateHidd
en"
            android:exported="true">
            <intent-filter>
                <action
android:name="android.intent.action.MA
IN" />
                <category
android:name="android.intent.category.
LAUNCHER" />
                <category
android:name="android.intent.category.
HOME" />
                <category
android:name="android.intent.category.
DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

app/src/main/java/com/example/kotlinsc
anner/BannerLoader.kt

```
package com.kotlinscanner
import android.content.Context
import okhttp3.*
import org.json.JSONObject
import java.io.IOException
class BannerLoader(private val context:
Context) {
    private val client = OkHttpClient()
    private     val     baseUrl     =
"https://service.shchodnia.com/api/v3"
    fun         fetchBanners(callback:
(List<String>) -> Unit) {
        val request = Request.Builder()

.url("$baseUrl/blog/articles")
            .get()
            .build()

client.newCall(request).enqueue(object
: Callback {
            override                fun
onFailure(call: Call, e: IOException) {
                callback(emptyList())
            }
            override                fun
onResponse(call:    Call,    response:
Response) {
                if
(!response.isSuccessful) {

callback(emptyList())
                    return
                }
                val      result      =
mutableListOf<String>()
                val       json       =
JSONObject(response.body?.string()   ?:
"")
                val       data       =
json.optJSONArray("data")   ?:   return
callback(emptyList())
                for   (i   in   0   until
data.length()) {
                    val     item     =
data.getJSONObject(i)
                    val    banner    =
item.optString("banner")
                    if
(banner.isNotBlank())
result.add(banner)
                }
                callback(result)
            }
        })
    }
}
```

app/src/main/java/com/example/kotlinsc
anner/DiagonalStrikeLayout.kt

```
package com.kotlinscanner
import android.content.Context
import android.graphics.*
```

```
import android.util.AttributeSet
import android.widget.FrameLayout
class DiagonalStrikeLayout
@JvmOverloads constructor(
    context: Context, attrs:
AttributeSet? = null
) : FrameLayout(context, attrs) {
    private val paint =
Paint(Paint.ANTI_ALIAS_FLAG).apply {
        style = Paint.Style.STROKE
        color = Color.WHITE
        strokeCap = Paint.Cap.ROUND
    }
    /** Товщина лінії як частка висоти
контейнера (було 0.08f) */
    var thicknessFactor = 0.03f //
3.5% від висоти - тонка як на макеті
    /** Відносні відступи по вертикалі
(керують кутом/позицією лінії) */
    var topBias = 0.25f
    var bottomBias = 0.75f
    /** Напрямок штриха: true = справа
→ вліво, false = зліва → вправо */
    var isRightToLeft: Boolean = true
    override fun dispatchDraw(canvas:
Canvas) {
        super.dispatchDraw(canvas)
        if (width == 0 || height == 0)
return
        val w = width.toFloat()
        val h = height.toFloat()
        paint.strokeWidth = h *
thicknessFactor
        val y1 = h * topBias
        val y2 = h * bottomBias
        if (isRightToLeft) {
            canvas.drawLine(w, y1, 0f,
y2, paint) // справа → вліво
        } else {
            canvas.drawLine(0f, y1, w,
y2, paint) // зліва → вправо
        }
    }
}
```

app/src/main/java/com/example/kotlinscanner/MainActivity.kt

```
package com.kotlinscanner
import
android.content.ActivityNotFoundException
import android.content.Intent
import
android.content.pm.PackageManager
import android.net.Uri
import android.os.Build
import android.os.Bundle
import android.os.Handler
import android.os.Looper
import android.util.Log
import android.view.KeyEvent
import android.view.View
import android.view.ViewGroup
import android.widget.ImageView
import android.widget.ViewFlipper
import
androidx.appcompat.app.AppCompatActivity
import androidx.fragment.app.commit
import com.bumptech.glide.Glide
import okhttp3.*
import org.json.JSONObject
import java.io.IOException
import android.widget.Toast
import
com.google.android.play.core.appupdate.AppUpdateInfo
import
com.google.android.play.core.appupdate.AppUpdateManager
import
com.google.android.play.core.appupdate.AppUpdateManagerFactory
import
com.google.android.play.core.install.InstallStateUpdatedListener
import
com.google.android.play.core.install.model.AppUpdateType
import
com.google.android.play.core.install.model.InstallStatus
import
com.google.android.play.core.install.model.UpdateAvailability
import
com.google.android.material.snackbar.Snackbar
import
com.google.android.material.dialog.MaterialAlertDialogBuilder
class MainActivity :
AppCompatActivity() {
    private lateinit var bannerFlipper:
ViewFlipper
    private val barcodeBuffer =
StringBuilder()
    private var isProductVisible =
false
    private lateinit var bannerLoader:
BannerLoader
    private val handler =
Handler(Looper.getMainLooper())
    private val client = OkHttpClient()
    private val baseUrl =
"https://service.shchodnia.com/api/v2"
    private val FORCE_PRODUCT_BARCODE:
String? = "4820097815556"
    private lateinit var
appUpdateManager: AppUpdateManager
    private val updateRequestCode =
1001
    private var
flexibleUpdateSnackbar: Snackbar? =
null
```

```
    private val installStateListener =
InstallStateUpdatedListener { state ->
        when (state.installStatus()) {
            InstallStatus.DOWNLOADED -
> showFlexibleUpdateReady()
            InstallStatus.INSTALLED ->
{
                isUpdateDialogVisible
= false

handler.removeCallbacks(rePromptRunnab
le)
            }
            else -> Unit
        }
    }
    private val UPDATE_INTERVAL_MS = 5
* 60 * 1000L
    private val updateCheckRunnable =
object : Runnable {
        override fun run() {
            Log.d("MainActivity",
"Scheduled update check tick")
            checkForUpdateAnySource()
            handler.postDelayed(this,
UPDATE_INTERVAL_MS)
        }
    }
    private val REPROMPT_DELAY_MS = 60
* 1000L
    private val rePromptRunnable =
object : Runnable {
        override fun run() {
            if (!isFinishing &&
!isDestroyed &&
!isUpdateDialogVisible) {

checkForUpdateAnySource()
            }
        }
    }
    private val PREFS = "update_prefs"
    private val KEY_LAST_PROMPT_TS =
"last_prompt_ts"
    private val PROMPT_INTERVAL_MS =
REPROMPT_DELAY_MS
    private val bannerRefreshRunnable =
object : Runnable {
        override fun run() {
            loadBanners()
            handler.postDelayed(this,
300000)
        }
    }
    private var isUpdateDialogVisible =
false
    override fun
onCreate(savedInstanceState: Bundle?)
{

super.onCreate(savedInstanceState)

setContentView(R.layout.activity_main)

        Toast.makeText(this, "Вітаю, у
вас остання версія. Ви оновились!",
Toast.LENGTH_LONG).show()
        bannerFlipper =
findViewById(R.id.bannerFlipper)
        bannerLoader =
BannerLoader(this)
        appUpdateManager =
AppUpdateManagerFactory.create(this)
        checkForUpdateAnySource()
        loadBanners()

handler.postDelayed(bannerRefreshRunna
ble, 300000)
        enterImmersiveMode()
        FORCE_PRODUCT_BARCODE?.let {
fetchProduct(it) }
    }
    override fun onResume() {
        super.onResume()
        try {
appUpdateManager.registerListener(inst
allStateListener) } catch (_:
Exception) {}

appUpdateManager.appUpdateInfo.addOnSu
ccessListener { info ->
            try {
                if
(info.installStatus() ==
InstallStatus.DOWNLOADED) {

showFlexibleUpdateReady()
                }
            } catch (_: Exception) {}
        }

appUpdateManager.appUpdateInfo.addOnSu
ccessListener { info ->
            if
(info.updateAvailability() ==
UpdateAvailability.DEVELOPER_TRIGGERED
_UPDATE_IN_PROGRESS &&

info.isUpdateTypeAllowed(AppUpdateType
.IMMEDIATE)
            ) {
                try {

appUpdateManager.startUpdateFlowForRes
ult(
                        info,
AppUpdateType.IMMEDIATE, this,
updateRequestCode
                    )
                } catch (e: Exception)
{

Log.e("MainActivity", "Resume
IMMEDIATE update failed", e)
                }
            }
        }
```

```
handler.removeCallbacks(updateCheckRunnable)

handler.postDelayed(updateCheckRunnable, UPDATE_INTERVAL_MS)
    }
    override fun onPause() {
        try { appUpdateManager.unregisterListener(installStateListener) } catch (_: Exception) {}

flexibleUpdateSnackbar?.dismiss()

handler.removeCallbacks(updateCheckRunnable)
        super.onPause()
    }
    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {

super.onActivityResult(requestCode, resultCode, data)
        if (requestCode != updateRequestCode) return
        isUpdateDialogVisible = false
        when (resultCode) {
            RESULT_OK -> Log.i("MainActivity", "In-app update accepted")
            RESULT_CANCELED -> {
                Log.w("MainActivity", "In-app update canceled by user")
                scheduleRePrompt()
            }
            else -> {
                Log.e("MainActivity", "In-app update failed with code=$resultCode")
                scheduleRePrompt()
            }
        }
    }
    override fun onDestroy() {
        try { appUpdateManager.unregisterListener(installStateListener) } catch (_: Exception) {}

handler.removeCallbacks(rePromptRunnable)
        super.onDestroy()
    }
    private fun checkForUpdateAnySource() {
        if (isInstalledFromPlayStore()) {
            checkForAppUpdate()
        } else {
            maybePromptToOpenPlay()
        }
    }
    private fun checkForAppUpdate() {
        appUpdateManager.appUpdateInfo
            .addOnSuccessListener { info ->
                if (info.updateAvailability() == UpdateAvailability.UPDATE_AVAILABLE) {

showUpdateOffer(info)
                } else {

Log.d("MainActivity", "No update available")
                }
            }
            .addOnFailureListener { e ->
                Log.e("MainActivity", "appUpdateInfo failed", e)
                scheduleRePrompt()
            }
    }
    private fun isInstalledFromPlayStore(): Boolean {
        return try {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
                val src = packageManager.getInstallSourceInfo(packageName)

src.installingPackageName == "com.android.vending"
            } else {

@Suppress("DEPRECATION")

packageManager.getInstallerPackageName(packageName) == "com.android.vending"
            }
        } catch (_: Exception) {
            false
        }
    }
    private fun showUpdateOffer(info: AppUpdateInfo) {
        if (isUpdateDialogVisible) return
        val localCode = localVersionCode()
        val localName = localVersionName()
        val remoteCode = try { info.availableVersionCode() } catch (_: Throwable) { null }
        val updateType = when {

info.isUpdateTypeAllowed(AppUpdateType.IMMEDIATE) -> AppUpdateType.IMMEDIATE

info.isUpdateTypeAllowed(AppUpdateType.FLEXIBLE) -> AppUpdateType.FLEXIBLE
```

```
            else -> null
        } ?: run {
            Log.i("MainActivity",
"Update   available   but   type   not
allowed")
            scheduleRePrompt()
            return
        }
        val msg = buildString {
            append("Доступна      нова
версія.\n")
            append("Встановлена:
$localName ($localCode)\n")
            if  (remoteCode  !=  null)
append("Доступна        у        Play:
($remoteCode)\n")
            append("\nОновити зараз?")
        }
        var startedFlow = false
        isUpdateDialogVisible = true
        val          dialog          =
MaterialAlertDialogBuilder(this)
            .setTitle("Доступне
оновлення")
            .setMessage(msg)
            .setCancelable(true)    //
дозволяємо відхилити/ігнорувати

.setNegativeButton("Пізніше") { _, _ -
>
                isUpdateDialogVisible
= false
                scheduleRePrompt()
            }

.setPositiveButton("Оновити") { _, _ -
>
                try {
                    startedFlow = true

handler.removeCallbacks(rePromptRunnab
le)

appUpdateManager.startUpdateFlowForRes
ult(
                        info,
updateType, this, updateRequestCode
                    )
                } catch (e: Exception)
{

isUpdateDialogVisible = false

Log.e("MainActivity",   "Update   flow
failed to start", e)
                    scheduleRePrompt()
                }
            }
            .create()

dialog.setCanceledOnTouchOutside(true)
        dialog.setOnCancelListener {
            isUpdateDialogVisible    =
false
            if          (!startedFlow)
scheduleRePrompt()
        }
        dialog.setOnDismissListener {
            isUpdateDialogVisible    =
false
            if          (!startedFlow)
scheduleRePrompt()
        }
        dialog.show()
    }
    private                           fun
maybePromptToOpenPlay() {
        val          prefs          =
getSharedPreferences(PREFS,
MODE_PRIVATE)
        val         lastShown        =
prefs.getLong(KEY_LAST_PROMPT_TS, 0L)
        val            now           =
System.currentTimeMillis()
        if   (now   -   lastShown   <
PROMPT_INTERVAL_MS) {
            scheduleRePrompt()
            return
        }
        if     (isUpdateDialogVisible)
return
        isUpdateDialogVisible = true
        val          dialog          =
MaterialAlertDialogBuilder(this)
            .setTitle("Перевірка
оновлень")
            .setMessage("Відкрити
сторінку застосунку в Google Play, щоб
перевірити наявність оновлення?")
            .setCancelable(true)

.setNegativeButton("Пізніше") { _, _ -
>
                isUpdateDialogVisible
= false
                scheduleRePrompt()
            }

.setPositiveButton("Відкрити") { _, _ -
>
                isUpdateDialogVisible
= false
                openStoreOrWeb(

"market://details?id=$packageName",

"https://play.google.com/store/apps/de
tails?id=$packageName"
                )
                scheduleRePrompt()
            }
            .create()

dialog.setCanceledOnTouchOutside(true)
        dialog.setOnCancelListener {
```

```
            isUpdateDialogVisible    =
false
            scheduleRePrompt()
        }
        dialog.setOnDismissListener {
            isUpdateDialogVisible    =
false
            scheduleRePrompt()
        }
        dialog.show()

prefs.edit().putLong(KEY_LAST_PROMPT_T
S, now).apply()
    }
    private fun scheduleRePrompt() {

handler.removeCallbacks(rePromptRunnab
le)

handler.postDelayed(rePromptRunnable,
REPROMPT_DELAY_MS)
    }
    private                          fun
openStoreOrWeb(playUrl:        String,
fallbackUrl: String) {
        try {

startActivity(Intent(Intent.ACTION_VIE
W, Uri.parse(playUrl)))
        }           catch           (_:
ActivityNotFoundException) {
            try {

startActivity(Intent(Intent.ACTION_VIE
W, Uri.parse(fallbackUrl)))
            } catch (e: Exception) {
                Log.e("MainActivity",
"Cannot open store/web: ${e.message}")
            }
        }
    }
    private fun localVersionCode(): Int
{
        return try {
            val      pi      =      if
(Build.VERSION.SDK_INT >= 33) {

packageManager.getPackageInfo(packageN
ame,
PackageManager.PackageInfoFlags.of(0))
            } else {

@Suppress("DEPRECATION")

packageManager.getPackageInfo(packageN
ame, 0)
            }
            if   (Build.VERSION.SDK_INT
>= 28) pi.longVersionCode.toInt()
            else
@Suppress("DEPRECATION")
pi.versionCode
        } catch (e: Exception) { 0 }
    }
    private   fun   localVersionName():
String {
        return try {
            val      pi      =      if
(Build.VERSION.SDK_INT >= 33) {

packageManager.getPackageInfo(packageN
ame,
PackageManager.PackageInfoFlags.of(0))
            } else {

@Suppress("DEPRECATION")

packageManager.getPackageInfo(packageN
ame, 0)
            }
            pi.versionName ?: ""
        } catch (e: Exception) { "" }
    }
    private                          fun
showFlexibleUpdateReady() {
        val           root           =
findViewById<View>(android.R.id.conten
t)

flexibleUpdateSnackbar?.dismiss()
        flexibleUpdateSnackbar       =
Snackbar
            .make(root,      "Оновлення
завантажено.    Перезапустити,    щоб
застосувати?",
Snackbar.LENGTH_INDEFINITE)

.setAction("Перезапустити") {
                try                   {
appUpdateManager.completeUpdate()    }
catch (_: Exception) {}
            }
        flexibleUpdateSnackbar?.show()
    }
    private fun loadBanners() {
        bannerLoader.fetchBanners    {
banners ->
            runOnUiThread {
                if (banners.isEmpty())
{

Log.w("MainActivity",    "Банери    не
завантажено")

return@runOnUiThread
                }

bannerFlipper.removeAllViews()
                for (url in banners) {
                    val   imageView   =
ImageView(this).apply {
                        layoutParams =
ViewGroup.LayoutParams(

ViewGroup.LayoutParams.MATCH_PARENT,
```

```
ViewGroup.LayoutParams.MATCH_PARENT
                        )
                        scaleType    =
ImageView.ScaleType.FIT_XY
                    }
                    Glide.with(this)
                        .load(url)

.placeholder(R.drawable.placeholder)

.into(imageView)

bannerFlipper.addView(imageView)
                }

bannerFlipper.startFlipping()
            }
        }
    }
    override                          fun
dispatchKeyEvent(event:     KeyEvent):
Boolean {
        if  (FORCE_PRODUCT_BARCODE  !=
null)                             return
super.dispatchKeyEvent(event)
        if       (event.action       ==
KeyEvent.ACTION_DOWN) {
            val     unicodeChar      =
event.unicodeChar
            if (unicodeChar != 0) {
                val        ch        =
unicodeChar.toChar()
                if (ch == '\n' || ch ==
'\r') {
                    val    barcode    =
barcodeBuffer.toString()

barcodeBuffer.clear()

fetchProduct(barcode)
                } else {

barcodeBuffer.append(ch)
                }
            }
        }
        return
super.dispatchKeyEvent(event)
    }
    private  fun  fetchProduct(barcode:
String) {
        val            url           =
"$baseUrl/products/$barcode"
        val          request         =
Request.Builder().url(url).get().build
()

client.newCall(request).enqueue(object
: Callback {
            override                fun
onFailure(call: Call, e: IOException) {
                runOnUiThread {
showProductFragment(barcode,       "",
"Товар не знайдено", 0.0, null, "")
                }
            }
            override               fun
onResponse(call:    Call,    response:
Response) {
                if
(!response.isSuccessful) {
                    runOnUiThread {

showProductFragment(barcode,       "",
"Товар не знайдено", 0.0, null, "")
                    }
                    return
                }
                val      bodyStr      =
response.body?.string() ?: return
                val       json       =
JSONObject(bodyStr)
                val       data       =
json.optJSONObject("data") ?: return
                val        sku       =
data.optString("sku", "")
                val       title      =
data.optString("title", "Товар")
                val       price      =
data.optDouble("price", 0.0)
                val         promoPrice:
Double? =
                    if
(!data.isNull("promo_price"))
data.optDouble("promo_price")     else
null
                val       image      =
data.optString("image", "")
                val  fullImageUrl  =  if
(image.startsWith("http"))  image  else
"$baseUrl$image"
                runOnUiThread {

showProductFragment(barcode,      sku,
title, price, promoPrice, fullImageUrl)
                }
            }
        })
    }
    private fun showProductFragment(
        barcode: String,
        sku: String,
        title: String,
        price: Double,
        promoPrice: Double?,
        image: String
    ) {
        val         isForced         =
FORCE_PRODUCT_BARCODE?.equals(barcode)
== true
        if (isProductVisible) {

supportFragmentManager.findFragmentByT
ag("ProductFragment")?.let {
```

```
supportFragmentManager.beginTransactio
n().remove(it).commit()
            }
        }
        bannerFlipper.stopFlipping()
        if                    (isForced)
bannerFlipper.visibility  =  View.GONE
// ← ДОДАТИ
        isProductVisible = true
        supportFragmentManager.commit
{
            replace(
                R.id.rootContainer,

ProductFragment.newInstance(barcode,
sku, title, price, promoPrice, image),
                "ProductFragment"
            )
        }
        if (!isForced) {
            handler.postDelayed({

supportFragmentManager.findFragmentByT
ag("ProductFragment")?.let {

supportFragmentManager.beginTransactio
n().remove(it).commit()
                }

bannerFlipper.startFlipping()

bannerFlipper.visibility             =
View.VISIBLE
                isProductVisible     =
false
            }, 10000)
        }
    }
    private fun enterImmersiveMode() {

window.decorView.systemUiVisibility =

View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY
or

View.SYSTEM_UI_FLAG_LAYOUT_STABLE or

View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGA
TION or

View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
or

View.SYSTEM_UI_FLAG_HIDE_NAVIGATION or

View.SYSTEM_UI_FLAG_FULLSCREEN
    }
    override                          fun
onWindowFocusChanged(hasFocus:
Boolean) {

super.onWindowFocusChanged(hasFocus)
        if                    (hasFocus)
enterImmersiveMode()
    }
}
```

app/src/main/java/com/example/kotlinsc
anner/ProductFragment.kt

```
package com.kotlinscanner
import android.graphics.Color
import android.graphics.Paint
import android.graphics.Typeface
import android.os.Bundle
import android.text.TextUtils
import android.util.TypedValue
import android.view.*
import android.widget.*
import
androidx.appcompat.content.res.AppComp
atResources
import
androidx.appcompat.widget.AppCompatTex
tView
import
androidx.constraintlayout.widget.Const
raintLayout
import
androidx.constraintlayout.widget.Const
raintLayout.LayoutParams.PARENT_ID
import
androidx.constraintlayout.widget.Const
raintLayout.LayoutParams.WRAP_CONTENT
import
androidx.constraintlayout.widget.Guide
line
import
androidx.core.content.res.ResourcesCom
pat
import androidx.core.os.bundleOf
import androidx.core.view.doOnLayout
import
androidx.core.widget.TextViewCompat
import androidx.fragment.app.Fragment
import com.bumptech.glide.Glide
import
com.google.android.material.card.Mater
ialCardView
import kotlin.math.floor
import kotlin.math.max
import kotlin.math.min
import kotlin.math.roundToInt
class ProductFragment : Fragment() {
    private object TypeScale {
        const val PRICE_MAJOR    = 240f
        const val PRICE_MINOR    = 100f
        const val PRICE_CURRENCY = 70f
        const val TITLE          = 50f
        const val ARTICLE        = 32f
    }
    private val OLD_SCALE = 2f / 3f
    private data class PricePack(
        val row: LinearLayout,
        val main: TextView,
        val col: LinearLayout,
```

```
        val minor: TextView,
        val curr: TextView,
        val scale: Float
    )
    private lateinit var newPack: PricePack
    private lateinit var oldPack: PricePack
    private lateinit var titleView: TextView
    private lateinit var priceMainView: TextView
    private lateinit var priceMinorView: TextView
    private lateinit var priceCurrencyView: TextView
    private lateinit var productImageView: ImageView
    private lateinit var priceCloud: FrameLayout
    private lateinit var cloudBg: ImageView
    private lateinit var priceColumn: LinearLayout
    private lateinit var priceRow: LinearLayout
    private lateinit var minorColumn: LinearLayout
    private lateinit var oldPriceWrap: DiagonalStrikeLayout
    private lateinit var oldPriceRow: LinearLayout
    private lateinit var oldMainView: TextView
    private lateinit var oldMinorView: TextView
    private lateinit var oldCurrencyView: TextView
    private lateinit var oldMinorColumn: LinearLayout
    private lateinit var articleView: AppCompatTextView
    private lateinit var logo: ImageView
    private lateinit var card: MaterialCardView
    private lateinit var imageFrame: MaterialCardView
    private lateinit var split: Guideline
    private fun px(dp: Int): Int = (dp * resources.displayMetrics.density).toInt()
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
    ): View {
        val ctx = requireContext()
        fun dp(v: Int) = (v * ctx.resources.displayMetrics.density).toInt()
        fun safeFont(fontRes: Int, fallbackStyle: Int = Typeface.BOLD): Typeface =
            try { ResourcesCompat.getFont(ctx, fontRes) ?: Typeface.defaultFromStyle(fallbackStyle) }
            catch (_: Exception) { Typeface.defaultFromStyle(fallbackStyle) }
        val fontPrice = safeFont(R.font.shchodnya_bold, Typeface.BOLD)
        val fontTitle = safeFont(R.font.shchodnya_bold, Typeface.BOLD)
        val fontArticle = safeFont(R.font.shchodnya_bold, Typeface.BOLD)
        val root = ConstraintLayout(ctx).apply {
            layoutParams = ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT, ViewGroup.LayoutParams.MATCH_PARENT)
        }
        root.background = AppCompatResources.getDrawable(ctx, R.drawable.bg_pattern)
        logo = ImageView(ctx).apply {
            id = View.generateViewId()
            setImageResource(R.drawable.shchodnya_logo)
            adjustViewBounds = true
            maxWidth = dp(280)
            maxHeight = dp(120)
            contentDescription = "Щодня"
        }
        root.addView(logo, ConstraintLayout.LayoutParams(WRAP_CONTENT, WRAP_CONTENT).apply {
            topToTop = PARENT_ID
            endToEnd = PARENT_ID
            topMargin = dp(16)
            marginEnd = dp(24)
        })
        logo.bringToFront(); logo.z = 10f
        split = Guideline(ctx).apply { id = View.generateViewId() }
        root.addView(split, ConstraintLayout.LayoutParams(WRAP_CONTENT, WRAP_CONTENT).apply {
            orientation = ConstraintLayout.LayoutParams.VERTICAL
            guidePercent = 0.45f
        })
        card = MaterialCardView(ctx).apply {
            id = View.generateViewId()
```

```
            radius = dp(24).toFloat()
            cardElevation            =
dp(6).toFloat()

setCardBackgroundColor(Color.WHITE)
        }
        root.addView(card,
ConstraintLayout.LayoutParams(0,
0).apply {
            startToEnd = split.id
            endToEnd   = PARENT_ID
            topToTop   = PARENT_ID
            bottomToBottom = PARENT_ID
            verticalBias = 1f
            val edge = dp(24)
            marginStart = dp(24)
            marginEnd   = edge
            bottomMargin = edge
            dimensionRatio = "1:1"
        })
        imageFrame                     =
MaterialCardView(ctx).apply {
            id = View.generateViewId()

setCardBackgroundColor(Color.TRANSPARE
NT)
            radius = 0f
            cardElevation = 0f
            preventCornerOverlap     =
false
            useCompatPadding = false
            setContentPadding(0, 0, 0,
0)
        }
        card.addView(imageFrame,
FrameLayout.LayoutParams(

FrameLayout.LayoutParams.MATCH_PARENT,

FrameLayout.LayoutParams.MATCH_PARENT
        ).apply { val m = dp(10);
setMargins(m, m, m, m) })
        val      imageContainer      =
ConstraintLayout(ctx).apply  {  id  =
View.generateViewId() }

imageFrame.addView(imageContainer,
FrameLayout.LayoutParams(

FrameLayout.LayoutParams.MATCH_PARENT,

FrameLayout.LayoutParams.MATCH_PARENT
        ))
        val          frameBg          =
ImageView(ctx).apply {
            id = View.generateViewId()
            adjustViewBounds = true
            scaleType                  =
ImageView.ScaleType.FIT_XY

setImageResource(R.drawable.main_place
holder)
        }

imageContainer.addView(frameBg,
ConstraintLayout.LayoutParams(0,
0).apply {
            startToStart = PARENT_ID
            endToEnd     = PARENT_ID
            topToTop     = PARENT_ID
            bottomToBottom = PARENT_ID
        })
        val         lineGuide         =
Guideline(ctx).apply     {     id     =
View.generateViewId() }

imageContainer.addView(lineGuide,
ConstraintLayout.LayoutParams(WRAP_CON
TENT, WRAP_CONTENT).apply {
            orientation                =
ConstraintLayout.LayoutParams.HORIZONT
AL
            guidePercent = 786f / 861f
        })
        productImageView               =
ImageView(ctx).apply {
            id = View.generateViewId()
            scaleType                  =
ImageView.ScaleType.FIT_CENTER
            adjustViewBounds = true

setBackgroundColor(Color.TRANSPARENT)
            setPadding(dp(12), dp(12),
dp(12), 0)
        }

imageContainer.addView(productImageVie
w,    ConstraintLayout.LayoutParams(0,
0).apply {
            startToStart = PARENT_ID
            endToEnd     = PARENT_ID
            topToTop     = PARENT_ID
            bottomToTop  = lineGuide.id
            bottomMargin = dp(10)
        })
        articleView                    =
AppCompatTextView(ctx).apply {
            id = View.generateViewId()

setTextColor(Color.parseColor("#143A85
"))

setTextSize(TypedValue.COMPLEX_UNIT_SP
, TypeScale.ARTICLE)
            maxLines = 1
            ellipsize                  =
TextUtils.TruncateAt.END
            textAlignment              =
View.TEXT_ALIGNMENT_CENTER
            gravity = Gravity.CENTER
            includeFontPadding = false
            setLineSpacing(0f, 1f)
            typeface = fontArticle
        }

imageContainer.addView(articleView,
```

```
ConstraintLayout.LayoutParams(0,
WRAP_CONTENT).apply {
            startToStart = PARENT_ID
            endToEnd = PARENT_ID
            topToBottom = lineGuide.id
            bottomToBottom = PARENT_ID
            verticalBias = 0.5f
        })
        priceCloud                    =
FrameLayout(ctx).apply {
            id = View.generateViewId()
            clipChildren = false
            clipToPadding = false
        }
        val       cloudDrawable       =
AppCompatResources.getDrawable(ctx,
R.drawable.price_bg)
        val      cw      =      max(1,
cloudDrawable?.intrinsicWidth ?: 634)
        val      ch      =      max(1,
cloudDrawable?.intrinsicHeight ?: 280)
        root.addView(priceCloud,
ConstraintLayout.LayoutParams(0,
0).apply {
            startToStart   = PARENT_ID
            endToStart     = card.id
            topToTop       = card.id
            bottomToBottom = card.id
            verticalBias   = 0.5f
            marginStart    = dp(24)
            marginEnd      = dp(28)
            topMargin      = dp(16)
            dimensionRatio             =
"H,${cw}:${ch}"
            horizontalBias = 0f

matchConstraintPercentWidth = 0.40f
        })
        cloudBg = ImageView(ctx).apply
{
            scaleType                  =
ImageView.ScaleType.FIT_XY

setImageResource(R.drawable.price_bg)
        }
        priceCloud.addView(cloudBg,
FrameLayout.LayoutParams(

FrameLayout.LayoutParams.MATCH_PARENT,

FrameLayout.LayoutParams.MATCH_PARENT
        ))
        priceColumn                    =
LinearLayout(ctx).apply {
            orientation                =
LinearLayout.VERTICAL
            gravity = Gravity.START or
Gravity.CENTER_VERTICAL
            clipChildren = false
            clipToPadding = false
        }
        priceCloud.addView(priceColumn,
FrameLayout.LayoutParams(

FrameLayout.LayoutParams.MATCH_PARENT,
FrameLayout.LayoutParams.MATCH_PARENT
        ))
        priceRow                       =
LinearLayout(ctx).apply {
            orientation                =
LinearLayout.HORIZONTAL
            gravity = Gravity.START or
Gravity.CENTER_VERTICAL
            isBaselineAligned = false
            clipChildren = false
            clipToPadding = false
        }
        priceColumn.addView(
            priceRow,
            LinearLayout.LayoutParams(

LinearLayout.LayoutParams.WRAP_CONTENT
,
                0 // тягнемося на всю
висоту за рахунок ваги
            ).apply {
                weight = 1f
                gravity                =
Gravity.START
            }
        )
        priceMainView                  =
AppCompatTextView(ctx).apply {
            setTextColor(Color.WHITE)
            includeFontPadding = false
            setLineSpacing(0f, 1f)
            text = "0"
            maxLines = 1
            typeface = fontPrice
        }
        minorColumn                    =
LinearLayout(ctx).apply {
            orientation                =
LinearLayout.VERTICAL
            gravity                    =
Gravity.CENTER_HORIZONTAL           or
Gravity.CENTER_VERTICAL
            setPadding(px(12), 0, 0, 0)
            isBaselineAligned = true
            clipChildren = false
            clipToPadding = false
        }
        priceMinorView                 =
AppCompatTextView(ctx).apply {
            setTextColor(Color.WHITE)
            includeFontPadding = false
            text = ".00"
            maxLines = 1
            gravity                    =
Gravity.CENTER_HORIZONTAL
            textAlignment              =
TextView.TEXT_ALIGNMENT_CENTER
            typeface = fontPrice
```

```
        }
        priceCurrencyView               =
AppCompatTextView(ctx).apply {
            setTextColor(Color.WHITE)
            includeFontPadding = false
            text = "грн"
            maxLines = 1
            gravity                     =
Gravity.CENTER_HORIZONTAL
            textAlignment               =
TextView.TEXT_ALIGNMENT_CENTER
            typeface = fontPrice
        }

priceRow.addView(priceMainView)

minorColumn.addView(priceMinorView)

minorColumn.addView(priceCurrencyView)
        priceRow.addView(minorColumn)
        oldPriceWrap                    =
DiagonalStrikeLayout(ctx).apply {
            id = View.generateViewId()
            //
thicknessFactor/topBias/bottomBias
можна підлаштувати за потреби
            // thicknessFactor = 0.08f;
topBias = 0.24f; bottomBias = 0.76f
            clipChildren = false
            clipToPadding = false
        }
        root.addView(oldPriceWrap,
ConstraintLayout.LayoutParams(WRAP_CON
TENT, WRAP_CONTENT).apply {
            endToEnd                    =
priceCloud.id
            bottomToTop                 =
priceCloud.id
            bottomMargin = px(2)
        })
        oldPriceRow                     =
LinearLayout(ctx).apply {
            id = View.generateViewId()
            orientation                 =
LinearLayout.HORIZONTAL
            gravity  =  Gravity.END  or
Gravity.CENTER_VERTICAL
            isBaselineAligned = false
            clipChildren = false
            clipToPadding = false
        }

oldPriceWrap.addView(oldPriceRow)
        oldMainView                     =
AppCompatTextView(ctx).apply {
            setTextColor(Color.WHITE)
            includeFontPadding = false
            maxLines = 1
            typeface = fontPrice

setTextSize(TypedValue.COMPLEX_UNIT_SP
, TypeScale.PRICE_MAJOR * OLD_SCALE)
        }

oldPriceRow.addView(oldMainView)
        oldMinorColumn                  =
LinearLayout(ctx).apply {
            orientation                 =
LinearLayout.VERTICAL
            gravity                     =
Gravity.CENTER_HORIZONTAL           or
Gravity.CENTER_VERTICAL
            setPadding(px(12), 0, 0, 0)
            isBaselineAligned = true
            clipChildren = false
            clipToPadding = false
        }
        oldMinorView                    =
AppCompatTextView(ctx).apply {
            setTextColor(Color.WHITE)
            includeFontPadding = false
            maxLines = 1
            gravity                     =
Gravity.CENTER_HORIZONTAL
            textAlignment               =
TextView.TEXT_ALIGNMENT_CENTER
            typeface = fontPrice

setTextSize(TypedValue.COMPLEX_UNIT_SP
, TypeScale.PRICE_MINOR * OLD_SCALE)
        }
        oldCurrencyView                 =
AppCompatTextView(ctx).apply {
            setTextColor(Color.WHITE)
            includeFontPadding = false
            maxLines = 1
            gravity                     =
Gravity.CENTER_HORIZONTAL
            textAlignment               =
TextView.TEXT_ALIGNMENT_CENTER
            typeface = fontPrice
            text = "грн"

setTextSize(TypedValue.COMPLEX_UNIT_SP
, TypeScale.PRICE_CURRENCY * OLD_SCALE)
        }

oldMinorColumn.addView(oldMinorView)

oldMinorColumn.addView(oldCurrencyView
)

oldPriceRow.addView(oldMinorColumn)
        newPack  =  PricePack(priceRow,
priceMainView,            minorColumn,
priceMinorView, priceCurrencyView, 1f)
        oldPack                         =
PricePack(oldPriceRow,    oldMainView,
oldMinorColumn,          oldMinorView,
oldCurrencyView, OLD_SCALE)
        titleView                       =
AppCompatTextView(ctx).apply {
            setTextColor(Color.WHITE)
            includeFontPadding = false
            setLineSpacing(0f, 1f)
```

```
            setPadding(px(8),       0,
px(8), 0)
            setShadowLayer(6f, 0f, 0f,
Color.parseColor("#80000000"))
            maxLines = 2
            ellipsize                =
TextUtils.TruncateAt.END
            typeface = fontTitle
        }
        val          titleLp          =
ConstraintLayout.LayoutParams(0,
WRAP_CONTENT).apply {
            startToStart = PARENT_ID
            endToStart   = split.id
            topToBottom              =
priceCloud.id
            bottomToBottom = card.id
            marginStart  = px(24)
            topMargin    = px(8)
        }
        root.addView(titleView,
titleLp)

TextViewCompat.setTextAppearance(title
View,
R.style.TextAppearance_Shchodnya_Title
)

TextViewCompat.setTextAppearance(price
MainView,
R.style.TextAppearance_Shchodnya_Price
Major)

TextViewCompat.setTextAppearance(price
MinorView,
R.style.TextAppearance_Shchodnya_Price
Minor)

TextViewCompat.setTextAppearance(price
CurrencyView,
R.style.TextAppearance_Shchodnya_Price
Minor)

TextViewCompat.setTextAppearance(artic
leView,
R.style.TextAppearance_Shchodnya_Artic
le)
        titleView.typeface           =
fontTitle
        priceMainView.typeface       =
fontPrice
        priceMinorView.typeface      =
fontPrice
        priceCurrencyView.typeface   =
fontPrice
        articleView.typeface         =
fontArticle
        root.doOnLayout {

applyResponsiveLayout(root)
            fitPriceToCloud()
        }
        priceCloud.doOnLayout        {
fitPriceToCloud() }

priceCloud.addOnLayoutChangeListener {
_,  _,  _,  _,  _,  _,  _,  _,  _  ->
fitPriceToCloud() }
        return root
    }
    private data class Tuning(
        val   leftPct:   Float,   val
rightPct: Float, val topPct: Float, val
bottomPct: Float,
        val  safeWFactor:  Float,  val
hiRowFactor: Float,
        val  overdrivePx:  Float,  val
minorShift:  Float,  val  currOverlap:
Float,
        val minorPadDp: Int
    )
    private   fun   tuningForDigits(d:
Int): Tuning = when {
        d <= 2 -> Tuning(
            leftPct = 0.16f, rightPct =
0.10f,
            topPct = 0.12f, bottomPct =
0.10f,
            safeWFactor = 0.99f,    //
було 0.97
            hiRowFactor = 0.94f,    //
було 0.92
            overdrivePx = 40f,
            minorShift = 0.28f,
            currOverlap = 0.22f,
            minorPadDp = 8
        )
        d == 3 -> Tuning(
            leftPct = 0.15f, rightPct =
0.12f,
            topPct = 0.13f, bottomPct =
0.11f,
            safeWFactor = 0.99f,
            hiRowFactor = 0.92f,
            overdrivePx = 30f,
            minorShift = 0.26f,
            currOverlap = 0.20f,
            minorPadDp = 8
        )
        else -> Tuning(
            leftPct = 0.16f, rightPct =
0.14f,
            topPct = 0.14f, bottomPct =
0.12f,
            safeWFactor = 0.985f,   //
було 0.94
            hiRowFactor = 0.90f,    //
було 0.86
            overdrivePx = 26f,
            minorShift = 0.24f,
            currOverlap = 0.20f,
            minorPadDp = 6
        )
    }
    private fun computeRowWidthPx(
```

```
        majorSp: Float,
        minorText: String,
        currencyText: String,
        t: Tuning
    ): Float {
        val            dm            =
resources.displayMetrics
        val sd = dm.scaledDensity
        val          majorPx          =
TypedValue.applyDimension(

TypedValue.COMPLEX_UNIT_SP,
            majorSp + t.overdrivePx /
sd,
            dm
        )
        val   minorSp   =   majorSp   *
(TypeScale.PRICE_MINOR                /
TypeScale.PRICE_MAJOR)
        val   currSp     =   majorSp   *
(TypeScale.PRICE_CURRENCY             /
TypeScale.PRICE_MAJOR)
        val        minorPxSize        =
TypedValue.applyDimension(TypedValue.C
OMPLEX_UNIT_SP, minorSp, dm)
        val     currPxSize            =
TypedValue.applyDimension(TypedValue.C
OMPLEX_UNIT_SP, currSp, dm)
        val          pMajor           =
Paint(priceMainView.paint).apply      {
textSize = majorPx }
        val          pMinor           =
Paint(priceMinorView.paint).apply     {
textSize = minorPxSize }
        val       pCurr               =
Paint(priceCurrencyView.paint).apply  {
textSize = currPxSize }
        val       minMinorWidth       =
pMinor.measureText(".00")
        val        minorWidth         =
max(pMinor.measureText(minorText),
minMinorWidth)
        val       currencyWidth       =
pCurr.measureText(currencyText)
        val       padLeftMinor        =
px(t.minorPadDp).toFloat()
        val         mainWidth         =
pMajor.measureText(priceMainView.text?
.toString() ?: "0")
        val        rightColumn        =
max(minorWidth, currencyWidth)
        return mainWidth + padLeftMinor
+ rightColumn
    }
    private     fun     applyPack(pack:
PricePack,   baseMajorSp:   Float,   t:
Tuning) {
        val            dm             =
resources.displayMetrics
        val sd = dm.scaledDensity
        val scale = pack.scale
        val  majorSp  =  baseMajorSp  *
scale
        val   minorSp   =   majorSp   *
(TypeScale.PRICE_MINOR                /
TypeScale.PRICE_MAJOR)
        val   currSp     =   majorSp   *
(TypeScale.PRICE_CURRENCY             /
TypeScale.PRICE_MAJOR)

pack.main.setTextSize(TypedValue.COMPL
EX_UNIT_SP, majorSp + (t.overdrivePx /
sd) * scale)

pack.minor.setTextSize(TypedValue.COMP
LEX_UNIT_SP, minorSp)

pack.curr.setTextSize(TypedValue.COMPL
EX_UNIT_SP,  currSp)

pack.col.setPadding(px(t.minorPadDp),
0, 0, 0)
        val          pMinor           =
Paint(pack.minor.paint)
        val         minMinor          =
pMinor.measureText(".00")
        pack.minor.minWidth           =
(minMinor                             +
TypedValue.applyDimension(TypedValue.C
OMPLEX_UNIT_DIP, 1f, dm)).toInt()
        val          currPx           =
TypedValue.applyDimension(TypedValue.C
OMPLEX_UNIT_SP, currSp, dm)
        val  currLift  =  -(currPx  *
t.currOverlap)
        (pack.curr.layoutParams     as
LinearLayout.LayoutParams).topMargin =
0
        pack.curr.translationY       =
currLift
        pack.main.translationY       =
currLift / 2f
        val    mainBoxHeight    =    if
(pack.main.measuredHeight > 0) {
            pack.main.measuredHeight
        } else {
            pack.main.lineHeight
        }
        (pack.col.layoutParams      as
LinearLayout.LayoutParams).apply {
            width                     =
LinearLayout.LayoutParams.WRAP_CONTENT
            height = mainBoxHeight
        }
        pack.col.gravity              =
Gravity.CENTER_HORIZONTAL             or
Gravity.CENTER_VERTICAL
        pack.col.translationY = 0f
        pack.col.requestLayout()
        if (pack.scale < 1f) {
            pack.row.clipChildren     =
false
            pack.col.clipChildren     =
false
        }
    }
```

```
    private fun fitPriceToCloud() {
        if
(!::priceCloud.isInitialized        ||
priceCloud.width      ==      0      ||
priceCloud.height == 0) return
        val          digits          =
priceMainView.text?.length ?: 1
        val t = tuningForDigits(digits)
        val padL = (priceCloud.width  *
t.leftPct ).toInt()
        val padR = (priceCloud.width  *
t.rightPct).toInt()
        val padT0 = (priceCloud.height
* t.topPct   ).toInt()
        val padB0 = (priceCloud.height
* t.bottomPct).toInt()
        val totalTB = padT0 + padB0
        val padT = totalTB / 2
        val padB = totalTB - padT
        priceColumn.setPadding(padL,
padT, padR, padB)
        val availW = (priceCloud.width
- padL - padR).coerceAtLeast(1)
        val availH = (priceCloud.height
- padT - padB).coerceAtLeast(1)
        val   safeW   =   (availW   *
t.safeWFactor).toInt()
        val           sd           =
resources.displayMetrics.scaledDensity
        val  hiByGeomSp  =  ((availH  *
t.hiRowFactor)                        /
sd).coerceAtMost(280f)
        var lo = 8f
        var   hi   =   min(hiByGeomSp,
TypeScale.PRICE_MAJOR)
        val         centsText         =
priceMinorView.text?.toString()     ?:
".00"
        val      currText            =
priceCurrencyView.text?.toString()  ?:
"грн"
        repeat(24) {
            val mid = (lo + hi) / 2f
            val       widthPx       =
computeRowWidthPx(mid,      centsText,
currText, t)
            val  fitsW  =  widthPx  <=
safeW
            applySizesForMeasure(mid,
t)
            priceRow.measure(

View.MeasureSpec.makeMeasureSpec(safeW
, View.MeasureSpec.AT_MOST),

View.MeasureSpec.makeMeasureSpec(avail
H, View.MeasureSpec.AT_MOST)
            )
            val         fitsH         =
priceRow.measuredHeight <= availH
            if (fitsW && fitsH) lo =
mid else hi = mid
        }
        applySizesForMeasure(lo,    t,
finalApply = true)
    }
    private                          fun
applySizesForMeasure(majorSp:   Float,
t: Tuning, finalApply: Boolean = false)
{
        applyPack(newPack, majorSp, t)
        applyPack(oldPack, majorSp, t)
    }
    private                          fun
applyResponsiveLayout(root:
ConstraintLayout) {
        val            dm            =
resources.displayMetrics
        val   wDp   =   root.width   /
dm.density
        val   hDp   =   root.height   /
dm.density
        val s = min(wDp / 1280f, hDp /
720f).coerceIn(0.65f, 1.05f)
        fun  pxs(dp:  Int)  =  (dp  *
dm.density * s).roundToInt()
        (split.layoutParams          as
ConstraintLayout.LayoutParams).apply {
            guidePercent = when {
                s < 0.72f -> 0.44f
                s < 0.86f -> 0.45f
                else      -> 0.46f
            }
            split.layoutParams = this
        }
        val edgeR = pxs(24)
        (logo.layoutParams           as
ConstraintLayout.LayoutParams).apply {
            marginEnd = edgeR
            topMargin = pxs(16)
            logo.layoutParams = this
        }
        logo.maxWidth = pxs(280)
        logo.maxHeight = pxs(120)
        val  safeTopPx  =  max(pxs(24),
logo.height + pxs(8))
        (card.layoutParams           as
ConstraintLayout.LayoutParams).apply {
            marginStart  = pxs(24)
            marginEnd    = edgeR
            topMargin    = safeTopPx
            bottomMargin = edgeR
            verticalBias = 1f
            card.layoutParams = this
        }
        (imageFrame.layoutParams     as
FrameLayout.LayoutParams).apply {
            val     m     =     pxs(10);
setMargins(m, m, m, m)
            imageFrame.layoutParams  =
this
        }
        (priceCloud.layoutParams     as
ConstraintLayout.LayoutParams).apply {
            marginStart = pxs(24)
            marginEnd   = pxs(28)
```

```
matchConstraintPercentWidth = when {
                s < 0.72f -> 0.38f  //
було 0.36
                s < 0.86f -> 0.40f  //
було 0.38
                else      -> 0.42f  //
було 0.40
            }
            topMargin = when {
                s < 0.75f -> pxs(8)
                s < 0.90f -> pxs(14)
                else      -> pxs(16)
            }
            priceCloud.layoutParams  =
this
        }
        (oldPriceWrap.layoutParams  as
ConstraintLayout.LayoutParams).apply {
            bottomMargin = pxs(2)
            oldPriceWrap.layoutParams
= this
        }
        val titleMax = (TypeScale.TITLE
* s).coerceIn(18f, 54f).toInt()
        val    titleMin    =    max(14,
(titleMax * 0.65f).toInt())

TextViewCompat.setAutoSizeTextTypeUnif
ormWithConfiguration(
            titleView,       titleMin,
titleMax,                          1,
TypedValue.COMPLEX_UNIT_SP
        )
        val        articleMax        =
(TypeScale.ARTICLE  *  s).coerceIn(22f,
44f).toInt()
        val   articleMin   =   max(16,
(articleMax * 0.75f).toInt())

TextViewCompat.setAutoSizeTextTypeUnif
ormWithConfiguration(
            articleView,   articleMin,
articleMax,                        1,
TypedValue.COMPLEX_UNIT_SP
        )
    }
    override  fun  onViewCreated(view:
View, savedInstanceState: Bundle?) {
        val            sku           =
arguments?.getString("sku") ?: ""
        val          barcode         =
arguments?.getString("barcode") ?: ""
        val           title          =
arguments?.getString("title")       ?:
"Товар"
        val           price          =
arguments?.getDouble("price") ?: 0.0
        val        promoPrice        =
arguments?.getDouble("promo_price")?.t
akeIf { it > 0 }
        val           image          =
arguments?.getString("image") ?: ""
        titleView.text = title
        articleView.text = when {
            sku.isNotBlank()        ->
"Артикул: $sku"
            barcode.isNotBlank()    ->
"Артикул: $barcode"
            else                    -> ""
        }
        if   (title   ==   "Товар   не
знайдено") {

productImageView.setImageResource(R.dr
awable.placeholder)
            priceCloud.visibility    =
View.GONE
            articleView.visibility   =
View.GONE
            return
        }
        if (image.isBlank()) {

productImageView.setImageResource(R.dr
awable.placeholder)
        } else {

Glide.with(requireContext())
                .load(image)

.placeholder(R.drawable.placeholder)

.error(R.drawable.placeholder)

.fallback(R.drawable.placeholder)

.into(productImageView)
        }
        fun   splitPrice(v:   Double):
Pair<String, String> {
            val vv = if (v.isFinite())
v else 0.0
            val   rounded   =   (vv   *
100.0).roundToInt() / 100.0
            val        whole        =
floor(rounded).toInt()
            val  cents  =  ((rounded  -
whole) * 100).roundToInt()
            return whole.toString() to
String.format(".%02d", cents)
        }
        if (promoPrice != null) {
            val    (wNew,    cNew)    =
splitPrice(promoPrice)
            priceMainView.text = wNew
            priceMinorView.text = cNew
            val    (wOld,    cOld)    =
splitPrice(price)
            oldMainView.text = wOld
            oldMinorView.text = cOld
            oldPriceWrap.visibility  =
View.VISIBLE
        } else {
            val      (w,      c)      =
splitPrice(price)
```

```
            priceMainView.text = w
            priceMinorView.text = c
            oldPriceWrap.visibility  =
View.GONE
        }
        priceCloud.post              {
fitPriceToCloud() }
    }
    companion object {
        fun newInstance(
            barcode: String,
            sku: String,
            title: String,
            price: Double,
            promoPrice: Double?,
            image: String
        ) = ProductFragment().apply {
            arguments = bundleOf(
                "barcode" to barcode,
                "sku" to sku,
                "title" to title,
                "price" to price,
                "promo_price"       to
promoPrice,
                "image" to image
            )
        }
    }
}
```

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Владислав МЕЛЬНИК

Співавтор:

Назва: Кіберфізична система «Сканер товарів» на базі Raspberry Pi

Експерт: Катерина БЕРЕЗЬКА

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:8.23%

Коефіцієнт подібності 2:1.05%

Мікропробіли: 3

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2026-05-27 10:47:06.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2026-05-28 Доцент Андрій Нічепорук

Дата експерт

Wed May 27 09:59:02 EEST 2026, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (http://ap.km.ua) v-15.701

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 12%**

	Документ		Сумарний збіг по Базі Даних	
ID: 272407 Назва: БКР Кіберфізична система «Сканер товарів» на базі Raspberry Pi Додано в БД: 2026-05-27 Автора: Владислав МЕЛЬНИК Керівники: Катерина БЕРЕЗЬКА Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	75395	669	3178 (4%)	48 (7%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Мельник Владислав Віталійович

Тема: Кіберфізична система «Сканер товарів» на базі Raspberry Pi

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3____ Кількість сторінок записки ___61___

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є розробка та програмно-апаратна реалізація кіберфізичної системи «Сканер товарів» на базі одноплатного комп'ютера Raspberry Pi для автоматизації процесу зчитування, обробки та відображення інформації про товари.

2. Висновок про відповідність роботи дипломному завданню: Кваліфікаційна робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі кваліфікаційної роботи проведено дослідження предметної області, виконано аналіз сучасних кіберфізичних систем у сфері автоматизації торгівлі, розглянуто принципи роботи сканерів товарів, технології штрихкодування та RFID-ідентифікації, а також проаналізовано програмно-апаратні засоби для побудови системи на базі Raspberry Pi.

У другому розділі виконано проєктування кіберфізичної системи «Сканер товарів», обґрунтовано вибір апаратних компонентів та програмного забезпечення, розроблено структурну, функціональну та інформаційну схеми системи, описано алгоритм функціонування системи, організацію обробки даних та взаємодію між програмними й апаратними компонентами.

У третьому розділі виконано програмно-апаратну реалізацію кіберфізичної системи «Сканер товарів» на базі Raspberry Pi. Реалізовано підключення сканера штрихкодів, програмну обробку інформації про товари, відображення результатів роботи системи, а також проведено тестування та аналіз працездатності розробленої системи в реальних умовах експлуатації.

4. Позитивні сторони роботи: До позитивних сторін роботи слід віднести актуальність тематики, практичну спрямованість розробки, використання сучасної платформи Raspberry Pi, а також успішну інтеграцію апаратних та програмних компонентів у єдину кіберфізичну систему.

5. Негативні сторони роботи: -

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка та графічна частина роботи оформлені відповідно до чинних стандартів і вимог нормативної документації. Матеріал викладено послідовно, логічно та технічно грамотно.

7. Відгук про роботу в цілому: Робота Кваліфікаційна робота виконана на високому технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: відмінно (A / 93)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

Федосенко Денис Петрович, зав.каф. ІПЗ, ХНУ

"03" червня 2026 р. _____ (підпис)

Зав. кафедри КІІС
д-р. філософії Ользі ПАВЛОВІЙ

Владислав МЕЛЬНИК

ПІБ здобувача вищої освіти

ФІТ, 3 курсу, групи КІ2с-23-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів академічної відповідальності, ознайомлений (а). Про використання спеціалізованих програмних засобів (СПЗ) StrikePlagiarism та Anti-Plagiarism для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а). Надаю університету право на передачу моєї роботи для обробки та збереження в базах даних СПЗ і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються СПЗ.

Також надаю свою згоду на обробку й збереження університетом моєї роботи в Інституційному репозитарії Хмельницького національного університету.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

1 травня 2026 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи ІКіберфізична система «Сканер товарів» на базі Raspberry Pi
Автор Владислав МЕЛЬНИК
Освітня програма Комп'ютерна інженерія та програмування
Рівень вищої освіти перший (бакалаврський)
Спеціальність 123 Комп'ютерна інженерія
Науковий керівник: к.т.н., доцент Катерина БЕРЕЗЬКА

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	відповідає
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
2) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;
3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.
4) значна частина знайденого плагіату відноситься до списку використаних джерел

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 8,23%; та системою Anti-Plagiarism складає 1,0%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

01.06.2026

Завідувач кафедри — Підпис — Ольга ПАВЛОВА (Ім'я, ПРІЗВИЩЕ)

Гарант освітньої програми — Підпис — Андрій НІЧЕПОРУК (Ім'я, ПРІЗВИЩЕ)

Керівник кваліфікаційної роботи — Підпис — Катерина БЕРЕЗЬКА (Ім'я, ПРІЗВИЩЕ)