Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка Java-додатку для розпізнавання EAN/UPC, Code 128 та QR-кодів з подальшим збереженням розшифрованих даних

Виконав(ла): студент(ка) 4 курсу, групи СПс-43
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____ Боднар Б. І.
(підпис) (прізвище та ініціали)

Керівник _____ Бревус В. М.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М. Р.
(підпис) (прізвище та ініціали)

Рецензент _____ Готович В. А.
(підпис) (прізвище та ініціали)

Тернопіль 2025

АНОТАЦІЯ

Кваліфікаційна робота на здобуття освітнього ступеню «бакалавр» за спеціальністю 121 Інженерія програмного забезпечення. Тернопільський національний технічний університет ім. Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група, 2025 рік. Пояснювальна записка до кваліфікаційної роботи на здобуття освітнього ступеню «бакалавр» містить: 62 с., 25 рис.

Тема: Розробка Java-додатку для розпізнавання EAN/UPC, Code 128 та QR-кодів з подальшим збереженням розшифрованих даних.

Об'єктом дослідження є процес розробки системи розпізнавання та декодування штрих-кодів і QR-кодів з використанням сучасних технологій.

Метод дослідження – в основу роботи закладено дослідження процесу розпізнавання та декодування штрих-кодів і QR-кодів, а також обґрунтування доцільності використання обраних технологій для реалізації даних задач.

Завданням даного проєкту є розробка ефективного та надійного програмного рішення для автоматизованого розпізнавання кодів з зображень та надання відповідної інформації про продукти.

Результат – система, що повністю готова до використання.

Ключові слова: Java, EAN/UPC, Code 128, QR-код.

ABSTRACT

Qualification work for obtaining the Bachelor's degree in Software Engineering (specialty 121). Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, Group , 2025. The explanatory note to the qualification work for obtaining the Bachelor's degree contains: pages, figures.

Topic: Development of a Java application for recognizing EAN/UPC, Code 128, and QR codes with subsequent storage of the decoded data.

The object of the research is the process of developing a system for recognizing and decoding barcodes and QR codes using modern technologies.

Research method: The basis of the work is the study of the process of recognizing and decoding barcodes and QR codes, as well as the justification of the feasibility of using the selected technologies to implement these tasks.

The task of this project is to develop an efficient and reliable software solution for the automated recognition of codes from images and the provision of relevant product information.

Result: A system that is fully ready for use.

Keywords: Java, EAN/UPC, Code 128, QR code.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

EAN – європейський артикуловий номер, міжнародний стандарт штрихового коду.

Code 128 – високощільний лінійний штрих-код, який кодує всі 128 символів ASCII.

QR-код – матричний двовимірний штрих-код, який використовується для кодування текстової інформації.

API (Application Programming Interface) – інтерфейс програмування додатків, який дозволяє взаємодіяти між різними програмами.

REST API – тип API, що дотримується архітектурного стилю REST (Representational State Transfer), який використовує HTTP протоколи.

Java – об'єктно-орієнтована мова програмування, що широко використовується для створення веб-додатків.

Spring Boot – фреймворк для створення автономних, продуктивних додатків на Java, що спрощує налаштування та розгортання.

БД – база даних.

DTO (Data Transfer Object) – об'єкт, що використовується для передачі даних між різними частинами додатку.

GUI (Graphical User Interface) – графічний інтерфейс користувача.

JUnit – фреймворк для написання юніт-тестів на Java.

Mockito – бібліотека для створення мок-об’єктів (імітацій об’єктів) у тестах.

ER-діаграма (Entity-Relationship Diagram) – графічне представлення сутностей бази даних і зв’язків між ними.

JSON (JavaScript Object Notation) – формат представлення структурованих даних у вигляді тексту.

ВСТУП

Розробка систем автоматизованого розпізнавання штрих-кодів та QR-кодів є актуальною темою в сучасному світі інформаційних технологій. З розвитком електронної комерції, логістики та управління запасами, попит на ефективні та надійні рішення для швидкого та точного розпізнавання кодів постійно зростає. Навички у розробці таких систем є надзвичайно корисними для кар'єрного зростання у галузі програмування та інформаційних технологій.

Проєктування та реалізація системи розпізнавання та декодування штрих-кодів і QR-кодів є складним і цікавим завданням. Вона вимагає глибокого розуміння роботи з зображеннями, алгоритмами обробки та розпізнавання кодів, а також вміння використовувати сучасні бібліотеки та фреймворки для створення надійних і ефективних програмних рішень. Сучасні підходи до обробки зображень дозволяють досягти високої точності розпізнавання та швидкості обробки, що є критично важливими у багатьох сферах застосування.

Spring Boot є одним із найпопулярніших фреймворків для розробки веб-додатків на Java, що дозволяє створювати масштабовані та надійні рішення. Використання цього фреймворку у поєднанні з бібліотеками для обробки зображень дозволяє реалізувати потужні інструменти для розпізнавання штрих-кодів та QR-кодів.

Мета даної дипломної роботи полягала у дослідженні та розробці системи розпізнавання та де кодування штрих-кодів і QR-кодів з використанням сучасних технологій.

Дослідження цієї теми дозволить краще зрозуміти принципи роботи з зображеннями та кодами, а також надасть можливість опанувати нові технології та підходи в програмуванні, що буде корисним у подальшій професійній діяльності.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУЛЮВАННЯ ЗАВДАННЯ

1.1 Аналіз існуючих аналогів

Аналіз існуючих альтернатив для дипломної роботи на тему "Розробка Java-додатку для розпізнавання EAN/UPC, Code 128 та QR-кодів з подальшим збереженням розшифрованих даних". В рамках дослідження ринку було проаналізовано кілька можливих варіантів аналогів, що розглядаються у деталях нижче.

Zebra Crossing (ZXing) API є одним із найпопулярніших і широко використовуваних інструментів для розшифрування штрих-кодів. Цей API пропонує можливість розпізнавання різних типів штрих-коди, включаючи QR-коди, Data Matrix та інші.

Втім, незважаючи на свою популярність, ZXing API має кілька значних недоліків, основним з яких є відсутність можливості автоматично шукати дані розшифрованого продукту у відкритих базах даних. Це означає, що користувач повинен самостійно здійснювати пошук інформації про продукт після розшифрування, що значно ускладнює процес та знижує зручність використання.

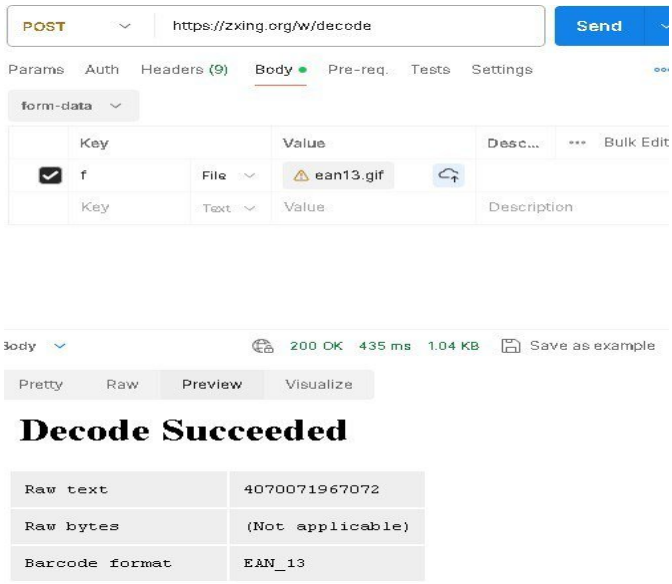

Рисунок 1.1.1 – Результат виконання запиту до ZXing API

OCR.Space є інструментом для оптичного розпізнавання символів, який також може використовуватися для розшифрування EAN/UPC, Code 128 та QR-кодів.

Цей сервіс надає високоякісне розпізнавання зображень, однак має свої недоліки:

- Необхідність авторизації та обмеження безкоштовної версії: Для використання OCR.Space необхідно пройти процес авторизації, що може бути незручним для користувачів, які шукають швидке рішення. Крім того, безкоштовна версія сервісу має обмеження на кількість запитів, що обмежує його використання для масових завдань.
- Відсутність пошуку продукту за кодом: Подібно до ZXing, OCR.Space не надає можливості автоматичного пошуку інформації про продукт за розшифрованим штрих-кодом, що знову ж таки додає додаткових кроків для користувача.

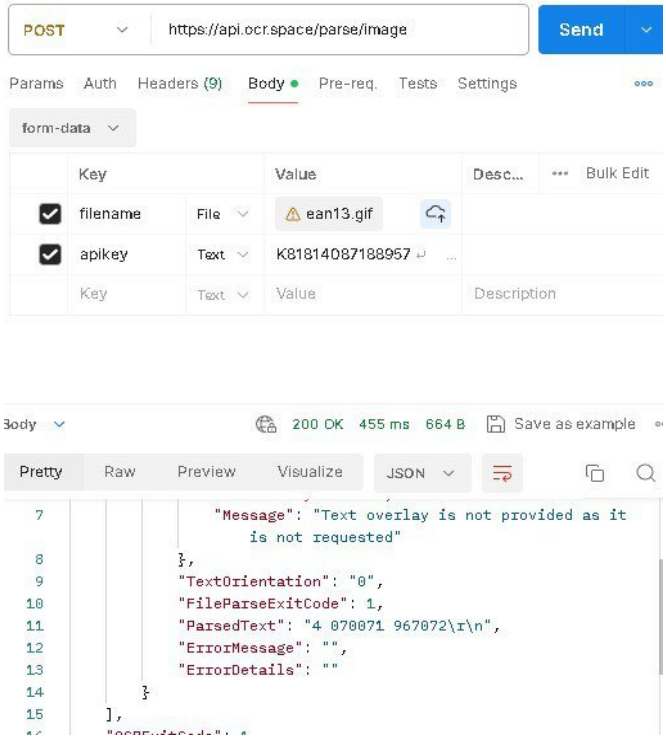

Рисунок 1.1.2 – Результат виконання запиту до OCR.Space

Cloudmersive Barcode API є потужним інструментом для розшифрування штрих-кодів з великим набором функціональних можливостей. Втім, цей API також має кілька недоліків, які варто враховувати:

- Складність у використанні: Через великий функціонал Cloudmersive Barcode API може бути важким для розуміння та використання початковими користувачами. Високий рівень складності інтерфейсу та велика кількість налаштувань можуть відлякувати новачків.
- Надлишковий функціонал: Для багатьох користувачів велика кількість функцій, які пропонує Cloudmersive, може бути зайвою. Це може призводити до того, що користувачі будуть змушені витрачати час на вивчення та налаштування непотрібних їм опцій.

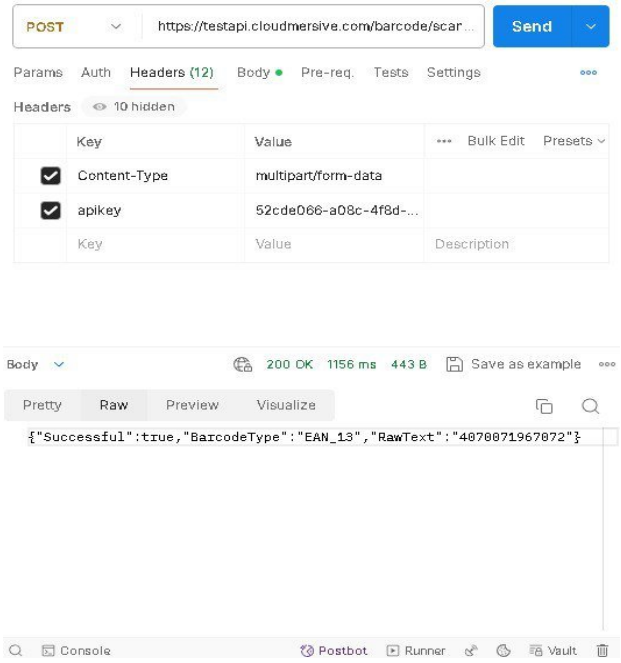

Рисунок 1.1.3 – Результат виконання запиту до Cloudmersive Barcode API

Розглянуті аналоги мають як сильні, так і слабкі сторони. Головними недоліками більшості існуючих рішень є відсутність можливості автоматичного пошуку даних розшифрованого продукту та складність використання для непідготовлених користувачів. Це створює можливість для розробки нового

додатку, який буде вирішувати ці проблеми та надавати користувачам більш зручний і ефективний інструмент для розшифрування штрих-кодів.

1.2 Обгрунтування доцільності створення додатку

Для обґрунтування доцільності створення додатку для розшифрування EAN/UPC, Code 128 та QR-кодів потрібно врахувати кілька ключових аспектів, які охоплюють технічні, ринкові та користувацькі фактори.

Перш за все, зростаючий попит на автоматизацію процесів робить розробку такого додатку вкрай актуальною. У сучасному світі швидкість і ефективність обробки інформації відіграють вирішальну роль у багатьох сферах діяльності.

Автоматизація процесів стала необхідністю для бізнесу, який прагне залишатися конкурентоспроможним. Штрих-коди використовуються повсюдно – від роздрібної торгівлі до логістики та охорони здоров’я. Прискорення процесу розшифрування і автоматичний пошук даних про продукти можуть значно підвищити ефективність роботи компаній та задовольнити потреби кінцевих користувачів [1].

По-друге, потреба в ефективних рішеннях для бізнесу та особистого використання є ще одним вагомим аргументом. Додаток для розшифрування EAN/UPC, Code 128 та QR-кодів, який може автоматично знаходити інформацію про продукти в відкритих базах даних, може стати незамінним інструментом для різних користувачів.

Роздрібні магазини зможуть швидко та точно ідентифікувати продукти та оновлювати інформацію про них. Приватні користувачі зможуть легко отримувати інформацію про продукти, що сприятиме їхньому інформованому вибору. Недоліки існуючих аналогів також підкреслюють доцільність розробки нового додатку.

Як було розглянуто в попередньому розділі, існуючі рішення для розшифрування штрих-кодів мають кілька суттєвих недоліків. Відсутність автоматичного пошуку даних, необхідність авторизації, складність у використанні та складність впровадження є основними перешкодами, з якими стикаються користувачі.

Новий додаток повинен врахувати ці недоліки та запропонувати вдосконалені можливості. Автоматичний пошук даних забезпечить інтеграцію з відкритими базами даних для автоматичного пошуку інформації про продукти після розшифрування.

Зручність використання, інтуїтивно зрозумілий інтерфейс і простота налаштування зроблять додаток доступним для широкого кола користувачів, включаючи тих, хто не має глибоких технічних знань.

Економічна ефективність, надання основних функцій зробить додаток привабливим для індивідуальних користувачів. Отже, створення додатку для розшифрування EAN/UPC, Code 128 та QR-кодів, який поєднує в собі автоматичний пошук даних, зручність використання та доступність, стане цінним інструментом для багатьох користувачів.

Враховуючи зростаючий попит на автоматизацію процесів і недоліки існуючих рішень, створення такого додатку є доцільним і обґрунтованим кроком. Впровадження нових функціональних можливостей і інновацій забезпечить його конкурентоспроможність і сприятиме його широкому використанню у різних сферах діяльності.

1.3 Постановка завдання для розробки додатку

На основі аналізу існуючих аналогів та потреб користувачів можна визначити основні завдання для розробки додатку. Головними завданнями створення додатку для розшифрування EAN/UPC, Code 128 та QR-кодів є:

Передусім необхідно провести детальний аналіз існуючих рішень для розшифрування штрих-кодів. Це включає вивчення різних додатків та сервісів, визначення їхніх сильних та слабких сторін для виявлення можливостей для вдосконалення та інновацій.

Також необхідно провести дослідження різних відкритих баз даних для пошуку інформації про продукти за штрих-кодами. Оцінка актуальності, доступності, обсягу інформації та зручності інтеграції цих баз даних є ключовим завданням для забезпечення належної функціональності додатку.

Розробка інтуїтивно зрозумілого та зручного графічного інтерфейсу є наступним важливим завданням. Інтерфейс повинен забезпечити швидке розшифрування EAN/UPC, Code 128 та QR-кодів та автоматичний пошук інформації про продукти, роблячи процес максимально комфортним для користувачів. Інтеграція інструментів для розшифрування EAN/UPC, Code 128 та QR-кодів є ще одним ключовим завданням.

Комплексне тестування розробленого додатку є необхідним для виявлення та виправлення помилок. Перевірка функціональності, зручності використання та продуктивності додатку включає як автоматизовані тести, так і тестування з участю реальних користувачів.

Забезпечення безпеки даних користувачів є критично важливим завданням. Впровадження сучасних методів шифрування та захисту даних, розробка політики конфіденційності та забезпечення відповідності додатку вимогам законодавства щодо захисту даних є обов'язковими кроками.

Виконання цих завдань дозволить створити ефективний, зручний та безпечний додаток для розшифрування EAN/UPC, Code 128 та QR-кодів, який буде відповідати потребам користувачів та вимогам ринку.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ ТА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Формулювання вимог до програмного забезпечення

Прикладний програмний інтерфейс для сканування штрихкодів є програмним рішенням, що забезпечує передачу та збереження даних про скановані коди через мережеві HTTP-запити на визначених кінцевих точках API. Розроблена система орієнтована на обробку таких типів штрихкодів, як EAN/UPC, Code 128 та QR-коди, з можливістю їхнього розшифрування, пошуку відповідної інформації про продукти через сторонні сервіси, наприклад, OpenFoodFacts та подальшого збереження даних у базу даних.

Основні функції програмного рішення включають:

- Встановлення довготривалого з'єднання зі сховищем даних.
- Розшифрування та збереження коду, включно з його типом (URL, JSON або текст).
- Передача отриманих даних у стандартизованих форматах
- Здійснення запитів до зовнішніх баз даних для отримання додаткової інформації.
- Інтеграція даних у власне сховище для подальшого доступу та аналізу.

Додаток не містить графічного інтерфейсу користувача, оскільки розроблений виключно як REST API. Проте передбачена можливість взаємодії з ним через HTTP-запити сторонніх клієнтів, наприклад, frontend-додатку або Postman.

Дані, що зберігаються, представлені в уніфікованому форматі, що дозволяє легко їх обробляти та масштабувати у випадку зростання навантаження. Результати моделювання логіки процесів відображено на рисунку 2.1.1.1, де продемонстровано послідовність взаємодій між основними модулями програми — користувачем, контролером, модулями розшифрування, збереження та пошуку інформації.

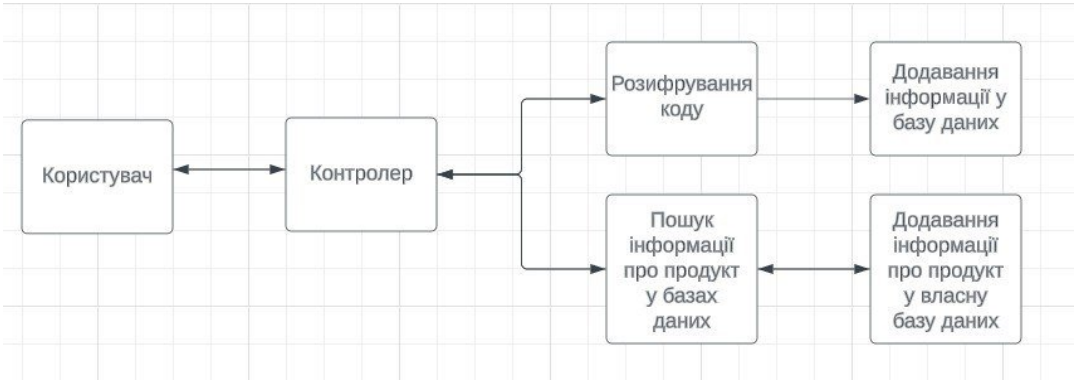

Рисунок 2.1.1 – Основні деталі процесу роботи програми

2.1.1 Основні вимоги до програмного забезпечення

Функціональні вимоги:

- Забезпечення довготривалого та стабільного з'єднання з мережею та сховищем даних.
- Надання інтуїтивно зрозумілих та задокументованих кінцевих точок API.
- Обробка HTTP-запитів на отримання та збереження штрихкодів.
- Автоматичне визначення типу отриманого коду (URL, JSON, текст).
- Запит до зовнішніх API з метою пошуку додаткової інформації.
- Інтеграція нових даних у локальну базу.

Нефункціональні вимоги до системи включають наступні пункти:

1. Надійність

- Підтримка: Програмне рішення не передбачає додаткової підтримки.
- Зручність: Інтуїтивно зрозумілі точки доступу до запитів.

2. Придатність до експлуатації:

- Оновлення: Підтримка оновлень, виправлення помилок та відновлення до попередніх версій.

- Масштабованість: Забезпечення масштабування та розширення кількості з'єднань.

3. Особливості:

- Модульність: Можливість модифікації окремих модулів без зміни всього рішення.
- Доступ до даних: Доступ до сховища даних про скановані коди.
- Масштабованість: Передбачення можливостей розширення та модифікації.

4. Обмеження:

- Платформа запуску: Windows-сервер або Linux з підтримкою HTTP.
- Система не містить механізмів захисту персональних даних, оскільки не працює з конфіденційною інформацією.
- Вся логіка реалізована через API, без GUI-компонентів.

Програмне рішення передає загальнодоступні персональні дані та не потребує додаткового захисту. Програмне рішення реалізує прикладний програмний інтерфейс і не включає графічного користувацького інтерфейсу.

Дані про скановані коди завжди актуальні та представлені в єдиній формі. Сховище даних має бути легко доступним, зручним та придатним для використання в API.

2.2 Проектування структури програми

2.2.1 Компонентна структура програми

Для ефективної реалізації функціональності розпізнавання штрихкодів, збереження та пошуку інформації про продукти, програму поділено на логічні модулі, кожен з яких відповідає за конкретні завдання. Цей підхід дозволяє покращити читабельність, супровідність та масштабованість системи.

Обробка запитів користувачів включає прийом зображень, запитів на пошук інформації та збереження даних. Розпізнавання штрихкодів передбачає підтримку таких типів, як EAN, UPC, Code 128 та QR-коди. Інтеграція з базами даних забезпечує можливість зчитування та збереження інформації як у внутрішній базі даних додатку, так і за допомогою зовнішніх API. Збереження результатів полягає у фіксації відомостей про продукти, а також про всі здійснені сканування для подальшого аналізу чи обробки.

Компоненти системи та їх функції(рис 2.2.1.1.):

- Контролер (Controller Module): обробляє запити користувача, викликає відповідні сервіси, повертає відповіді.
- Сервіс для взаємодії зі штрихкодами: перевіряє формат зображення, розшифровує код, шукає інформацію про продукт у зовнішніх API.
- Модуль бази даних: додає дані про сканування та шукає товари в локальній базі.
- Модуль обробки помилок: ловить виключення, генерує користувацькі повідомлення.

Ці модулі взаємодіють послідовно. Наприклад, контролер передає зображення сервісу, той розпізнає штрихкод, здійснює запит у стороннє API, а потім результат зберігається у базі даних. Якщо виникає помилка — її перехоплює відповідний обробник.

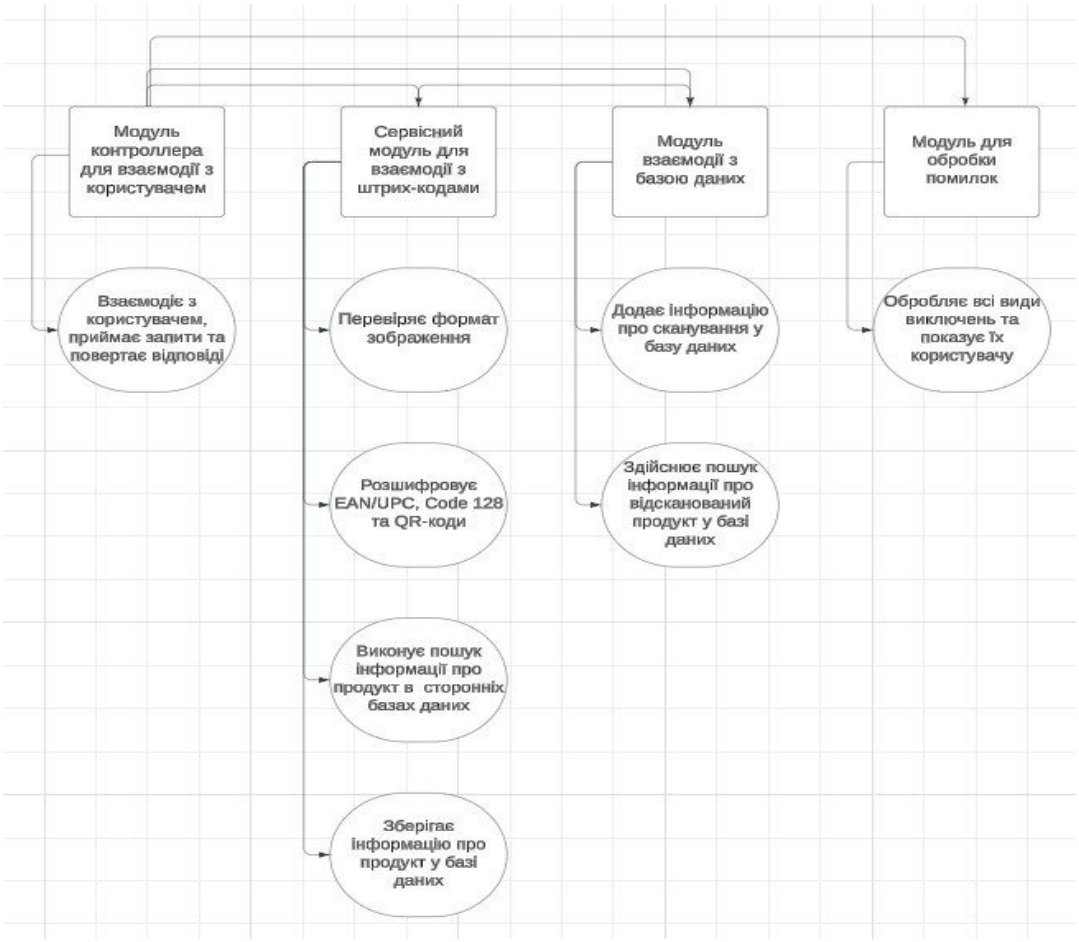

Рисунок 2.2.1.1 – Компоненти додатку

2.2.2 Внутрішня логіка контролера

Контролер відіграє ключову роль у структурі програми, виконуючи функцію координаційного центру між запитом користувача і внутрішньою логікою застосунку. Для спрощення підтримки та розширення функціоналу контролер структуровано на кілька базових дій.

Компоненти контролера (Рис 2.2.2.1):

- Розкодувати зображення: отримує зображення, передає його на розшифрування та повертає результат.

- Виконати пошук продукту: приймає код, шукає інформацію про продукт.
- Розкодувати та знайти продукт: об’єднує два попередні кроки — від завантаження зображення до повернення детальної інформації про продукт.

Ця структура дозволяє зручно тестувати кожну дію окремо, забезпечуючи гнучкість і масштабованість при додаванні нових форматів кодів чи логіки обробки.

Рисунок 2.2.2.1 – Компоненти класу CodeController

2.3 Опис архітектури програми

Після того як сформовано загальну компонентну структуру програми, важливо детально описати її архітектуру на рівні класів. Архітектура повинна дотримуватися принципу розділення відповідальностей, згідно з яким кожен клас виконує чітко окреслену функцію.Цей етап передбачає визначення ролей кожного класу, способів взаємодії між ними, принципів спадкування та відповідальності за реалізацію окремих частин бізнес-логіки [2].

У запропонованій архітектурі ядро програми становить головний контролерний клас CodeController, який виконує роль фасаду, з яким безпосередньо взаємодіє клієнт через REST API. Він отримує HTTP-запити, валідує дані, викликає необхідні сервісні методи, обробляє винятки та формує структуровані відповіді. Однак сам контролер не виконує складних обчислень — його основна задача полягає у делегуванні завдань відповідним службам.

Навколо CodeController реалізовано набір допоміжних класів, які відповідають за реалізацію різних аспектів логіки програми. Архітектура дотримується принципу розділення відповідальностей, згідно з яким кожен клас виконує чітко окреслену функцію. Такий розподіл забезпечує слабке зв’язування компонентів, що позитивно впливає на тестованість і масштабованість проєкту [3].

Усі класи умовно можна поділити на декілька груп:

1. Сервісні класи — обробляють основну логіку:

- CodeDecryptor відповідає за аналіз зображення та спробу розпізнавання штрихкоду. Він працює з бібліотеками зчитування кодів, наприклад, ZXing.
- BarcodeInfoReturner виконує пошук інформації про товар. Спочатку відбувається запит до локальної бази, якщо ж результату не знайдено — ініціюється запит до стороннього API.
- CodeRecordService фіксує інформацію про успішні сканування в базі даних, разом із часовими мітками та додатковою інформацією.

2. Класи DTO — забезпечують передачу даних між шарами:

- ProductCodeDto, ScanInfoDto — містять лише необхідні поля для відповіді або запиту. Наприклад, DTO для відповіді про товар включає назву, бренд, опис і штрихкод, але не містить метаданих з бази.

3. Класи валідації та обробки помилок:

- ImageFormatValidator — перевіряє отримані зображення на відповідність форматам (розширення, розмір).
- MainExceptionHandler — є глобальним обробником помилок (наприклад, некоректні запити, відсутність результатів або помилки при взаємодії зі сторонніми API).
- Винятки, як-от EmptyImageException, InvalidDecryptionFormatException, InvalidImageFormatException обробляються централізовано через механізм Spring @ControllerAdvice.

4. Інтерфейси та абстрактні класи:

- BarcodeProcessor — інтерфейс для сервісів розпізнавання коду, який може мати різні реалізації залежно від типу коду (наприклад, QR, EAN, Code 128).
- AbstractBarcodeService — базовий абстрактний клас із загальною логікою для роботи з кодами, який може бути успадкований для конкретних форматів.

Архітектура передбачає також певний рівень успадкування. Наприклад, класи для обробки результатів сканування наслідують загальну структуру, в якій визначено основні поля (timestamp, status, message) та методи (toDto(), logResult()). Це дозволяє уникнути дублювання коду та підтримувати єдиний стиль обробки.

Взаємозв'язки між класами чітко визначені: контролер знає лише про сервіси, вони взаємодіють із базою даних або зовнішніми API, а DTO використовуються для передачі даних між шарами. Такий розподіл забезпечує слабке зв’язування компонентів, що позитивно впливає на тестованість і масштабованість проєкту.

У центрі архітектурної діаграми, зображеної рисунку 2.3.1, знаходиться CodeController, який безпосередньо пов’язаний із сервісними класами. Ці сервіси, у свою чергу, посилаються на DTO, обробники винятків та допоміжні утилітарні класи. Важливою деталлю є також можливість розширення: у майбутньому можна

легко додати нові типи кодів або нові джерела пошуку (наприклад, інший API), просто реалізувавши відповідні сервіси, не змінюючи існуючу логіку контролера.

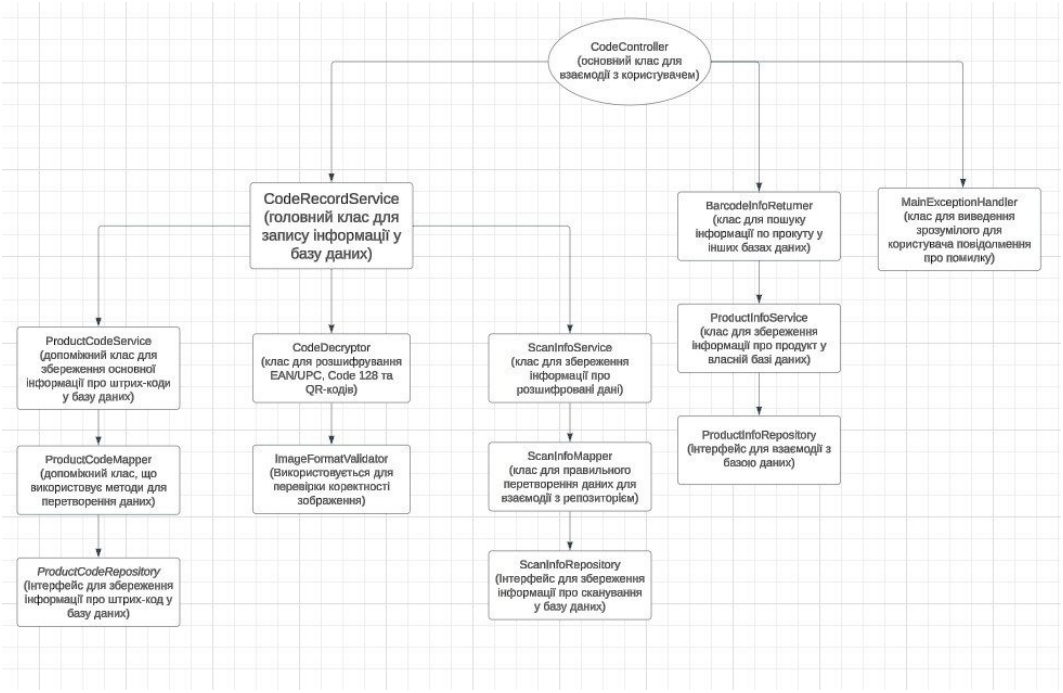

Рисунок 2.3.1 – Архітектура класів додатку

2.4 Опис архітектури бази даних

Для ефективного зберігання результатів сканування, розпізнаних штрихкодів та пов’язаної з ними інформації була спроєктована логічна модель бази даних, яка охоплює основні аспекти функціонування програми. Архітектура бази даних побудована таким чином, щоб забезпечити зручне, структуроване та логічно пов’язане зберігання всіх необхідних даних, пов’язаних із процесом розпізнавання та подальшою обробкою штрихкодів.

До складу бази входять три ключові таблиці: scan_info, product_codes та product_info. Кожна з них виконує окрему роль у загальній структурі даних та взаємодіє з іншими через логічні зв’язки.

Таблиця scan_info є однією з центральних у всій системі. Вона зберігає всі дані, пов’язані з процесом сканування кодів. Кожен запис у цій таблиці представляє одну спробу сканування — незалежно від того, була вона успішною чи ні. Основними полями таблиці є ідентифікатор сканування, дата й час, коли воно було здійснено, посилання на код, який було намагались розпізнати, а також прапорець, що визначає успішність цієї спроби. Додатково вказується тип зчитаного значення. Така структура дозволяє вести хронологічний облік усіх спроб сканування, а також отримувати статистику — наприклад, кількість успішних та неуспішних зчитувань за певний період, або частоту використання певного типу кодів.

Таблиця product_codes слугує для зберігання даних про унікальні штрихкоди. Тут фіксуються усі коди, які хоча б раз були оброблені системою. До кожного запису зберігається тип коду — наприклад, QR-код, EAN-13 або Code 128 — а також його значення. Крім того, зберігається ім’я файлу, у якому цей код було виявлено або який став джерелом для сканування. Це дозволяє здійснювати зворотне відстеження походження кожного коду. Структура таблиці спроєктована так, щоб уникати дублювання даних: один і той самий код зберігається лише один раз, незалежно від кількості сканувань, які були з ним пов’язані. Через зовнішній ключ відбувається зв’язок з таблицею scan_info, де зберігається історія спроб його розпізнавання.

Окрему роль відіграє таблиця product_info, яка містить додаткові відомості про товар, отримані із зовнішніх джерел. Вона не дублює інформацію з попередніх таблиць, а розширює її. У цій таблиці зберігаються текстові описи, назви товарів, бренди, категорії, країни виробництва та інші метадані, які можуть бути отримані за допомогою сторонніх API або баз даних. До кожного запису додається також інформація про джерело — наприклад, назва API, з якого було отримано дані. Це важливо для забезпечення прозорості походження інформації та можливості її верифікації у разі потреби. Зв’язок із таблицею product_codes здійснюється через значення коду, що дозволяє швидко знайти інформацію про конкретний товар, виходячи зі значення штрихкоду.

Завдяки логічним зв’язкам між таблицями забезпечується ефективна організація даних, де одна таблиця не дублює іншу, а розширює її функціональність. Наприклад, для одного штрихкоду може існувати кілька записів сканувань, що дозволяє аналізувати поведінку користувачів або стабільність розпізнавання конкретного коду. Так само один запис у таблиці product_info може обслуговувати кілька кодів, які вказують на однаковий товар, або бути пов’язаний з одним конкретним кодом.

Розроблена архітектура бази даних відповідає принципам нормалізації. Це дозволяє мінімізувати надлишковість, уникнути логічних помилок та забезпечити ефективне виконання SQL-запитів. Крім того, така структура легко масштабується: у майбутньому можна безболісно додати нові таблиці, наприклад, для зберігання зображень, користувацьких налаштувань, історії змін, або ж для інтеграції з додатковими джерелами інформації.

Представлена ER-діаграма на рисунку 2.4.1 відображає взаємозв’язки між основними сутностями. Вона дозволяє візуалізувати, як саме дані зберігаються, які зв’язки існують між таблицями, та як реалізована логічна модель, що лежить в основі всього функціонування системи. З цієї діаграми можна побачити, що всі компоненти бази взаємопов’язані та взаємодіють між собою у межах чітко визначеної логіки, що значно спрощує подальшу розробку, супровід і масштабування програмного забезпечення.

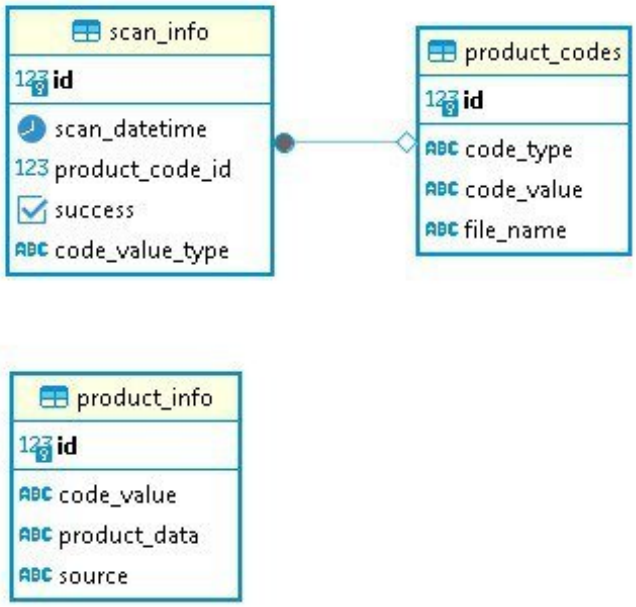

Рисунок 2.4.1 ER-діаграма сутностей бази даних

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДОДАТКУ

3.1 Технології та інструменти розробки

Для розробки програмного рішення був використаний ретельно підібраний стек технологій та інструментів, кожен з яких забезпечував ефективність, якість і надійність.

Java – мова програмування, що була обрана для розробки додатку. Це сучасна об'єктно-орієнтована мова програмування, яка забезпечує високу продуктивність та надійність. Java дозволила реалізувати складну бізнес-логіку та взаємодію з базою даних завдяки своїй універсальності та можливості використання доступних бібліотек [4, 5].

Spring Boot – потужний фреймворк, що спрощує налаштування та розгортання Java додатків. Він допоміг швидко налаштувати та реалізувати RESTful Web API, а також інтегруватися з базою даних [6].

PostgreSQL – Для проекту була обрана реляційна база даних PostgreSQL. Вона забезпечує високу продуктивність та надійність, а також підтримує велику кількість одночасних запитів [7].

Postman – програма, яка використовувалася для тестування розробленого додатку. Це інтуїтивно зрозумілий інструмент, який дозволив ефективно перевіряти роботу різних RESTful сервісів через ручне тестування, забезпечуючи правильність функціонування API [8].

Flyway – фреймворк, який був вибраний для управління міграціями бази даних. Цей інструмент автоматизує процеси оновлення схем бази даних, забезпечуючи їхню узгодженість і стабільність на всіх етапах розробки [9].

ZXing – бібліотека для роботи з EAN/UPC, Code 128 та QR-кодами. Цей інструмент дозволяє розпізнавати та генерувати штрихкоди, що є важливим елементом у функціональності додатку [10].

3.2 Реалізація функціоналу додатку

У цьому розділі буде розглянуто детальну реалізацію ключових класів, які використовуються в додатку. Будуть описані класи, відповідальні за збереження та обробку даних про подорожі, а також класи для взаємодії з користувачем, включаючи форми вводу та відображення інформації. Перед реалізацією логіки важливо організувати класи відповідно до їхньої ролі у структурі додатку [11].

На початку необхідно додати всі потрібні залежності для налаштування проекту. Це забезпечить підключення необхідних бібліотек для коректної роботи програми.

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.2.5'
    id 'io.spring.dependency-management' version '1.1.4'
}

group = 'com.example.coderecognizer'
version = '0.0.1-SNAPSHOT'

java {
    sourceCompatibility = '21'
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.flywaydb:flyway-core'
    runtimeOnly 'org.postgresql:postgresql'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'
    implementation 'org.springframework.boot:spring-boot-starter-validation'
    implementation 'com.google.zxing:core:3.2.1'
    implementation files('libs/zxing-javase.jar')
}

tasks.named('test') { Task it ->
    useJUnitPlatform()
}
```

Рисунок 3.2.1 – Підключення залежностей для збірки додатку

Наступним кроком є створення міграцій для бази даних. Це дозволить автоматично створювати необхідні таблиці та колонки при запуску додатку.

Перша міграція застосовується створення таблиць product_codes та scan_info.

```sql
-- Створення таблиці product_codes
CREATE TABLE product_codes (
    id SERIAL PRIMARY KEY,
    code_type VARCHAR(50) CHECK (CHAR_LENGTH(code_type) BETWEEN 4 AND 50),
    code_value TEXT NOT NULL,
    file_name VARCHAR(255) CHECK (CHAR_LENGTH(file_name) BETWEEN 2 AND 255)
);

-- Створення таблиці scan_info
CREATE TABLE scan_info (
    id SERIAL PRIMARY KEY,
    scan_datetime TIMESTAMP,
    product_code_id BIGINT,
    success BOOLEAN,
    code_value_type VARCHAR(15),
    CONSTRAINT fk_product_code FOREIGN KEY (product_code_id) REFERENCES product_codes(id) ON DELETE CASCADE
);
```

Рисунок 3.2.2 – Файл міграції для створення таблиць product_codes та scan_info

Друга міграція слугує для створення таблиці product_info. Створення таблиці product_info: Таблиця для збереження інформації про продукти, включаючи значення коду, дані про продукт та джерело інформації.

```sql
-- Створення нової таблиці для зберігання інформації про продукт
CREATE TABLE product_info (
    id SERIAL PRIMARY KEY,
    code_value VARCHAR(255) NOT NULL,
    product_data JSONB NOT NULL,
    source VARCHAR(255) NOT NULL
);
```

Рисунок 3.2.3 – Файл міграції для створення таблиці product_info

Клас CodeController має анотацію @RestController, що позначає його як контролер, який обробляє HTTP-запити. CodeController є головним контролером,

який відповідає за прийом запитів від користувача, їх обробку та повернення відповідей. Основними завданням класу CodeController є прийом HTTP-запитів від користувачів та взаємодія з сервісним модулем для розшифрування штрих-кодів та пошуку інформації про продукти для формування відповідей для користувачів.

```java
@Validated
@RestController
@RequiredArgsConstructor
@RequestMapping("/recognizer")
public class CodeController {
    private final CodeRecordService codeRecordService;
    private final BarcodeInfoReturner barcodeInfoReturner;

    @GetMapping("/decode")
    public ResponseEntity<String> getFile(@RequestPart("file") MultipartFile file) throws InvalidImageFormatException {
        String result = codeRecordService.proccess(file);
        return ResponseEntity.ok(result);
    }

    @GetMapping("/searchProduct")
    public ResponseEntity<String> checkProductOnOpenFood(@RequestParam("codeInfo") String codeInfo) {
        String jsonInfo = barcodeInfoReturner.getProductInfoFromAllAPIs(codeInfo);
        return ResponseEntity.ok(jsonInfo);
    }

    @GetMapping("/decodeAndSearch")
    public ResponseEntity<String> decodeWith(@RequestPart("file") MultipartFile file) throws InvalidImageFormatException {
        String result = codeRecordService.proccess(file);
        String json = barcodeInfoReturner.getProductInfoFromAllAPIs(result);
        HttpHeaders headers = new HttpHeaders();
        headers.add(HttpHeaders.CONTENT_DISPOSITION,  headerValue: "attachment; filename=info.json");
        return ResponseEntity.ok()
                .headers(headers)
                .contentType(MediaType.APPLICATION_JSON)
                .body(json);
    }
}
```

Рисунок 3.2.4 – Клас CodeController

ImageFormatValidator відповідає за перевірку формату зображення. Він дозволяє визначити, чи є формат зображення допустимим для обробки.

```java
@Service
public class ImageFormatValidator {
    private final List<String> validFormats = Arrays.asList("JPEG", "PNG", "GIF");

    private String recognizeFormat(MultipartFile file) {
        try {
            ByteArrayInputStream bis = new ByteArrayInputStream(file.getBytes());
            BufferedImage image = ImageIO.read(bis);
            return getImageFormat(image);
        } catch (IOException | EmptyImageException | InvalidImageFormatException e) {
            e.printStackTrace();
        }
        return "unknown image";
    }

    private String getImageFormat(BufferedImage image) throws EmptyImageException, InvalidImageFormatException {
        if (image == null) {
            throw new EmptyImageException();
        }
        String formatName = ImageIO.getImageReadersByMIMEType("image/jpeg").next().getOriginatingProvider().getFormatNames()[0];

        if (!formatName.equals("JPEG") && !formatName.equals("PNG") && !formatName.equals("GIF")) {
            throw new InvalidImageFormatException(formatName);
        }
        return formatName;
    }

    public boolean isValidImage(MultipartFile file) {
        return validFormats.contains(recognizeFormat(file).toUpperCase());
    }
}
```

Рисунок 3.2.5 – Клас для перевірки допустимості зображення

CodeRecordService відповідає за розшифрування EAN/UPC, Code 128 та QR-кодів та збереження у базу даних. Основним методом є функція proccess(MultipartFile file), яка обробляє вхідний файл, розшифровує штрих-код, визначає його тип, зберігає інформацію про код та сканування у базу даних та викликає методи для розшифровки коду.

```java
public String proccess (MultipartFile file) throws InvalidImageFormatException {
    String decryptedResult = decryptor.decryptCode(file);
    ValueType valueType = valueTypeFinder.analyzeFileType(file);

    String[] parts = decryptedResult.split( regex: ": ",  limit: 2);
    if (parts.length < 2) {
        throw new RuntimeException("Invalid decrypted result format");
    }
    String codeValue;
    String codeType = parts[0];
    codeValue = parts[1];

    if(valueType == ValueType.JSON){
        codeValue = parseJson(codeValue);
    }
    ProductCodeDto productCode = new ProductCodeDto();
    productCode.setCodeType(codeType);
    productCode.setCodeValue(codeValue);
    productCode.setFileName(file.getOriginalFilename());
    ProductCodeDto saved = productService.save(productCode);

    ScanInfoDto scanInfoDto = new ScanInfoDto();
    scanInfoDto.setValueType(valueType);
    scanInfoDto.setScanDateTime(LocalDateTime.now());
    scanInfoDto.setProductCode(productCodeMapper.toProductCodeEntity(saved));
    scanInfoDto.setSuccess(true);
    scanInfoService.save(scanInfoDto);
    return productCode.getCodeValue();
}
```

Рисунок 3.2.6 – Збереження розшифрованого коду у базі даних

CodeDecryptor використовує бібліотеку zxing для розшифрування штрих-кодів. Він підтримує декілька типів штрих-кодів, включаючи QR-коди, EAN-8, EAN-13 та Code 128.

Основні методи: decryptCode(MultipartFile file): Розшифровує штрих-код з вхідного файлу та повертає його тип та значення. Використовує ImageFormatValidator для перевірки формату зображення.

Читає зображення та розшифровує штрих-код за допомогою бібліотеки ZXing. Повертає тип та значення штрих-коду або викидає відповідне виключення у разі помилки.

```
public String decryptCode(MultipartFile file) throws InvalidImageFormatException {
    try {
        if(!formatRecognizer.isValidImage(file))
            throw new InvalidImageFormatException("InvalidImageFormat");
        BufferedImage image = ImageIO.read(new ByteArrayInputStream(file.getBytes()));
        Map<DecodeHintType, Object> hints = new EnumMap<>(DecodeHintType.class);
        hints.put(DecodeHintType.TRY_HARDER, Boolean.TRUE);
        hints.put(DecodeHintType.POSSIBLE_FORMATS, EnumSet.allOf(BarcodeFormat.class));
        MultiFormatReader reader = new MultiFormatReader();
        BinaryBitmap binaryBitmap = new BinaryBitmap(new HybridBinarizer(new BufferedImageLuminanceSource(image)));
        Result result = reader.decode(binaryBitmap, hints);
        BarcodeFormat barcodeFormat = result.getBarcodeFormat();

        switch (barcodeFormat) {
            case QR_CODE:
                return "QR Code: " + result.getText();
            case EAN_8:
                EAN8Reader ean8Reader = new EAN8Reader();
                result = ean8Reader.decode(binaryBitmap);
                return "EAN-8: " + result.getText();
            case EAN_13:
                EAN13Reader ean13Reader = new EAN13Reader();
                result = ean13Reader.decode(new BinaryBitmap(new HybridBinarizer(new BufferedImageLuminanceSource(image))));
                return "EAN-13: " + result.getText();
            case CODE_128:
                return "Code 128: " + result.getText();
            default:
                throw new RuntimeException("Unsupported Code Format");
        }
    } catch (IOException | NotFoundException e) {
        e.printStackTrace();
        throw new RuntimeException("Erorr decoding this code!");
    } catch (FormatException e) {
        throw new RuntimeException("Bad image format");
    }
}
```

Рисунок 3.2.7 – Клас для розшифурвання отриманого коду

getProductInfoFromAllAPIs є головним методом класу BarcodeInfoReturner, який шукає інформацію про продукт за штрих-кодом у всіх доступних API. Якщо інформація знайдена, зберігає її у базу даних та повертає користувачу.

```
public String getProductInfoFromAllAPIs(String barcode) {
    Optional<ProductInfo> productInfoOptional = productInfoService.findByCodeValue(barcode);
    if (productInfoOptional.isPresent())
        return productInfoOptional.get().getProductData();
    for (Map.Entry<String, String> entry : API_URLS.entrySet()) {
        String apiUrl = String.format(entry.getValue(), barcode);
        String response = getProductInfo(apiUrl);
        if (response != null && !response.isEmpty()) {
            productInfoService.saveProductInfo(barcode, response, entry.getKey());
            return response;
        }
    }
    throw new RuntimeException("Product not found on any API");
}
```

Рисунок 3.2.8 – Пошук інформації про продукт за штрих-кодом

ProductCodeService відповідає за управління об'єктами ProductCode. Він забезпечує операції збереження та отримання даних про штрих-коди.

Основні методи: listAll(): Повертає список всіх об'єктів ProductCodeDto. save(ProductCodeDto product): Зберігає об'єкт ProductCodeDto у базу даних і повертає збережений об'єкт.

ProductInfoService відповідає за управління об'єктами ProductInfo. Він забезпечує збереження інформації про продукти та пошук продуктів за штрих-кодом. Основні методи: saveProductInfo(String codeValue, String productData, String source): Зберігає об'єкт ProductInfo у базу даних з відповідним значенням коду, даними продукту та джерелом. findByCodeValue(String codeValue): Повертає об'єкт ProductInfo за значенням коду, якщо він існує у базі даних.

ScanInfoService відповідає за управління об'єктами ScanInfo. Він забезпечує операції збереження та отримання даних про сканування штрих-кодів. Основні методи: listAll(): Повертає список всіх об'єктів ScanInfoDto. save(ScanInfoDto scanInfoDto): Зберігає об'єкт ScanInfoDto у базу даних і повертає збережений об'єкт.

MainExceptionHandler відповідає за глобальну обробку винятків, що можуть виникнути під час виконання запитів.

Основні завдання класу MainExceptionHandler полягають в обробці всіх видів винятків та у формуванні відповідних повідомлень про помилки для користувачів.

```java
@RestControllerAdvice
public class MainExceptionHandler {
    @ExceptionHandler(value = {Exception.class})
    public ResponseEntity<Map<String, List<String>>> anotherException(Exception ex) { return getErorMap(ex); }

    private ResponseEntity<Map<String, List<String>>> getErorMap(Exception ex) {
        Map<String, List<String>> map = new HashMap<>();
        map.put("errors", Collections.singletonList(ex.getMessage()));
        return new ResponseEntity<>(map, HttpStatus.BAD_REQUEST);
    }
}
```

Рисунок 3.2.9 – Клас MainExceptionHandler для обробки виключень

3.3 Тестування програмного забезпечення

Важливим етапом розробки програмного забезпечення є забезпечення якості та перевірка надійності створеного продукту. У процесі роботи було проведено ручне тестування за допомогою інструменту Postman.

Ручне тестування — це метод тестування програмного забезпечення, при якому тестові випадки виконуються вручну без використання автоматизованих інструментів. Це дозволяє виявити помилки, які можуть бути пропущені автоматизованими тестами [12].

Для тестування функцій API було створено кілька запитів у Postman. Для перевірки функції пошуку продукту за штрихкодом, створено новий запит у Postman. Після натискання кнопки "Send", отримано відповідь у форматі JSON, що містить інформацію про продукт. Це свідчить про коректну роботу функції пошуку продукту за штрих-кодом.

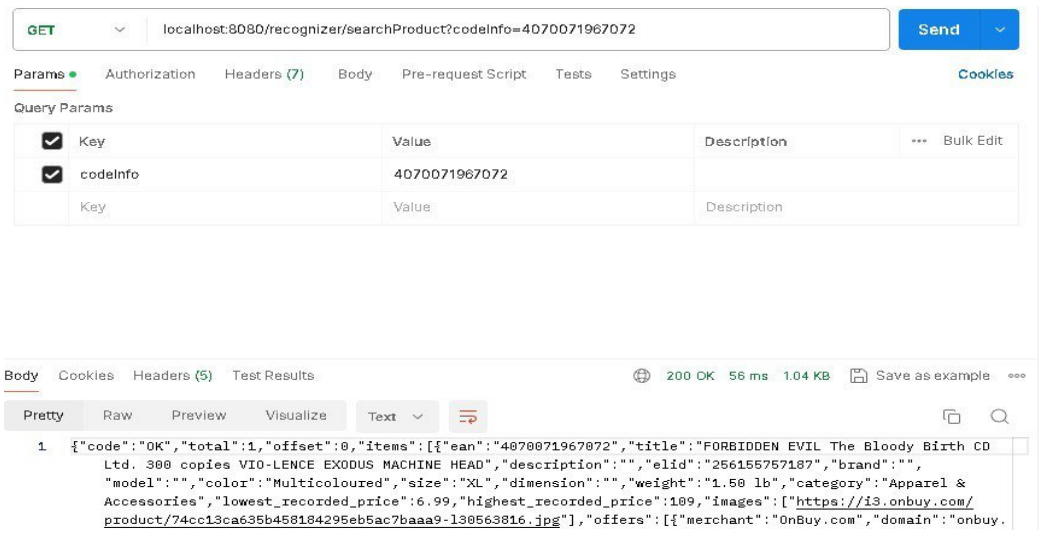

Рисунок 3.3.1 – Результат пошуку інформації про продукт

Для перевірки функції розшифровки штрихкоду із зображення, створено новий запит у Postman. У розділі "Body" обрано тип form-data та додано файл із зображенням штрих-коду. Після надсилання запиту отримано відповідь у форматі JSON із розшифрованими даними штрихкоду. Це підтверджує правильність роботи функції розшифровки.

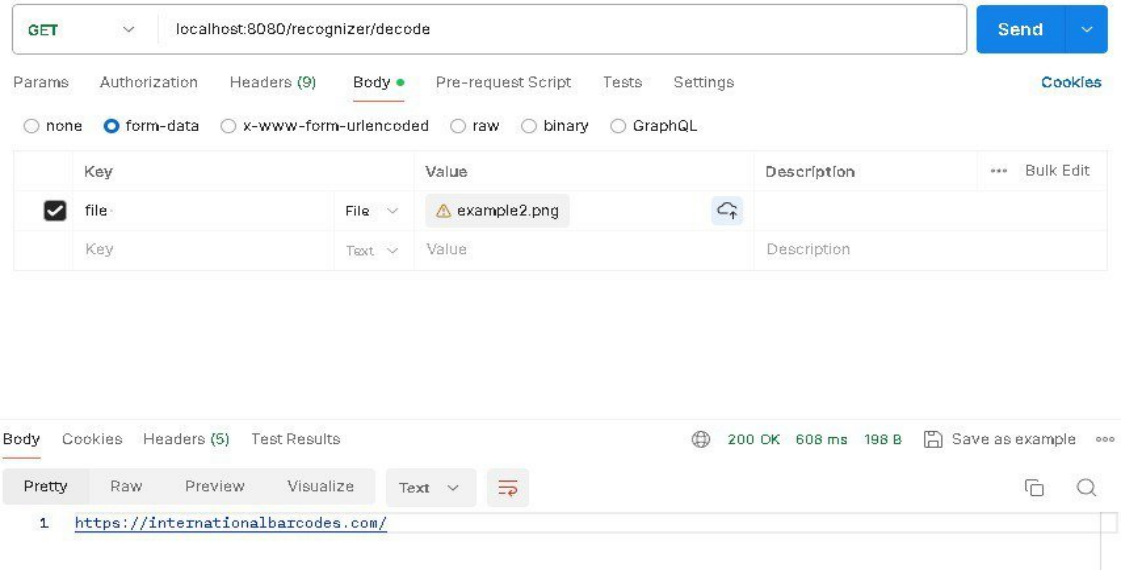

Рисунок 3.3.2 – Результат розшифрування QR-коду

Для перевірки функції розшифровки штрих-коду та пошуку продукту, створено новий запит у Postman.

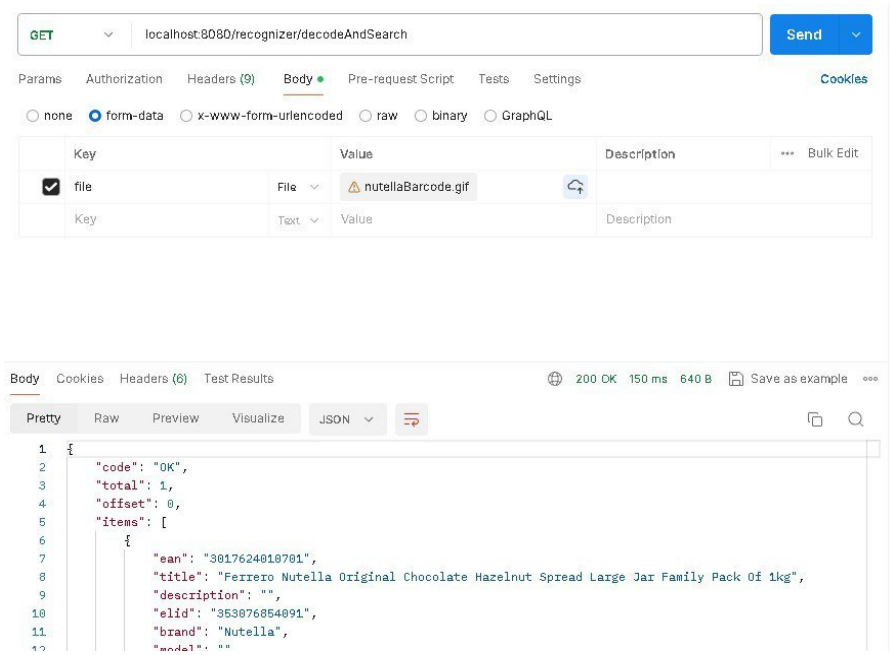

Рисунок 3.3.3 – Результат розшифрування та автоматичного пошуку інформації про продукт

Ручне тестування з використанням Postman дозволило перевірити коректність роботи розроблених функцій, забезпечуючи високу якість та надійність програмного продукту.

3.4 Розробка та проведення Unit-тестування

Unit-тестування є невід'ємною частиною процесу розробки якісного програмного забезпечення. Його основна мета полягає у перевірці окремих частин програми — так званих модулів або компонентів — на правильність їхньої роботи в ізоляції від інших частин системи. Це дозволяє на ранніх етапах виявляти та усувати помилки, що в свою чергу забезпечує стабільність і передбачувану поведінку додатку в цілому [13]. У процесі створення Java-додатку для розпізнавання штрихкодів значна увага приділялася саме модульному тестуванню найважливіших складових логіки програми.

Зважаючи на те, що функціональність розробленого застосунку включає складні операції, такі як обробка зображень, розпізнавання вмісту штрихкодів, взаємодія з зовнішніми сервісами через API, а також подальше збереження декодованої інформації, постала необхідність у ретельному тестуванні сервісних компонентів, які реалізують цю логіку. Для цього було створено відповідний набір unit-тестів, які дозволяють ізолювати і перевірити кожну частину функціоналу окремо, не покладаючись на повноцінне функціонування всієї системи.

Для реалізації тестів було обрано фреймворк JUnit 5, який на сьогодні є одним із найпоширеніших інструментів для модульного тестування у середовищі Java. Його гнучкий синтаксис, розширюваність та інтеграція зі Spring Framework роблять його зручним інструментом для тестування як простих методів, так і складних логічних залежностей [14]. Окрім JUnit, було активно використано бібліотеку Mockito — потужний інструмент для створення підробок (mock-об'єктів), які імітують поведінку реальних залежностей. Завдяки цьому підходу вдалося протестувати компоненти системи, уникаючи при цьому взаємодії з реальними зовнішніми сервісами, такими як база даних чи API сторонніх постачальників. Це дозволяє зменшити час виконання тестів та уникнути можливих збоїв, спричинених зовнішніми факторами.

Окремі методи додатку приймають зображення у вигляді файлів, які надсилаються через HTTP-запити. Для забезпечення тестування таких сценаріїв використовувався клас MockMultipartFile з Spring Framework. Завдяки цьому стало можливим створити імітацію файлу зображення без потреби у реальному завантаженні файлів або запуску вебсерверу.

У межах модульного тестування були охоплені основні компоненти додатку, кожен із яких відповідає за окремий логічний блок:

- Модуль розпізнавання штрихкодів — реалізований у класі CodeDecryptor, відповідає за перетворення зображення на символьне значення, підтримує типи кодування QR, Code128 та EAN/UPC.

- Модуль визначення типу даних — реалізований у класі ValueTypeFinder, виконує аналіз розпізнаного значення та визначає його тип, наприклад, URL, JSON або звичайний текст.
- Модуль збереження інформації — реалізований у сервісі CodeRecordService, відповідає за збереження результатів декодування у базу даних разом з супутніми метаданими.
- Модуль отримання додаткової інформації — реалізований у класі BarcodeInfoReturner, здійснює пошук даних про товар за штрихкодом шляхом взаємодії з зовнішніми API.

3.4.1 Тестування модуля BarcodeInfoReturner

Модуль BarcodeInfoReturner виконує функцію отримання розширеної інформації про товар, ідентифікований за допомогою розпізнаного штрихкоду. Його головне завдання полягає у виконанні запиту до відкритого зовнішнього API, такого як OpenFoodFacts або подібні сервіси, з метою отримання додаткових даних про продукт: його назву, категорію, бренд, харчову інформацію, країну виробництва та інші характеристики. Таким чином, цей компонент фактично виступає посередником між внутрішньою логікою застосунку і публічними базами даних, забезпечуючи користувача максимально повною інформацією про товар без потреби в ручному введенні.

З огляду на те, що функціонування модуля напряму залежить від стабільності мережевого з’єднання, затримок у відповіді, а також доступності зовнішніх сервісів, тестування його функціоналу без ізоляції могло б призводити до непередбачуваних результатів або фальшивих помилок. Щоб уникнути залежності від сторонніх API під час запуску автоматизованих тестів, було використано бібліотеку Mockito для створення імітацій відповідей, що дало змогу сфокусуватись саме на перевірці логіки компонента.

У рамках тестування цього компонента були змодельовані й перевірені наступні основні сценарії:

- Отримання валідної відповіді за відомим кодом: перевірка, що метод повертає повний об’єкт із заповненими полями.
- Відсутність інформації: ситуація, коли зовнішнє API не має запису для вказаного коду — у цьому випадку сервіс повертає null.
- Обробка винятків: симуляція ситуацій із мережею або недоступністю сервісу дозволила переконатися, що винятки не призводять до аварійного завершення роботи додатку.

Тести підтвердили стійкість модуля до збоїв, непередбачуваних ситуацій і відсутності даних. У разі виникнення проблем модуль повертає контрольовані значення або генерує очікувані винятки, які можуть бути оброблені на вищому рівні. Це дозволяє іншим частинам системи, зокрема сервісу збереження даних або інтерфейсу користувача, гнучко реагувати на такі ситуації — або зберігати штрихкод без метаданих, або інформувати користувача про обмежену наявність інформації. Таким чином, модуль BarcodeInfoReturner довів свою надійність, адаптивність до зовнішніх умов та важливу роль у побудові зручного і стабільного користувацького досвіду.

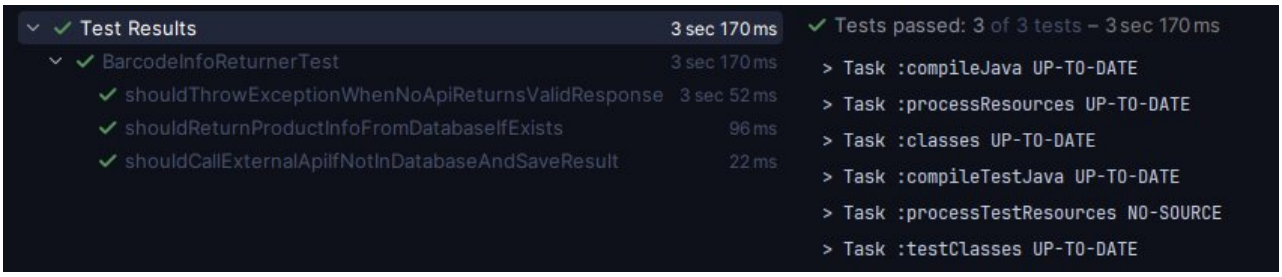

Рисунок 3.4.1.1. Результат виконання тестів для BarcodeInfoReturnerTest

3.4.2 Тестування модуля CodeDecryptor

Компонент CodeDecryptor є одним із центральних функціональних блоків у структурі розробленого додатку. Він відповідає за первинну обробку зображення, що надходить у вигляді файлу від користувача, а також за спробу дешифрувати штрихкод, якщо він наявний. Основна логіка класу полягає в аналізі переданого файлу, перевірці його валідності та, у разі успішного розпізнавання, витяганні символьного значення коду. Підтримуються такі типи штрихкодів, як QR, Code128, EAN та UPC.

Під час тестування були враховані різноманітні можливі варіанти поведінки програми, щоб охопити всі типові та граничні сценарії. Особливу увагу було приділено модульному тестуванню цього компонента через його високу чутливість до якості вхідних даних. Під час тестування були враховані різноманітні варіанти файлів, щоб максимально охопити всі можливі сценарії, з якими може зіткнутись користувач:

- Розпізнавання зображення, що містить коректний штрихкод — основний позитивний сценарій.
- Спроба обробки файлу, який не містить жодного коду, наприклад, довільне зображення або текстовий файл.
- Пошкоджене або порожнє зображення: імітація ситуації, коли файл не містить зображення або не підлягає декодуванню.
- Завантаження файлу неправильного типу
- Кілька кодів на одному зображенні — перевірка на пріоритетність або правильне розпізнавання першого з них.

Для реалізації тестів активно використовувався клас MockMultipartFile, що дозволяє створювати імітаційні файли в пам’яті без завантаження фізичних ресурсів. Це дозволило забезпечити швидкість виконання тестів і уникнути додаткових залежностей. Крім того, було застосовано Mockito для створення

підставних об’єктів, що дозволило ізолювати логіку самого CodeDecryptor від сторонніх сервісів.

Результати тестування підтвердили високу стабільність компонента та його стійкість до різноманітних помилкових ситуацій. У випадку некоректного вводу модуль не викликає критичних винятків, натомість повертає зрозумілі повідомлення про помилку, які можуть бути використані для відповідного інформування користувача. Це підвищує зручність використання застосунку та зменшує ризик непередбачених ситуацій.

Варто також зазначити, що CodeDecryptor закладає основу для масштабованості: у разі потреби до нього легко можна додати підтримку нових типів кодів або інші алгоритми обробки. Такий підхід забезпечує якісну обробку зображень на стороні сервера та зменшує ймовірність помилок під час декодування.

Рисунок 3.4.2.1 Результат виконання тестів для CodeDecryptor

3.4.3 Тестування модуля CodeRecordService

CodeRecordService — сервісний компонент, що відповідає за збереження розпізнаної інформації у базу даних. Після того як штрихкод було оброблено і отримано розширену інформацію про продукт, ця сутність має бути збережена з усіма відповідними атрибутами — включаючи тип коду, його значення, дату сканування, користувача, інформацію про товар тощо. Крім того, цей клас може

перевіряти, чи код уже існує, або оновлювати наявну інформацію, якщо вона змінилася.

Основні сценарії, що були протестовані в межах цього модуля:

- Створення нового запису: перевірка, що після передачі коректного об’єкта відбувається збереження всіх атрибутів до бази даних.
- Обробка некоректних або порожніх даних: змодельовано випадки, коли частина полів відсутня або значення є недійсним.
- Перевірка на дублювання: перевірено, що при повторному збереженні одного й того ж коду не створюється новий запис, якщо такий вже є.

Усі зовнішні залежності, такі як CodeRecordRepository, UserService та BarcodeInfoReturner, були замінені на моки, що дозволило сфокусуватися на внутрішній логіці методу збереження.

Тести показали, що сервіс обробляє як прості, так і складні сценарії. Завдяки перевіркам перед збереженням дублювання не виникає, а структура збереженого об’єкта відповідає очікуванням. Також перевірено, що винятки при некоректних даних обробляються через спеціалізовані повідомлення.

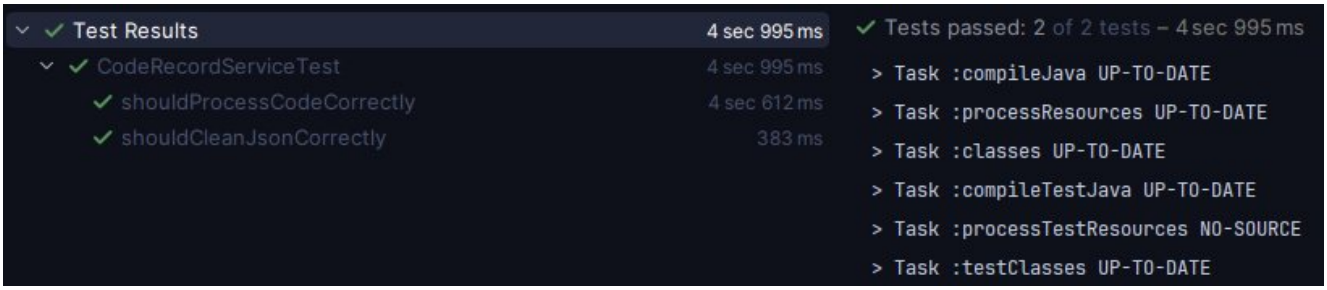

Рисунок 3.4.3.1 Результат виконання тестів для CodeRecordService

3.4.4 Тестування модуля ValueTypeFinder

Модуль ValueTypeFinder виконує функцію класифікації розпізнаного коду за його структурою. На основі переданого рядка він визначає тип штрихкоду — наприклад, QR-код, EAN-13, UPC-A або Code 128. Така класифікація необхідна для правильної подальшої обробки, збереження та інтерпретації зчитаних даних.

Оскільки ValueTypeFinder є чистою функцією, яка не має зовнішніх залежностей і працює лише з переданим вхідним значенням, для його тестування застосовувався підхід unit-тестування з використанням фіксованих вхідних значень. Було протестовано наступні сценарії:

- Передача коректних значень для типів: EAN-13, UPC-A, QR, Code 128 — перевірялася точність класифікації;
- Передача некоректно сформованих кодів
- Граничні випадки, коли передане значення не підпадає під жоден з визначених форматів.

Результати тестування підтвердили, що модуль правильно ідентифікує підтримувані формати кодів, а всі невідомі або некоректні значення класифікуються як UNKNOWN. Такий підхід гарантує стабільну поведінку системи при обробці нетипових або помилкових даних.

Оскільки ValueTypeFinder не має залежностей і працює на основі чітких правил валідації, його тестування не вимагало використання моків. Модуль пройшов усі перевірки й може вважатися надійним та ізольованим компонентом системи.

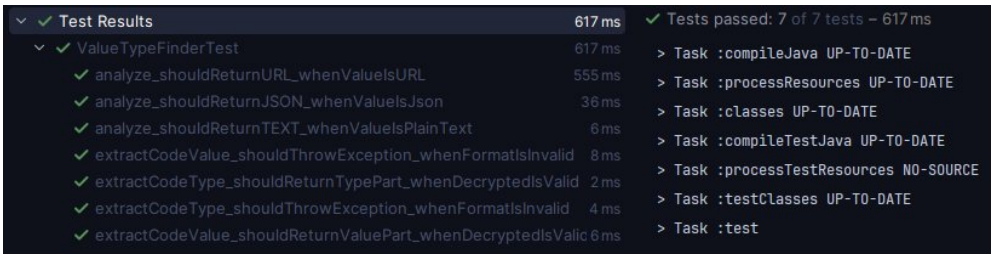

Рисунок 3.4.4.1. Результат виконання тестів для ValueTypeFinder

3.4.5 Узагальнення результатів тестування

У результаті проведення модульного тестування основних компонентів Java-додатку для розпізнавання штрихкодів можна зробити висновки щодо коректності реалізованої логіки, стійкості до помилок та загальної стабільності роботи системи. Тестування охоплювало всі ключові модулі, які реалізують окремі функціональні частини повного циклу обробки штрихкодів.

Було ретельно перевірено компонент, що відповідає за отримання розшифрованої інформації про товар з відкритих джерел, відомий як BarcodeInfoReturner. Крім того, протестовано CodeDecryptor, який виконує дешифрування кодів із зображень, а також CodeRecordService, що зберігає як сам штрихкод, так і пов’язані з ним метадані. Допоміжний інструмент ValueTypeFinder теж не був залишений поза увагою: він визначає тип штрихкоду відповідно до його структурних характеристик.

Під час тестування були враховані як звичайні сценарії, де компоненти отримують коректні вхідні дані, так і виняткові випадки. До них належить передача некоректної або порожньої інформації, моделювання ситуацій із відсутністю записів у зовнішніх джерелах, або обробка нестандартних форматів штрихкодів. Такі підходи дозволили переконатися, що система здатна працювати надійно не лише в оптимальних умовах, а й у стресових чи непередбачуваних ситуаціях.

Особливу увагу було приділено перевірці реакції на помилки. Модулі мали не лише правильно обробляти дані, а й грамотно реагувати на ситуації, коли декодування неможливе або інформація відсутня. Для ізоляції логіки кожного окремого компонента активно використовувалася бібліотека Mockito, яка дала змогу створювати підставні об'єкти та замінити реальні залежності: базу даних, API-запити, файлову систему. Це дозволило зосередитися саме на внутрішній логіці модулів, не враховуючи фактори зовнішнього середовища [15].

Структура програми в цілому відповідає принципам інверсії залежностей, що забезпечує її гнучкість і полегшує процес тестування. Усі сервіси мають чітко

визначені зони відповідальності, що робить їх придатними для незалежної перевірки без потреби розгортання повноцінного середовища. Цей підхід значно підвищує рівень автоматизації тестування та дозволяє легко вбудовувати модульні тести в CI/CD-процеси [16].

Підтвердженням успішного проходження тестів є результати, отримані за допомогою вбудованого механізму тестування. Загалом було виконано 16 тестів, з яких усі пройшли успішно, без жодного збою. Сумарний час виконання склав 5.332 секунди, що вказує на високу продуктивність і швидкість тестового покриття. Такий рівень підготовки і стабільності програмного забезпечення свідчить про його готовність до інтеграції та розгортання у робочому середовищі. Структура додатку дозволяє легко масштабувати його функціональність, додавати підтримку нових типів штрихкодів або джерел даних, не змінюючи фундаментальної логіки.

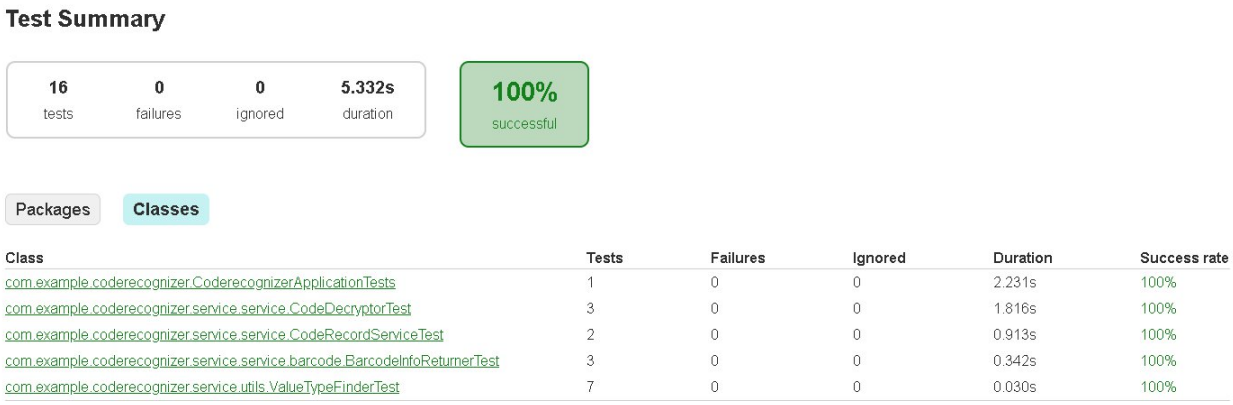

Рисунок 3.4.5.1. Результати модульного тестування у вигляді зведеного звіту

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Долікарська допомога при пораненнях

Надання долікарської допомоги при пораненнях є важливим компонентом загальної медичної допомоги, що охоплює невідкладні заходи на місці події до прибуття медичних фахівців. Цей процес включає використання різних методів та знань, які дозволяють стабілізувати стан постраждалого, мінімізувати ризик ускладнень та підготувати його до подальшого лікування.

Поранення – це пошкодження шкіри, слизових оболонок або глибоких тканин, яке супроводжується болем і кровотечею, і має вигляд зяючого отвору. Рани можуть бути вогнепальні, різані, рубані, колоті, вдарені, рвані, вкушені, розчавлені. Всі рани, крім хірургічних, вважаються первісно інфікованими, оскільки мікроби можуть потрапити до рани разом з предметом, яким було нанесено пошкодження, а також через землю, шматки одягу, повітря і при контакті з раною руками. В залежності від напрямку витікання крові з пошкоджених судин розрізняють зовнішні і внутрішні кровотечі.

Зовнішні кровотечі характеризуються витіканням крові назовні через пошкодження шкіри. Внутрішні кровотечі є дуже небезпечними і часто смертельними. У постраждалого різко блідне обличчя, частішає пульс, з'являється загальна слабкість, запаморочення, задуха, спрага, а на шкірі можуть утворюватися чорні плями у вигляді висипки.

Навіть незначна рана може становити серйозну загрозу для життя постраждалого, оскільки вона може стати джерелом інфекції різними мікроорганізмами, а також супроводжуватися значною кровотечею. Основна мета профілактики таких ускладнень під час надання першої медичної допомоги полягає в якнайшвидшому накладенні стерильної пов'язки на рану, дотриманні правил асептики й антисептики, а також у зупинці кровотечі [17].

При наданні першої допомоги у випадках поранення категорично забороняється:

- промивати рану;

- видаляти будь-які сторонні тіла;

- класти в рану вату, змочену йодом.

Дотримання цих правил є критично важливим для запобігання подальшим ускладненням та інфікуванню рани, що може значно погіршити стан постраждалого.

Обробка рани – важливий крок у наданні домедичної допомоги, який спрямований на запобігання інфекціям і мінімізацію ускладнень. Важливо дотримуватися належних методів та процедур для забезпечення безпеки постраждалого.

Антисептикою називається система заходів, спрямованих на зменшення кількості мікробів та їхнє знищення у рани. Розрізняють механічну, фізичну, хімічну і біологічну антисептику. Механічна антисептика полягає у первинній хірургічній обробці ран.

Фізична антисептика полягає у застосуванні таких методів, при яких створюються несприятливі умови у рані для виживання мікробів.

Хімічна антисептика заснована на застосуванні різних лікарських засобів, що мають протимікробну дію. Найбільш широко застосовуються такі антисептики, як настойка йоду, етиловий спирт, розчини хлораміну, риванолу та перманганату калію. До біологічних антисептиків відносяться антибіотики, що використовуються для профілактики і лікування раневої інфекції.

Пов’язка – це перев’язувальний матеріал, яким закривають рану. Процес накладання пов’язки на рану називають перев’язкою. Пов’язки використовують для закріплення перев’язувального матеріалу, тиску на яку-небудь частину тіла — в основному з метою зупинки кровотечі, попередження набряку тканин або утримання кінцівки або іншої частини тіла в нерухомому стані. Відповідно розрізняють пов’язки закріплюючі, тиснучі та знерухомлюючі [18].

Накладаючи пов’язку, треба намагатися не завдати зайвого болю постраждалому. Бинт тримають у правій руці і розкручують його, не відриваючи

від пов’язки, яку підтримують лівою рукою. Бинтують зліва направо, кожним наступним обертом перекриваючи попередній наполовину.

4.2 Компоновка приміщення та обладнання цеху у відповідності з вимогами техніки безпеки, санітарних та пожежних вимог

Створення здорових та безпечних умов праці починається з правильного вибору майданчика для розміщення підприємства та раціонального розташування на ньому виробничих, допоміжних та інших будівель і споруд.Площа промислового підприємства визначається за формулою:

$$S = N*a*b/\eta,$$

де N - кількість працюючих на даному підприємстві;

a - площа забудови, що припадає на одного працюючого (а = 15 - 20 м2/люд.);

b - площа, зайнята транспортними шляхами;

η - коефіцієнт зайнятості площі (0,35-0,50).

Важлива роль у створенні безпечних умов праці належить новій техніці та технологіям.

Сьогодні більшість робіт в промисловій галузі проводиться з широким використанням різноманітного технологічного обладнання.

Безпечна праця з використанням технологічного обладнання можлива лише тоді, коли його конструкція відповідає вимогам техніки безпеки, виробничої санітарії та заходам протипожежної безпеки.

Методів забезпечення безпеки обладнання існує дуже багато, і з часом вони постійно розширюються та вдосконалюються. Усі методи забезпечення безпеки обладнання розподіляють на загальні та часткові.

До загальних методів належать механізація і автоматизація процесів, дистанційне управління і спостереження, блокування і сигналізація, надійність і міцність технологічного обладнання [19].

Призначення часткових методів полягає у захисті обладнання від певної небезпеки. При проектуванні і конструюванні технологічного обладнання та технологічних процесів велике значення мають контрольно-вимірювальні засоби.

Контрольно-вимірювальні пристрої мають розміщуватись так, щоб оператор не знаходився надто близько до небезпечної зони технологічного обладнання, не був змушений надмірно напружуватися або порушувати рівновагу тіла при управлінні технологічним процесом. Контрольно-вимірювальні пристрої мають бути влаштовані таким чином, щоб звести помилки оператора до мінімального значення.

Робочим місцем є ділянка приміщення або території підприємства, де постійно або періодично перебувають працівники для виконання технологічних операцій.

Якщо окремі операції виконуються на різних ділянках приміщення, то робочим місцем слід вважати все приміщення. Простір, який охоплює всю площу робочого місця і має висоту 2 м над її рівнем, називається робочою зоною. Від правильної організації робочого місця в значній мірі залежать умови праці і її ефективність [20].

Основним елементом організації робочого місця вважають його планування, визначення основних і допоміжних робочих рухів, компоновку обладнання, вибір інструменту, пристосувань та інвентаря, а також допоміжних пристроїв, які забезпечують безпеку праці.

Для попередження виникнення пожеж на промислових підприємствах має велике значення виконання правил пожежної безпеки.

До основних профілактичних заходів можна віднести:

- Заміна небезпечних технологічних процесів в пожежнім відношенні – на менш небезпечні.
- Герметизація.
- Розміщення джерел вогню тільки в окремому приміщенні.
- Використання вогнестійких будов.

Таким чином, пожежна безпека повинна забезпечуватися шляхом проведення організаційних, технічних та інших заходів, які спрямовані на попередження пожеж, забезпечення безпеки людей, зниження можливих майнових втрат і зменшення негативних екологічних наслідків у разі їх виникнення, створення умов для швидкого виклику пожежних підрозділів та успішного гасіння пожеж [21].

ВИСНОВКИ

Результатом дипломного проєкту є система розпізнавання та декодування штрих-кодів та QR-кодів, розроблена з використанням технологій Spring Boot та бібліотек для обробки зображень і роботи з кодами. Основною метою цього проєкту було створення ефективного та надійного програмного рішення для автоматизованого розпізнавання кодів з зображень та надання відповідної інформації про продукти.

Під час розробки дипломного проєкту були розглянуті основні принципи роботи з зображеннями, обробки штрих-кодів та QR-кодів, а також проаналізовано сучасні підходи та технології для реалізації даних задач.

Розроблена система забезпечує зручний та інтуїтивно зрозумілий інтерфейс для користувачів, дозволяючи швидко отримувати інформацію про продукти за допомогою завантаження зображень з кодами. Під час тестування було забезпечено стабільну функціональність системи та відповідність очікуванням.

Цей проєкт демонструє можливості сучасних технологій для автоматизації процесів розпізнавання та обробки кодів, що може бути корисним для широкого спектру застосувань, включаючи роздрібну торгівлю, логістику та управління запасами.

Крім того, впровадження такої системи дозволяє підвищити ефективність та швидкість обробки інформації, що є важливим фактором у сучасному бізнесі.

Отже, реалізація даного проєкту сприяла глибшому розумінню принципів роботи з зображеннями та кодами, а також дала можливість опанувати нові технології та підходи в програмуванні, що буде корисним у подальшій професійній діяльності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Штрих-код [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Штрих-код (дата звернення: 14.05.2024).

2. IEEE Computer Society. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0. Chapter 2: Software Architecture [Електронний ресурс]. – Режим доступу: https://ieeecs-media.computer.org/media/education/swebok/swebok-v4.pdf (дата звернення: 13.06.2025).

3. Bass L., Clements P., Kazman R. Software Architecture in Practice. – Addison-Wesley, 2012. – 512 p.

4. Мова програмування Java [Електронний ресурс]. – Режим доступу: https://www.oracle.com/java/ (дата звернення: 14.05.2024).

5. Еккель Б. Філософія Java. – Київ: Видавнича група BHV, 2003. – 1168 с.

6. Spring Boot [Електронний ресурс]. – Режим доступу: https://spring.io/projects/spring-boot (дата звернення: 14.05.2024).

7. PostgreSQL [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/ (дата звернення: 14.05.2024).

8. Postman [Електронний ресурс]. – Режим доступу: https://www.postman.com/ (дата звернення: 14.05.2024).

9. Flyway [Електронний ресурс]. – Режим доступу: https://flywaydb.org/ (дата звернення: 14.05.2024).

10. ZXing [Електронний ресурс]. – Режим доступу: https://github.com/zxing/zxing (дата звернення: 14.05.2024).

11. Fowler M. Refactoring: Improving the Design of Existing Code. – Addison-Wesley, 2018. – 448 p.

12. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. – Wiley, 2011. – 256 p.

13. IEEE Computer Society. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0. Chapter 5: Software Testing [Електронний ресурс]. – Режим доступу: https://ieeecs-media.computer.org/media/education/swebok/swebok-v4.pdf (дата звернення: 13.06.2025).

14. Martin R. Clean Code: A Handbook of Agile Software Craftsmanship. – Prentice Hall, 2008. – 464 p.

15. Meszaros G. xUnit Test Patterns: Refactoring Test Code. – Addison-Wesley, 2007. – 896 p.

16. Koskela L. Test-Driven: TDD and Acceptance TDD for Java Developers. – Manning Publications, 2007. – 296 p.

17. Стеблюк М.І. Цивільна оборона: навч. посіб. – Київ: Знання, 2009. – 209 с.

18. Зеркалов Д.В. Безпека життєдіяльності: навч. посіб. – Київ: Основа, 2011. – 208 с.

19. Гандзюк М.П., Желібо Є.П., Халімовський М.О. Основи охорони праці: підручник. 5-е вид. / за ред. М.П. Гандзюка. – Київ: Каравела, 2023. – 107 с.

20. Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99 [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/rada/show/va042282-99#Text (дата звернення: 14.05.2024).

21. Яремко З.М. Безпека життєдіяльності: навч. посіб. – Львів: Видавничий центр ЛНУ ім. І. Франка, 2005. – 301 с.

ДОДАТКИ

ДОДАТОК А

Програмний код проєкту

Лістинг А1 – Код файлу CodeController.java

```
@Validated
@RestController
@RequiredArgsConstructor
@RequestMapping("/recognizer")
public class CodeController {
    private final CodeRecordService codeRecordService;
    private final BarcodeInfoReturner barcodeInfoReturner;

    @GetMapping("/decode")
    public ResponseEntity<String> getFile(@RequestPart("file")
MultipartFile file) throws InvalidImageFormatException {
        String result = codeRecordService.proccess(file);
        return ResponseEntity.ok(result);
    }
    @GetMapping("/searchProduct")
    public ResponseEntity<String>
checkProductOnOpenFood(@RequestParam("codeInfo") String codeInfo) {
        String jsonInfo =
barcodeInfoReturner.getProductInfoFromAllAPIs(codeInfo);
        return ResponseEntity.ok(jsonInfo);
    }
    @GetMapping("/decodeAndSearch")
    public ResponseEntity<String> decodeWith(@RequestPart("file")
MultipartFile file) throws InvalidImageFormatException {
        String result = codeRecordService.proccess(file);
        String json =
barcodeInfoReturner.getProductInfoFromAllAPIs(result);
        HttpHeaders headers = new HttpHeaders();
        headers.add(HttpHeaders.CONTENT_DISPOSITION, "attachment;
filename=info.json");
        return ResponseEntity.ok()
                .headers(headers)
                .contentType(MediaType.APPLICATION_JSON)
                .body(json);
    }
}
```

Лістинг А1 включає клас для взаємодії з користувачем, включає методи для обробки запитів користувача.

Лістинг А2 – Код файлу BarcodeInfoReturner.java

```
@Component
@RequiredArgsConstructor
public class BarcodeInfoReturner {
    private final ProductInfoService productInfoService;
    private static final Map<String, String> API_URLS = new
HashMap<>();
    static {
```

Продовження лістинга А2

```
        API_URLS.put("OpenFoodFacts",
"https://world.openfoodfacts.org/api/v0/product/%s.json");
        API_URLS.put("Product Open Data",
"https://pod.opendatasoft.com/api/records/1.0/search/?dataset=openda
tasoft-barcodes&q=%s");
        API_URLS.put("UPCItemDB",
"https://api.upcitemdb.com/prod/trial/lookup?upc=%s");
        API_URLS.put("Datakick",
"https://www.gtinsearch.org/api/items/%s");
        API_URLS.put("Open Beauty Facts",
"https://world.openbeautyfacts.org/api/v0/product/%s");
    }
    private String getProductInfo(String Url) {
        try {
            URL url = new URL(Url);
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            if (connection.getResponseCode() == 400)
                return null;
            BufferedReader reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
            StringBuilder response = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                response.append(line);
            }
            if (containsNotFoundMessage(response.toString()))
                return null;
            reader.close();
            connection.disconnect();
            return response.toString();
        }catch (Exception e) {
            return null;
    public String getProductInfoFromAllAPIs(String barcode) {
        Optional<ProductInfo> productInfoOptional =
productInfoService.findByCodeValue(barcode);
        if (productInfoOptional.isPresent())
            return productInfoOptional.get().getProductData();
        for (Map.Entry<String, String> entry : API_URLS.entrySet())
{
            String apiUrl = String.format(entry.getValue(),
barcode);
            String response = getProductInfo(apiUrl);
            if (response != null && !response.isEmpty()) {
 productInfoService.saveProductInfo(barcode, response,
entry.getKey());
                return response;
            }
        }
        throw new RuntimeException("Product not found on any API");
    private boolean containsNotFoundMessage(String response) {
```

Продовження лістинга А2

```
        String notFoundMessage1 =
"\"status\":0,\"status_verbose\":\"product not found\"";
        String notFoundMessage2 = "status\":0";
        return response.contains(notFoundMessage1) ||
response.contains(notFoundMessage2);
    }
}
```

Лістинг А2 визначає клас для отримання інформації про продукт за штрих-кодом.

Лістинг А3 – Код файлу CodeDecryptor.java

```
@Service
@RequiredArgsConstructor
public class CodeDecryptor {
    private final ImageFormatValidator formatRecognizer;
    public String decryptCode(MultipartFile file) throws
InvalidImageFormatException {
        try {
            if(!formatRecognizer.isValidImage(file))
                throw new
InvalidImageFormatException("InvalidImageFormat");
            BufferedImage image = ImageIO.read(new
ByteArrayInputStream(file.getBytes()));
            Map<DecodeHintType, Object> hints = new
EnumMap<>(DecodeHintType.class);
            hints.put(DecodeHintType.TRY_HARDER, Boolean.TRUE);
            hints.put(DecodeHintType.POSSIBLE_FORMATS,
EnumSet.allOf(BarcodeFormat.class));
            MultiFormatReader reader = new MultiFormatReader();
            BinaryBitmap binaryBitmap = new BinaryBitmap(new
HybridBinarizer(new BufferedImageLuminanceSource(image)));
            Result result = reader.decode(binaryBitmap, hints);
            BarcodeFormat barcodeFormat = result.getBarcodeFormat();
            switch (barcodeFormat) {
                case QR_CODE:
                    return "QR Code: " + result.getText();
                case EAN_8:
                    EAN8Reader ean8Reader = new EAN8Reader();
                    result = ean8Reader.decode(binaryBitmap);
                    return "EAN-8: " + result.getText();
                case EAN_13:
                    EAN13Reader ean13Reader = new EAN13Reader();
                    result = ean13Reader.decode(new BinaryBitmap(new
HybridBinarizer(new BufferedImageLuminanceSource(image))));
                    return "EAN-13: " + result.getText();
                case CODE_128:
                    return "Code 128: " + result.getText();
                default:
```

Продовження лістинга А3

```
throw new RuntimeException("Unsupported Code Format");
            }
        } catch (IOException | NotFoundException e) {
            e.printStackTrace();
            throw new RuntimeException("Erorr decoding this code!");
        } catch (FormatException e) {
            throw new RuntimeException("Bad image format");
        }
    }
}
```

Лістинг А3 визначає клас для розшифрування різних типів кодів з файлів зображень.

Лістинг А4 – Код файлу CodeRecordService.java

```
@Service
@RequiredArgsConstructor
public class CodeRecordService {
    private final ValueTypeFinder valueTypeFinder;
    private final CodeDecryptor decryptor;
    private final ProductCodeService productService;
    private final ScanInfoService scanInfoService;
    private final ProductCodeMapper productCodeMapper;

    public String proccess (MultipartFile file) throws
InvalidImageFormatException {
        String decryptedResult = decryptor.decryptCode(file);
        ValueType valueType =
valueTypeFinder.analyzeFileType(file);

        String[] parts = decryptedResult.split(": ", 2);
        if (parts.length < 2)
            throw new RuntimeException("Invalid decrypted result
format");
        String codeValue;
        String codeType = parts[0];
        codeValue = parts[1];
        if(valueType == ValueType.JSON)
            codeValue = parseJson(codeValue);
        ProductCodeDto productCode = new ProductCodeDto();
        productCode.setCodeType(codeType);
        productCode.setCodeValue(codeValue);
        productCode.setFileName(file.getOriginalFilename());
        ProductCodeDto saved = productService.save(productCode);

        ScanInfoDto scanInfoDto = new ScanInfoDto();
        scanInfoDto.setValueType(valueType);
        scanInfoDto.setScanDateTime(LocalDateTime.now());
```

Продовження лістинга А4

```
scanInfoDto.setProductCode(productCodeMapper.toProductCodeEntity(sav
ed));
          scanInfoDto.setSuccess(true);
          scanInfoService.save(scanInfoDto);
          return productCode.getCodeValue();
      }

      private String  parseJson(String jsonString) {
          ObjectMapper objectMapper = new ObjectMapper();
          try {
              JsonNode jsonNode = objectMapper.readTree(jsonString);

              String json = objectMapper.treeToValue(jsonNode,
String.class);

              return json;
          } catch (Exception e) {
              e.printStackTrace();
              return null;
          }
      }
 }
```

Лістинг А4 визначає клас для обробки файлів із штрих-кодами.

Лістинг А5 – Код файлу ImageFormatValidator.java

```
@Service
public class ImageFormatValidator {
    private final List<String> validFormats = Arrays.asList("JPEG",
"PNG", "GIF");

    private String recognizeFormat(MultipartFile file) {
        try {
            ByteArrayInputStream bis = new
ByteArrayInputStream(file.getBytes());
            BufferedImage image = ImageIO.read(bis);
            return getImageFormat(image);
        } catch (IOException | EmptyImageException |
InvalidImageFormatException e) {
            e.printStackTrace();
        return "unknown image";}
    private String getImageFormat(BufferedImage image) throws
EmptyImageException, InvalidImageFormatException {
        if (image == null)
            throw new EmptyImageException();
        String formatName =
ImageIO.getImageReadersByMIMEType("image/jpeg").next().getOriginatin
gProvider().getFormatNames()[0];
```

Продовження лістинга А5

```
if (!formatName.equals("JPEG") && !formatName.equals("PNG")
&& !formatName.equals("GIF"))
            throw new InvalidImageFormatException(formatName);
        return formatName;
    }
    public boolean isValidImage(MultipartFile file) {
        return
validFormats.contains(recognizeFormat(file).toUpperCase());
```

Лістинг А5 визначає сервіс для перевірки формату зображень.

Лістинг А6 – Код файлу MainExceptionHandler.java

```
@RestControllerAdvice
public class MainExceptionHandler {
    @ExceptionHandler(value = {Exception.class})
    public ResponseEntity<Map<String, List<String>>>
anotherException(Exception ex) {
        return getErorMap(ex);
    }

    private ResponseEntity<Map<String, List<String>>>
getErorMap(Exception ex) {
        Map<String, List<String>> map = new HashMap<>();
        map.put("errors",
Collections.singletonList(ex.getMessage()));
        return new ResponseEntity<>(map, HttpStatus.BAD_REQUEST);
    }
}
```

Лістинг А6 визначає глобальний обробник виключень, який використовує методи для обробки виключень, формування відповіді з повідомленням про помилку.

ДОДАТОК Б
Диск із матеріалами кваліфікаційної роботи