МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія (кафедра) комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

МАГАЗИН ДОДАТКІВ ВІД КОНЦЕПЦІЇ ДО РЕАЛІЗАЦІЇ

Виконав: студент групи 1П-21
Спеціальності
121 Інженерія програмного забезпечення
Віталій ЦЬОМА
Керівник:
Станіслав МАРЧЕНКО

Черкаси 2025

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

Кафедра комп'ютерної інженерії та інформаційних технологій

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри КІ та ІТ

Владислав ХОТУНОВ

_____

(підпис)

«_____»_____2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Цьомі Віталію Сергійовичу

1. Тема кваліфікаційної роботи Магазин додатків від концепції до реалізації

Керівник роботи Марченко Станіслав Віталійович, спеціаліст І категорії

затверджені наказом закладу вищої освіти від «07» жовтня 2024 року № 68У.

2. Строк подання студентом кваліфікаційної роботи 02.06.2025

3. Вихідні дані до кваліфікаційної роботи: мова програмування C#, набір інструментів .NET Aspire, СКБД MongoDB, реверсивний проксі YARP, мова опису діаграм UML, бібліотека React.js, бібліотеки xUnit та TestContainers, інструменти k6 та Playwright

4. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити): Аналіз предметної області (операційні основи онлайн-магазинів, бізнес-процеси функціонування магазинів додатків, огляд можливостей наявних магазинів додатків, постановка задачі на розробку), проєктування та реалізація програмного забезпечення (аналіз вимог до програмного забезпечення, проєктування архітектури програмного забезпечення, реалізація користувацького інтерфейсу), тестування програмного продукту (вибір методів тестування, формування тест-плану, організація комп'ютерних експериментів).

5. Дата видачі завдання 16.09.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання з підписами керівника і студента
1	Вступ	14.10.2024	
2	Розділ 1. Аналіз предметної області	09.12.2024	
3	Розділ 2. Проєктування та реалізація програмного забезпечення	10.03.2025	
4	Розділ 3. Тестування програмного продукту	28.05.2025	
5	Висновки	12.05.2025	
6	Оформлення випускної роботи (чистовий варіант)	26.05.2025	
7	Перевірка випускної роботи на наявність ознак плагіату (за 10 днів до захисту)	02.06.2025	
8	Подання випускної роботи на затвердження завідувачу кафедри (за 7 днів до захисту)	10.06.2025	

Студент _____ Віталій ЦЬОМА

(підпис)

Керівник роботи _____ Станіслав МАРЧЕНКО

(підпис)

АНОТАЦІЯ

Кваліфікаційна робота присвячена аналізу, проєктуванню та розробленню сервісу для дистрибуції програмного забезпечення – магазину додатків, реалізованого за допомогою сучасних технологій веброзробки. У межах роботи створено програмне рішення – API, що забезпечує функціонування повноцінного магазину, включаючи можливості публікації, пошуку, завантаження та оцінювання додатків.

У роботі було проаналізовано сучасний стан ринку платформ цифрової дистрибуції, таких як Google Play та Microsoft Store, а також визначено характерні технічні особливості їх реалізації.

Визначено технічні вимоги до архітектури системи, розроблено її мікросервісну структуру та реалізовано серверну частину, що включає компоненти автентифікації, управління користувачами, рецензіями, рейтингами та збереженням метаданих додатків. Система підтримує рольову модель з розмежуванням прав доступу для розробників, адміністраторів та кінцевих користувачів. Особливу увагу приділено проєктуванню інтерфейсів взаємодії між модулями та захисту API через токенну авторизацію.

У процесі реалізації було використано сучасний стек технологій: мову програмування C#, інструментарій .NET Aspire, базу даних MongoDB. Проведено тестування працездатності основних компонентів, включаючи перевірку логіки авторизації, завантаження додатків та генерації відгуків.

Результати роботи можуть бути використані як основа для створення комерційної платформи розповсюдження додатків або як технологічний фундамент для стартапів, що працюють у сфері цифрової дистрибуції. Запропоноване рішення є гнучким, масштабованим і відповідає сучасним вимогам до систем електронної комерції.

Ключові слова: API, МАГАЗИН ДОДАТКІВ, АРХІТЕКТУРА, РОЗРОБЛЕННЯ

ABSTRACT

The qualification work is devoted to the analysis, design, and development of a software distribution service – an app store implemented using modern web development technologies. As part of the work, a software solution was created – an API that ensures the functioning of a full-fledged store, including the ability to publish, search, download, and evaluate applications.

The paper analyzes the current state of the market for digital distribution platforms, such as Google Play and Microsoft Store, and identifies the characteristic technical features of their implementation.

The technical requirements for the system architecture were defined, its microservice structure was developed, and the server part was implemented, including components for authentication, user management, reviews, ratings, and application metadata storage. The system supports a role-based model with differentiated access rights for developers, administrators, and end users. Particular attention was paid to the design of interfaces between modules and API security through token authorization.

In the process of implementation, we used a modern technology stack: C# programming language, .NET Aspire toolkit, and MongoDB database. The main components were tested, including checking the logic of authorization, application loading, and feedback generation.

The results of this work can be used as a basis for creating a commercial application distribution platform or as a technological foundation for startups working in the field of digital distribution. The proposed solution is flexible, scalable, and meets modern requirements for e-commerce systems.

Keywords: API, APP STORE, ARCHITECTURE, DEVELOPMENT

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ

B2B	Business to Business
B2C	Business to Consumer
C2C	Consumer to Consumer
B2B2C	Business to Business to Consumer
DUNS	Data Universal Numbering System
USD	US Dollar
RSS	Rich Site Summary
DRM	Digital Rights Management
API	Application Programming Interface
SDK	Software Development Kit
ANR	Application Not Responding
CI/CD	Continuous Integration Continuous Delivery
СКБД	Система Управління Базою Даних
CRUD	Create Read Update Delete operations

ВСТУП

У сучасному цифровому суспільстві, де мобільні пристрої та настільні операційні системи стали невіддільною частиною повсякденного життя, магазини додатків посідають центральне місце у процесі поширення, оновлення та монетизації програмного забезпечення. Ці платформи, що спочатку виникли як зручний інтерфейс для інсталяції додатків, згодом трансформувалися у складні екосистеми з власними економічними, технічними та регуляторними правилами, забезпечуючи багаторівневу взаємодію між розробниками, користувачами, провайдерами платіжних сервісів та адміністраторами платформ. За даними дослідницької компанії Statista [1], станом на 2024 рік у двох найбільших на ринку мобільних платформ – Google Play Store та Apple App Store – сумарна кількість опублікованих додатків перевищувала 4,3 мільярда одиниць. Такий обсяг прямо свідчить не лише про масштаб світового споживчого попиту, а й про рівень довіри до централізованих цифрових інфраструктур, що здійснюють регуляцію, модерацію та підтримку життєвого циклу цих продуктів.

Водночас, попри величезну роль великих екосистем, останніми роками спостерігається пожвавлення інтересу до незалежних або альтернативних магазинів додатків, які могли б запропонувати більш гнучкі, відкриті або етичні моделі взаємодії з розробниками. Проте реалізація таких платформ стикається з низкою об’єктивних труднощів: по-перше, відсутність уніфікованих підходів до архітектури магазинів ускладнює адаптацію або повторне використання вже апробованих рішень; по-друге, складність у підтримці кросплатформної сумісності, особливо у випадку настільних середовищ, призводить до появи фрагментованих, ізольованих рішень із обмеженою функціональністю; по-третє, через відсутність відкритої документації чи формалізованих бізнес-процесів, нові ініціативи змушені відтворювати логіку роботи комерційних платформ інтуїтивно, без чіткого розуміння внутрішніх механізмів їхньої взаємодії, що неминуче призводить до технічного боргу або архітектурної неузгодженості.

Враховуючи ці обставини, очевидним стає запит на науково-практичне осмислення фундаментальних принципів, що лежать в основі побудови магазинів додатків як інженерних систем. Потреба в критичному аналізі наявних практик, їх формалізації та адаптації для відкритих чи незалежних рішень набуває дедалі більшої актуальності у контексті розвитку вільного програмного забезпечення, цифрового суверенітету та етичної комерціалізації ПЗ. Зважаючи на вищезазначене, дослідження, присвячене аналізу архітектурних, процесних та бізнесових аспектів функціонування сучасних цифрових платформ розповсюдження додатків, становить не лише академічний, але й практичний інтерес.

Об’єктом дослідження виступають процеси розповсюдження та управління програмними додатками у цифрових магазинах.

Предметом дослідження є інженерні підходи, архітектурні рішення та бізнес-процеси, що лежать в основі побудови та функціонування магазинів додатків, а також методи їх адаптації для створення незалежної крос-платформної системи.

Метою дослідження є узагальнення інженерних підходів та бізнес-процесів шляхом їх аналізу серед наявних платформ та подальша імплементація прототипу магазину додатків на основі результатів досліджень.

Часткові результати кваліфікаційної роботи були апробовані на XVII Студентській науково-практичній конференції студентів, аспірантів та молодих вчених «Тенденції розвитку ІТ-технологій в Україні» [2].

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Операційні основи роботи онлайн-магазинів

Магазин додатків – це програмна система, яка поєднує в собі функції сервісу онлайн-дистрибуції контенту та e-commerce платформи. Таким чином, розгляд магазину додатків як окремого випадку онлайн-магазину дозволить отримати краще уявлення щодо основ його роботи.

Термін «e-commerce» не має загальноприйнятого визначення, яке б охоплювало весь спектр від базової інфраструктури до роздрібних транзакцій. Більш широкі визначення включають у себе все мережеве обладнання, програмне забезпечення та електронні транзакції в основі комерції, в той час як жорсткіші визначення фокусуються тільки на купівлі та продажу товарів і послуг онлайн [3].

Концептуально, електронна комерція – це «ведення бізнес-процесів у мережі Інтернет», таких як продаж та купівля продуктів, проведення віртуальних платежів, управління запасами та ланцюгами постачання [4]. Практично, електронна комерція розглядається як цифровий канал продажів, який може працювати у різних моделях: B2C (продажі від бізнесу до споживача на платформі роздрібної продажі), B2B (оптові продажі від одного до іншого бізнесу), C2C (платформи «споживач-споживачу») або B2B2C (платформа постачання). Одна зі спільних однак поданих бізнес-моделей – створення багатосторонніх платформ для взаємодії різних груп користувачів: клієнти, продавці, платіжні провайдери, логістичні перевізники тощо [5].

На рисунку 1.1 подана BPMN-діаграма, на якій описано основні бізнес-процеси типової e-commerce платформи онлайн-магазину.

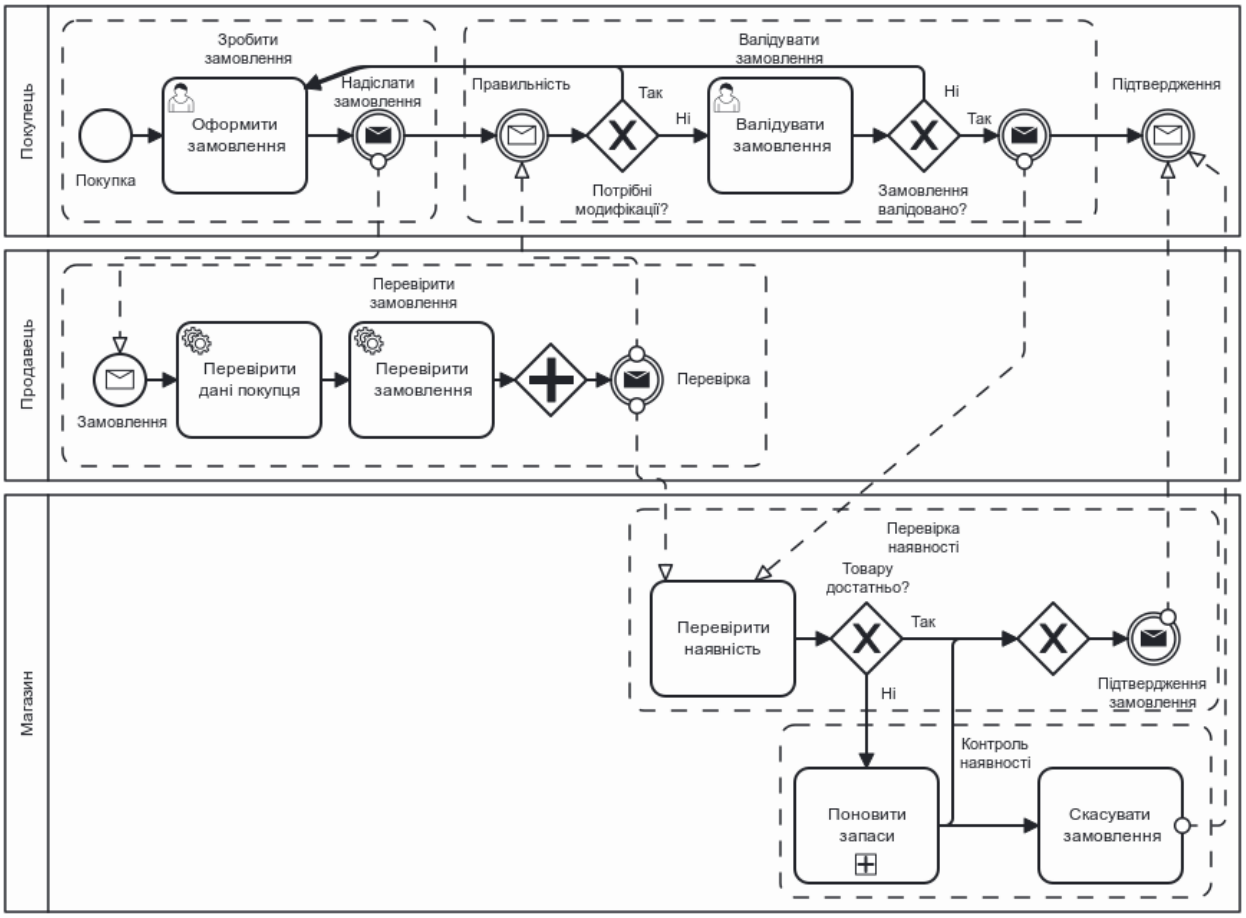

Рисунок 1.1 – Основні бізнес-процеси онлайн-магазину [3]

Перший процес зі сторони покупця – «Зробити замовлення». Його виконання починається з рішення покупця придбати товар. Зразу після цього він ініціює оформлення замовлення – досліджує каталог магазину, формує замовлення шляхом додавання потрібних товарів у список покупок – «кошик» та зміни їх кількості, після цього користувач заповнює деталі щодо доставки та оплати. У результаті, повідомлення зі сформованим замовленням відправляється в систему продавця.

Вхідна подія наступного процесу «Валідувати замовлення» перевіряє чи відповідає подане замовлення критеріям повноти та синтаксичної коректності. За умови виявлення помилок чи неточностей, таких як відсутність адреси доставки чи неправильний ідентифікатор товарної позиції, потік спрямовує покупця до попереднього процесу з ціллю виправлення замовлення. Як тільки замовлення помічається як коректне, система надає покупцю можливість ручної

перевірки замовлення, зокрема таких аспектів як вартість самого замовлення чи доставки та, при наявності, акційних знижок.

Подальше розгалуження «Замовлення валідовано?» перевіряє чи задоволений покупець станом замовлення. У випадку негативної відповіді, користувач спрямовується на початок загального процесу з метою уточнення деталей замовлення. Процес створює повідомлення про підтвердження замовлення тільки після позитивної відповіді від користувача. Це повідомлення виконує дві ролі: воно повідомляє продавця про остаточність замовлення та одночасно запускає підпроцес перевірки замовлення магазином. Це є останньою активною дією покупця в процесі.

Роль продавця відкриває процес «Перевірити замовлення», який очікує отримання повідомлення про замовлення від покупця. Його отримання активізує два типово автоматизовані сервісні завдання: «Перевірити дані покупця» та «Перевірити замовлення». Перевірка даних покупця може аутентифікувати особу користувача, зробити перехресне посилання на кредитний ліміт або перевірити наявність авторизації для проведення платежу. Далі, перевірка власне замовлення містить у собі перехресне звіряння наданих товарних ідентифікаторів, кількості товарів та їх цін щодо каталогової бази даних продавця для запевнення коректності кожної обраної товарної позиції та правильності застосування бізнес-правил щодо формування цін чи знижок.

Після проходження верифікації одночасно ініціюються декілька процесів. Один із них надає користувачу можливість перевірки замовлення, про яку було сказано вище, інший надсилається в магазин з метою перевірки наявності товарів. Вказано можливість додаткових процесів, наприклад запобігання шахрайству, підготовка чеку чи внесення правок у облікову базу даних. Після запуску цих двох процесів відповідальність продавця за відправлене замовлення закінчується, і смуга продавця закінчується до надходження нових вхідних повідомлень.

Смуга магазину починається не звичайною подією запуску, а надходженням підтверджувального повідомлення від продавця. Це активує

згорнутий підпроцес «Перевірка наявності», в якому користувацьке завдання перевіряє поточний рівень запасів на відповідність запитуваній покупцем кількості. Після цього ексклюзивний шлюз оцінює, чи достатньо запасів. Якщо наявні запаси можуть задовольнити замовлення в повному обсязі, потік переходить безпосередньо до кінцевої події «Підтвердження замовлення», яка сповіщає покупця та продавця про відправку товару. На цьому роль магазину у цій транзакції завершується.

Однак, якщо шлюз визначає що запасів недостатньо, діаграма спрямовує потік до вторинного підпроцесу, який називається «Контроль наявності». У межах цього підпроцесу сервісна задача «Поновлення запасів» видає внутрішні директиви – можливо, формує замовлення на закупівлю для постачальників вищого рівня або ініціює переміщення товару з іншої точки. Паралельна гілка цього підпроцесу – «Скасувати замовлення» – залишається доступною, якщо поновлення запасів виявилося неможливим або якщо покупець відмовляється від замовлення.

Зрештою, виникає один з двох результатів: або магазин відновлює достатній запас і повертається назад, щоб підтвердити замовлення (об’єднуючись у шлях «Достатній запас»), або він остаточно скасовує замовлення та інформує покупця і продавця про скасування за допомогою вихідного повідомлення. У будь-якому випадку, шлях магазину досягає свого кінця після відправлення останнього повідомлення.

Через те, що електронна комерція існує виключно у цифровому просторі, концепції класичної комерції переносяться на неї по-іншому. Наприклад, ланцюжок створення вартості Майкла Портера можна інтерпретувати по-новому: вхідна логістика (пошук продуктів), операції (програмне забезпечення платформи), вихідна логістика (доставка), маркетинг/продажі (онлайн-маркетинг і персоналізація) і послуги (підтримка клієнтів) – все це реалізується за допомогою цифрових систем [7].

Електронна комерція знижує транзакційні та інформаційні витрати завдяки глобальному охопленню та детальним даним щодо клієнтів. Однак вона також

створює нові проблеми довіри та безпеки: покупці та продавці, які можуть ніколи не зустрітися особисто, покладаються на механізми довіри (рейтинги, гарантії, шифрування та правила). Звідси, операції електронної комерції ґрунтуються на технічній архітектурі, яка забезпечує стабільне обслуговування, та на бізнес-процесах, які керують потоками інформації, товарів і грошей у розподіленій мережі.

1.2. Бізнес-процеси функціонування магазинів додатків

Для аналізу було обрано наступні магазини додатків: Google Play Store, Apple App Store та Microsoft Store. Ці магазини пропонують додатки для платформ Android, iOS та Windows відповідно. Також вибір пояснюється тим, що ці магазини оперуються розробниками відповідних платформ, тобто можуть вважатись офіційними та такими, що дотримуються найкращих практик щодо дистрибуції програмного забезпечення для обраної платформи.

Процес публікації додатку в каталозі Google Play починається зі створення профілю розробника на платформі Play Console. Наявні дві опції: створити персональний акаунт чи акаунт організації. Якщо вибрано другий тип акаунту, організацію потрібно верифікувати за допомогою ідентифікатора DUNS. Далі надається форма, в полях якої потрібно зазначити базову інформацію, таку як назву розробника та контактну адресу електронної пошти, прив’язати платіжний профіль, пройти опитування про досвід створення додатків та наміри розробника. Останній крок – надання згоди з умовами платформи та одноразова оплата розміром 25 USD. Після цього, використовуючи Play Console, розробник може розміщувати додатки в каталозі [8]. Для цього потрібно створити додаток, дати йому назву та категорію. Потім – надати більш детальну інформацію та заповнити його сторінку. Після створення додатку, розробник може завантажити бінарний файл, який у подальшому буде розглянуто на предмет відповідності політикам контенту. Останньою вимогою є бета-тестування додатку, у якому повинно взяти участь як мінімум 25 тестувальників з окремими обліковими

записами Google. За умови успішного виконання вищезазначених умов, додаток стає доступний на широкий загал. Використовуючи додаток Google Play Store, кінцевий користувач може переглядати каталог, купляти та встановлювати додатки [9].

Для публікації додатку у Apple App Store, розробнику потрібно зареєструвати свій обліковий запис Apple у Apple Developer Program [10]. Процес є майже ідентичним до такого в Google Play Store, за винятком деяких аспектів, зокрема оплати – у App Store вона є щорічною та становить 99 USD, більшої кількості правил та, як результат, більш суворої перевірки [11]. Так, додаток для платформ Apple повинен правильно функціонувати на всіх таргетованих пристроях, незалежно від їх характеристик.

Процес реєстрації облікового запису розробника у Microsoft Store є схожим до такого у попередніх магазинах додатків. Одноразова оплата становить 19 USD. Процес публікації додатку дещо відрізняється від типового – першим кроком є резервування унікального ID додатку, після чого розробник подає бінарний файл разом із заповненою формою з базовими даними (назва додатку, ціна) та наповненням його сторінки (опис, скриншоти). Подана заява проходить процес модерації та сертифікації, після завершення якого додаток стає доступним для покупки та завантаження кінцевим користувачам [12].

Для узагальнення бізнес-процесів створення акаунту та публікації додатків у магазинах додатків було побудовано BPMN-діаграму на рисунку 1.2. В основі узагальненого процесу лежить тріада зацікавлених сторін, а саме незалежний розробник, оператор магазину та кінцевий користувач. На діаграмі ці учасники розподілені на три площини, а весь процес сегментовано на декілька етапів: залучення розробника, сплата внеску, подання заяви на розміщення додатку, модерація контенту та подальша його публікація і придбання та встановлення користувачем.

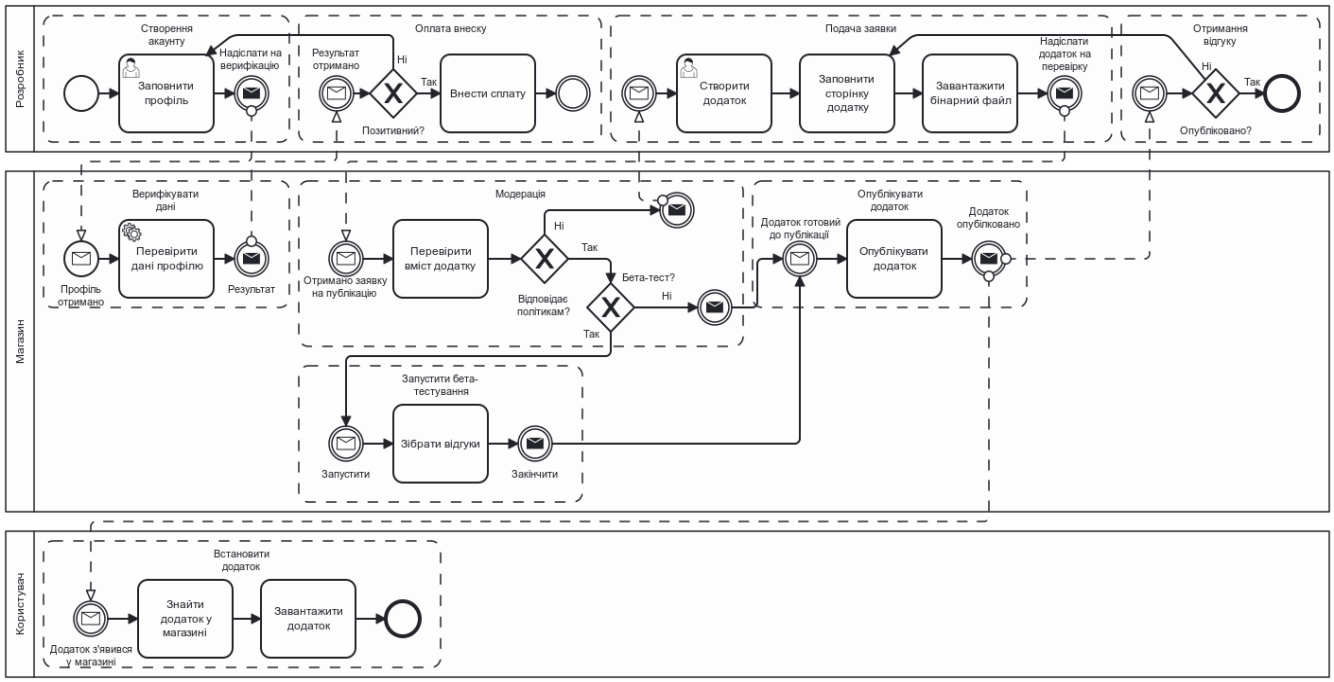

Рисунок 1.2 – BPMN-діаграма узагальненого процесу

Бізнес-процес починається з рішення розробника взяти участь у цифровому ринку, яке позначено стартовим маркером. Це негайно переносить розробника в підпроцес «Створення облікового запису», він заповнює форму зі своїми ідентифікаційними даними (юридична назва, організаційний статус, код DUNS, контактні дані, банківські реквізити тощо). Заповнена форма надсилається до команди модераторів магазину, це може бути реалізовано безліччю способів, наприклад через виклик API або електронний лист, але діаграма абстрагується від реалізації. Розробник очікує на результат перевірки. При негативній відповіді, розробник дізнається про невідповідність наданих відповідей та повинен повернутись до попереднього етапу. Натомість, позитивний результат відкриває розробнику шлях до наступного етапу – сплати реєстраційного внеску.

Подолавши бар’єр ідентифікації, розробник переходить до підпроцесу «Оплата внеску». Таке завдання несе важливе стратегічне значення – акт переказу реєстраційного внеску сигналізує про зобов’язання розробника, фінансує операційні витрати магазину та дозволяє відсіяти спам-запити. Успішне завершення цього платіжного завдання надсилає розробнику

повідомлення про підтвердження платежу – квитанція або токен платежу. Після виконання фінансових зобов'язань обліковий запис розробника набуває повного статусу видавця.

Далі розробник переходить до підпроцесу «Подача заяви». Першим кроком є створення метаданих, що відображається у завданні «Заповнити сторінку додатка», де розробник надає магазину все що побачить кінцевий користувач: назву додатка, текстовий опис, теги категоризації, банери та, можливо, журнал змін. Одночасно з цим розробник повинен надати власне скомпільований виконуваний файл за допомогою завдання «Завантажити двійковий код». Хоча на діаграмі це розглядається як єдина, атомарна дія, реальність може включати перевірку контрольної суми, перевірку обмеження розміру і профілювання сумісності.

Зібравши метадані та двійковий код, розробник ініціює повідомлення «Надіслати додаток на перевірку». Ця подія, надіслана до магазину, означає передачу контролю: відтепер команда модераторів магазину визначатимуть, чи відповідає ця заява вимогам політики та якості. Надалі розробник очікує вердикту від магазину – або відхилення та список недоліків чи проблем або зелене світло для публікації. Ексклюзивний шлюз діаграми – «Опубліковано?» – інкапсулює цей момент. Якщо заяву було відхилено, то розробника перенаправляють назад до підпроцесу «Подати заяву».

Процеси магазину активуються при отриманні повідомлень паралельно процесам розробника. Подія запуску повідомлення «Отримано профіль» запускає підпроцес «Перевірка даних профілю». У ньому завдання «Перевірка даних профілю» запускає автоматизовану або ручну перевірку на відповідність юридичним, фінансовим і контентним політикам сховищ. Після завершення перевірки «Результат» надсилається назад розробнику, замикаючи цикл, встановлений на етапі приєднання до проєкту.

Коли Магазин отримує заяву розробника, окрема подія запуску повідомлення ініціює підпроцес «Перевірка вмісту додатку». Він містить два послідовні сервісні завдання: «Перевірка метаданих», мета якого переконатися,

що текстові описи є точними, не вводять в оману і не містять заборонених посилань (наприклад, захищеного авторським правом контенту або мови ворожнечі) та «Перевірка двійкових файлів», яке піддає виконуваний файл технічній перевірці: скануванню на наявність шкідливих програм, профілюванню продуктивності, тестам сумісності на рівні API чи випробуванням розгортання в пісочниці.

Після цих перевірок ексклюзивний шлюз «Чи відповідає політикам?» оцінює, чи повністю заява відповідає правилам магазину. Якщо ні, розробнику надсилається повідомлення «Потребує змін», і процес призупиняється, доки не надійде нова заява. Якщо вимоги було задовільнені, робочий процес стикається з другим шлюзом - «Потрібне бета-тестування?», який визначає, чи потрібно проводити тестування спільнотою.

При необхідності бета-тестування магазин переходить до підпроцесу «Бета-тестування», надсилаючи збірку програми групі тестувальників («Запустити бета-версію»), збираючи структуровані відгуки («Зібрати відгуки») та звітуючи про результати («Завершити бета-тестування»).

Після того, як всі перешкоди щодо якості та відповідності вимогам усунуто, магазин переходить до підпроцесу «Публікація додатку». Це передбачає агрегування остаточних метаданих і двійкових файлів у публічному каталозі («Додаток готовий до публікації») і виконання процедури розгортання, яка робить додаток доступним для пошуку, категоризації та завантаження («Публікація додатку»). Кульмінацією процесу є повідомлення «Додаток опубліковано», яке надсилається розробнику, можливо, через RSS-стрічку або цільове сповіщення на електронну пошту.

1.3 Огляд можливостей наявних магазинів додатків

Задля опису магазинів додатків у роботі «What is an App Store? The Software Engineering Perspective» [13] використовується такий набір

високорівневих категорій: опції монетизації, наявність DRM, потреба в акаунті, залежність додатків від інших продуктів, цільова аудиторія, націленість розробника, ціль магазину додатків, посередницька роль магазину, композиційність продуктів, аналітика, канали комунікації.

Опції монетизації описують опції для проведення платіжних операцій, які надаються користувачу безпосередньо магазином додатків. Вся вибірка магазинів додатків (Google Play Store, Apple App Store, Microsoft Store) підтримує опції одноразової покупки додатка, покупок всередині додатка та підписок за часом, не надаючи можливості монетизації за допомогою підписок за кількістю інсталяцій та використаними ресурсами. Ці опції більш характерні для підприємницького контексту.

Кожен із магазинів у вибірці підтримує DRM, але в той час як Google Play та App Store примусово захищають кожен додаток за його допомогою, Microsoft Store лише надає інструменти, перекладаючи відповідальність за забезпечення безпеки додатка на розробника.

Потреба в обліковому записі – ще один важливий фактор при огляді. Вся вибірка не потребує акаунту для перегляду каталогу. Більшість магазинів вимагають створити акаунт для завантаження будь-яких додатків, але Microsoft Store дозволяє анонімним користувачам завантажувати безкоштовні додатки. Проте для подальшої взаємодії з платформою (наприклад, щоб залишити відгук чи купити платний додаток) наявність облікового запису обов’язкова.

Щодо залежності додатків, правила Apple App Store не дозволяють додаткам взаємодіяти один з одним, тобто кожен додаток повинен бути незалежним [10]. У той самий час, інші магазини не мають такого обмеження, але вимоги до додатків-плагінів варіюються між різними магазинами. Google Play дає можливість створення додатків без функціональності, але з якимись даними, наприклад ключем для розблокування повної версії іншого додатка, хоча й такий підхід вважається застарілим та небажаним. Microsoft Store вимагає щоб функціональність додатка-плагіна постачалась саме в ньому.

Цільова аудиторія магазинів додатків поділяється на загальну та тематичну, як приклад останньої – Adobe Magento, який фокусується на створенні e-commerce-платформ. Усі магазини додатків у вибірці спрямовані на загальну аудиторію.

Націленість розробника описує типовий фокус розробників, які публікують додатки в магазин. За цим фактором розділяють два типи розробників: розробники, які переслідують комерційні чи бізнес-цілі, та розробники з фокусом на спільноту, наприклад розробники вільного та відкритого програмного забезпечення.

Мета магазину додатків визначає його основні високорівневі завдання та наміри. Значення, отримані з власних описів цілей магазинів додатків, часто можна знайти на вебсторінках «Про нас». Ціль Google Play – бути офіційним та довіреним джерелом додатків і контенту для пристроїв під керуванням ОС Android, підтримуючи успіх розробників та безпеку користувачів за допомогою масштабованої комерційної екосистеми. App Store має на меті стати висококонтрольованим і безпечним цифровим ринком, який сприяє довірі, інноваціям та дизайну, орієнтованому на користувача в екосистемі Apple. Мета Microsoft Store – запропонувати універсальну та інклюзивну платформу для розповсюдження програм і контенту для Windows, що поєднує доступність для користувачів із гнучкістю для підприємств і свободою розробників.

Посередницька роль описує ролі, які магазин додатків відіграє між користувачами та розробниками. Це – сервіси, пов’язані з інженерією програмного забезпечення, які є зазвичай незалежними один від одного. Наприклад, перевірки під час виконання відстежують чи забезпечує магазин додатків коректну роботу своїх продуктів.

Google надає Google Play Publishing API для автоматизації потоку публікації. За допомогою цього API та таких плагінів як Gradle Play Publisher, розробники можуть програмно відвантажувати файли APK/App Bundle з новими версіями додатку, призначати їх до різних релізних каналів та оновлювати лістинги додатків у магазинах. Android-проєкти часто використовують Android

Studio в поєднанні з конвеєрами на Gradle. Сервіси для безперервної інтеграції (Jenkins, Github Actions, Gitlab CI тощо) взаємодіють з програмним інтерфейсом Publishing API для публікації збірок, для тестування у конвеєр може бути впроваджено Firebase Test Lab – хмарний сервіс для тестування додатків на великій кількості конфігурацій пристроїв [14, 15].

Для моніторингу під час виконання Google Play впроваджує Android Vitals – набір моніторингових інструментів, який агрегує дані на пристроях користувачів (кількість збоїв, ANR, використання батереї тощо) для кожного додатка [16]. Коли користувачі надають згоду на збір даних, їхні пристрої передають ці показники назад до Google, розробники переглядають їх у Play Console або отримують через Google Play Developer Reporting API. Такі показники як кількість збоїв та ANR безпосередньо впливають на видимість додатку в магазині. Крім того, розробники можуть інтегрувати Firebase Crashlytics для звітування про збої в режимі реального часу [17]. Також Google Play надає змогу запустити автоматизовані тести на пререлізних збірках з метою знаходження вразливостей, збоїв та проблем з відображенням. Це відбувається на базі Firebase Test Lab.

При випуску оновлення, розробники можуть виконати поетапне розгортання, вибравши відсоток користувачів, які отримають нову версію. Це забезпечує поступове розгортання [18]. У будь-який момент розгортання може бути зупинено: нові користувачі не отримають оновлення, при цьому ті, хто вже оновився, залишаться на ньому. Пізніше розгортання може бути відновлено або розширено. Якщо виявлено проблему, розробники можуть зупинити і швидко опублікувати нову збірку. Google Play також пропонує функцію «Instant app» і внутрішній тестовий трек, але відкат означає зупинку розгортання або непублікацію версії. Звідси, Google Play Console забезпечує гнучкий контроль над оновленнями, щоб зменшити кількість неякісних випусків.

Екосистема Android має тісну інтеграцію між Android Studio, Android SDK та сервісами Google Play. Бібліотеки білінгу та ліцензування Google Play легко додаються через Gradle. Плагіни Play Console та Firebase для Android Studio

спрощують розгортання та тестування. Конвеєри CI/CD часто використовують Google Cloud Build або Firebase Test Lab для автоматизованої збірки і тестування.

Для створення CI/CD-конвеєрів Apple пропонує App Store Connect API та Xcode Cloud. App Store Connect API дозволяє розробникам автоматизувати майже всі завдання App Store Connect – створення версій додатків, управління поетапними випусками, налаштування TestFlight і відправку на рецензування [19]. Xcode Cloud – це вбудований в Xcode сервіс CI/CD, який автоматично збирає додатки, запускає паралельні тести і може розгортати збірки в TestFlight і App Store. На практиці розробники iOS зазвичай використовують Xcode (з Xcode Cloud або без нього) та інструменти для написання сценаріїв, такі як Fastlane (який використовує App Store Connect API) для реалізації конвеєрів CI/CD [20].

App Store Connect також надає дані про продуктивність та діагностику. Розробники можуть завантажувати інформацію про енергоспоживання та продуктивність додатка (час запуску, використання пам’яті тощо) зі збірок, зібраних Apple. App Analytics в App Store Connect показує статистику використання та залучення. Доступ до зібраних журналів збоїв можна отримати через Xcode Organizer.

Для бета-тестування використовується сервіс TestFlight [21]. Збірки, що розповсюджуються зовнішнім тестувальникам через TestFlight, повинні пройти огляд бета-версії програми Apple перед розповсюдженням. Процес тестування додатків для iOS, як правило, складається з тестування додатків самими розробниками з подальшим рецензуванням від App Store; власне Apple не надає загальнодоступного автоматизованого набору тестів. Однак розробники можуть використовувати XCTest і користуватися симуляторами, які надаються Xcode або послугами CI для перевірки функціональності.

Інструменти для розробників від Apple базуються на Xcode. Він підтримує безпосереднє архівування додатків та їхнє завантаження до Apple Store Connect. Включено симулятори та інструменти для профілювання. Для CI доступні Xcode Server та новіший Xcode Cloud. Також середовище Xcode може запускати збірки

TestFlight за допомогою викликів API. Розробники також застосовують сторонні інструменти автоматизації, що дозволяють спростити процес завантаження збірок до App Store, взаємодіючи з API Apple Store Connect. Загалом, система інстументів Apple дуже добре інтегрована, процес публікації вбудовано в середовище програмування та пов’язані з ним сервіси.

Компанія Apple дозволяє поетапні випуски оновлень для додатків. Розробники можуть обрати опцію «Випустити протягом 7-денного періоду», щоб все більший відсоток користувачів отримував оновлення щодня. Цей процес можна призупинити або відновити в будь-який час протягом 7-денного періоду. Якщо з’являється термінова проблема, розробник може призупинити поетапне розгортання або відключити автоматичні оновлення [22]. На відміну від поетапного розгортання Google, поетапний реліз Apple триває строго 7 днів і не обмежує аудиторію: будь-який користувач може завантажити оновлення вручну в будь-який час. Компанія Apple не пропонує прямого «відкочування до попередньої версії», скоріше, розробник зазвичай вилучає додаток з продажу та повторно публікує останню версію або представляє виправлену нову версію.

Microsoft Store підтримує CI/CD переважно через зовнішні системи збірки та Partner Center API. Розробники, які створюють програми для Windows (UWP, Win32), використовують Visual Studio або інші конвеєри збірки (Azure DevOps, GitHub Actions) для компіляції та пакування програм. API заявок Microsoft Store (Dev Center API) автоматизує процес публікації для MSI/EXE або UWP додатків [23].

Центр партнерів Microsoft Store надає показники стану програм за допомогою звіту про стан – «Health report» [24]. Сюди входять дані про збої та зависання для випущених програм, розробники можуть переглядати діаграми збоїв за вибраний період часу. Для глибшого аналізу доступне трасування стека, розробники мають змогу використовувати REST API для програмного отримання цих даних. Раніше Visual Studio App Center надавав аналітику та звіти про збої для UWP/Win32 додатків, але сьогодні більшість розробників Windows покладаються на звіти з Центру для партнерів та SDK для телеметрії.

Для розробки під ОС Windows використовується Visual Studio та Windows SDK [25]. Visual Studio може пакувати програми як пакунки MSIX або UWP. До складу SDK входить Windows App Certification Kit (WACK) для перевірки додатків перед надсиланням заяви. Системи CI (Azure Pipelines, GitHub Actions або Azure DevOps) можуть використовувати інструменти командного рядка центру партнерів або REST API для створення нових релізів. До свого закриття, Visual Studio App Center пропонував плагіни збірки та тестування додатків, але на сьогодні Microsoft не має універсального вбудованого рішення, схожого на Xcode Cloud, хоча надає інструменти для інтеграції автоматичного розгортання в магазин.

Перевірка додатків на відповідність вимогам продуктивності, пакування та безпеки відбувається за допомогою набору інструментів WACK [26]. Цей комплект може бути запущений на ПК розробника або інтегрований у CI. Після подання заяви, Microsoft виконує власну автоматизовану сертифікацію (безпека, технічна відповідність, перевірка політики). Якщо будь-який тест не пройдено, заяву відхиляють. Через це розробники часто покладаються на запуск WACK локально (або через Azure DevOps) для виявлення проблем. У Microsoft Store існує офіційний інструмента для бета-тестування, подібний до TestFlight [27]. Додаток може розгортатись тільки для обмеженого коли користувачі або бути схованим у каталозі, в такому випадку користувачі можуть отримати до нього доступ за допомогою промокодів та посилань.

Microsoft Store має вбудований механізм відсоткового розгортання, принцип роботи якого схожий до такого в Google Play [28].

1.4 Постановка задачі на розробку

Одним із найбільш суттєвих недоліків сучасних магазинів додатків є їхня закритість як з технічного, так і з організаційного погляду. Архітектури таких систем, як правило, не є публічно доступними, що ускладнює сторонній аналіз, інтеграцію з іншими платформами, а також обмежує можливості розширення або

адаптації до альтернативних сценаріїв використання. Відсутність єдиного стандарту побудови магазинів додатків призводить до надмірної фрагментації ринку, ускладнює міжплатформену взаємодію та створює додаткові бар'єри для розробників і користувачів, особливо в контексті альтернативних або незалежних операційних середовищ.

У зв'язку з цим, у межах кваліфікаційної роботи здійснюється спроба розробки відкритої, модульної та максимально гнучкої архітектури магазину додатків, яка б не залежала від конкретної операційної системи чи пропрієтарного середовища виконання. Проєктована система ґрунтується на принципах інкапсуляції функціональності, завдяки чому кожен окремий компонент реалізує чітко визначені функції й може бути незалежно протестований, оновлений або замінений без впливу на інші частини системи. Такий підхід забезпечує не лише прозорість та масштабованість, але й дозволяє адаптувати рішення до широкого кола сценаріїв дистрибуції додатків, від традиційних ПК до мобільних пристроїв, віртуальних середовищ і навіть IoT-систем.

Для реалізації демонстраційного варіанту платформи, що ґрунтується на запропонованій відкритій архітектурі, обрано сучасний стек технологій, орієнтований на високу модульність, масштабованість та підтримку міжплатформенності. Серверна частина системи буде реалізована мовою програмування C# із використанням інструментарію .NET Aspire, який забезпечує підтримку хмарної інфраструктури, сервісної орієнтації та розширеного моніторингу на етапі розробки. В якості системи управління базами даних застосовується MongoDB – документоорієнтована NoSQL СКБД, що добре масштабується та забезпечує гнучке зберігання даних.

Архітектура бекенду побудована відповідно до мікросервісного підходу, де кожен компонент реалізований як незалежний REST API-додаток на основі ASP.NET Core. Така структура дозволяє розподіляти навантаження, ізолювати збої та забезпечує можливість незалежного оновлення окремих сервісів без впливу на всю систему в цілому.

Фронтенд адміністративної частини магазину буде реалізовано з використанням JavaScript-фреймворку React, який забезпечує високу динамічність інтерфейсу та полегшує реалізацію реактивної логіки. Такий розподіл технологій дозволяє досягти оптимального балансу між продуктивністю, зручністю розробки та можливістю подальшого розширення платформи.

Висновки до першого розділу

У першому розділі було оглянуто та проаналізовано основні аспекти платформи e-commerce, особливу увагу приділено бізнес-процесам інтернет-магазинів. Також було проаналізовано документації магазинів додатків Google Play Store, Apple App Store та Microsoft Store. На основі цього змодельовано бізнес-процеси функціонування магазину додатків.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз вимог до програмного забезпечення

Система має підтримувати базову інфраструктуру дистрибуції додатків: каталог, пошук, перегляд, публікацію, завантаження, оновлення та систему відгуків. Розробники створюють власні облікові записи розробників, потім заповнюють заяви, які в подальшому розглядають адміністратори. Якщо заява відповідає правилам та політикам щодо контенту, додаток додається в каталог, у якому його можуть знайти та встановити кінцеві користувачі.

Використовуючи аналіз наявних магазинів додатків, який було проведено в першому розділі, можемо виділити три типи користувачів: кінцеві користувачі, розробники та адміністратори. Загальні функціональні вимоги до системи наведено в таблиці 2.1.

Таблиця 2.1 – Функціональні вимоги до системи

Категорія	Функціональна вимога
Реєстрація та вхід	Користувач може зареєструватись або увійти через email та пароль
Перегляд каталогу	Користувач бачить список додатків, може шукати додатки за запитом та застосовувати фільтри
Сторінка додатка	Повинна включати опис, скріншоти, рейтинг та відгуки та кнопку завантаження
Публікація додатка	Розробник може створити сторінку додатка та завантажити бінарний файл
Система відгуків	Авторизовані користувачі можуть залишати коментарі та оцінки
Завантаження	Користувач може завантажити обраний додаток
Адміністративна панель	Адміністратор може переглядати заяви щодо публікації, схвалювати або відхиляти їх

Розроблення бекенду ведеться з використанням мови C#, інструментарію .NET Aspire та технологій ASP.NET Core REST API, MongoDB, YARP у ролі

реверсивного проксі. Нефункціональні вимоги до API зібрано в таблиці 2.2. Набір технологічних вимог до магазину додатків описано у таблиці 2.3.

Таблиця 2.2 – Нефункціональні вимоги до системи

Категорія	Функціональна вимога
Продуктивність	Час відповіді сервера не повинен перевищувати 1.5 секунд
Масштабованість	Система повинна працювати без збоїв при 100+ запитів у секунду
Безпека	Всі дані передаються через HTTPS, автентифікація за допомогою JWT
Надійність	Система повинна працювати без збоїв при 100+ запитів у секунду
UX	Сайт повинен бути інтуїтивно зрозумілим для користувачів без технічних знань

Таблиця 2.3 – Технологічні вимоги до системи

Назва	Опис
Архітектура REST API	Програмний інтерфейс системи повинен бути реалізований у стилі REST API
Контейнеризація	Усі мікросервіси повинні бути упаковані у Docker-контейнери для спрощення розгортання та масштабування
CI/CD	Система має підтримувати автоматичне тестування та деплой через CI/CD-контейнери
Масштабованість	Архітектура має підтримувати горизонтальне масштабування за допомогою оркестратора
Файлова система	Завантажені додатки повинні зберігатися або в файловій системі, або у хмарному сховищі

2.2 Попередня архітектура програмного продукту

API виступає посередником між представленням програми та її бізнес-логікою, яка виконується на сервері. Для взаємодії з REST API використовуються різні HTTP-методи, які описують мету запиту [29]. Натомість, зі сторони сервера передаються дані у форматі JSON, який часто використовується для репрезентації даних у frontend-частині системи.

Власне REST API побудований мовою програмування C# за допомогою інструментарію .NET Aspire. Кожен мікросервіс є окремим ASP.NET Core додатком, який виконує операції в межах свого домену. Важливою частиною архітектури є YARP – реверсивний проксі, який надає можливість викликати різні мікросервіси з однієї точки входу [30], а також виконати авторизацію чи аутентифікацію вхідного запиту перед його передачею до цільового мікросервісу [31]. При наявності, вхідні дані серіалізуються у класи, що дає змогу гнучкої роботи з ними. Аутентифікація відбувається за допомогою JWT-токену, який надається користувачу після успішної авторизації. Запити до захищених ресурсів повинні включати токен у заголовку Authorization [32].

З метою точнішого відображення структури та взаємодій у системі, що проєктується, було створено серію діаграм у відповідності до методології C4. Зокрема, на рисунку 2.1 представлено діаграму контекстів, основним завданням якої є формування загального уявлення про зовнішнє середовище, в якому функціонує система, та про ключових акторів, які з нею взаємодіють [33]. Такий рівень абстракції дозволяє побачити систему як «чорну скриньку», зосередившись на її інтерфейсах взаємодії з користувачами або іншими інформаційними системами, залишаючи поза увагою її внутрішню реалізацію. На цій діаграмі можна спостерігати трьох акторів, які взаємодіють із вебзастосунком різними способами – через SPA React-додаток та за допомогою API. Як результат взаємодії, акторами переслідуються різні цілі.

Наступний рівень деталізації у методології C4 – діаграма контейнерів, яка розкриває загальну архітектуру системи на рівні окремих програмних компонентів, здатних до самостійного розгортання та виконання. У контексті системи «Магазин додатків», зображеній на попередньому рисунку, діаграма контейнерів дає змогу деталізувати внутрішню структуру системи, представляючи кожну її функціональну підсистему в вигляді логічної одиниці, що може бути незалежно реалізована, розгорнута, масштабована або замінена [34]. У цьому випадку контейнерами виступають окремі мікросервіси, що відповідають за різні аспекти роботи системи (наприклад, обробку заявок,

управління користувачами, антивірусне сканування тощо), а також модулі доступу до бази даних (DAO), які здійснюють безпосередню взаємодію з документоорієнтованою СКБД MongoDB.

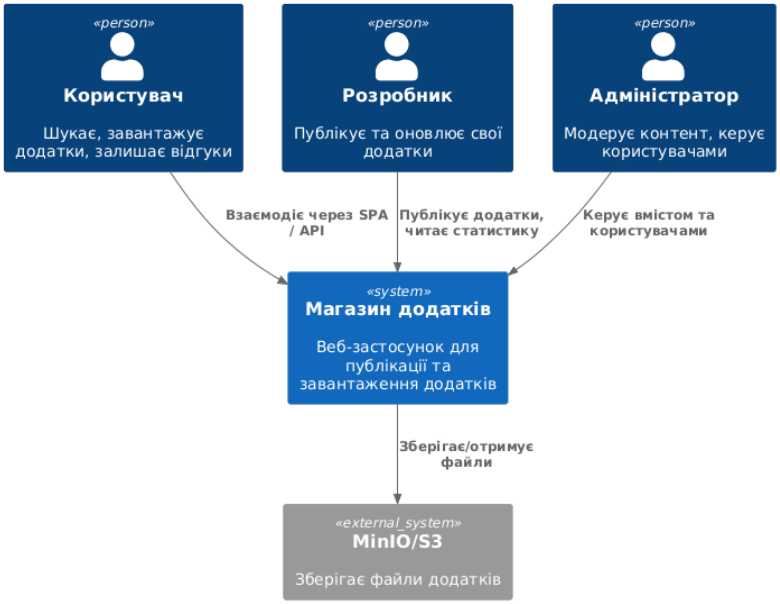

Рисунок 2.1 – Діаграма контекстів

Діаграма такого типу зображена на рисунку 2.2. На більш детальному представленні можна бачити не тільки краще подану взаємодію акторів із системою, а й взаємодію індивідуальних мікросервісів з базою даних, сховищем файлів та один з одним. У цілому, перших двох діаграм зазвичай достатньо для представлення попередньої архітектури програмного проєкту на практиці.

2.3 Деталізоване проєктування програмного забезпечення

Для глибшого розуміння принципів функціонування програмної системи на рисунку 2.3 зображено UML-діаграму пакетів, яка деталізує внутрішню структуру одного з контейнерів, представлених на діаграмі контейнерів. Цей рівень деталізації дозволяє проаналізувати логічну організацію окремого мікросервісу або іншого програмного контейнера, розкриваючи його внутрішні компоненти, такі як контролери, сервіси, репозиторії, допоміжні модулі тощо.

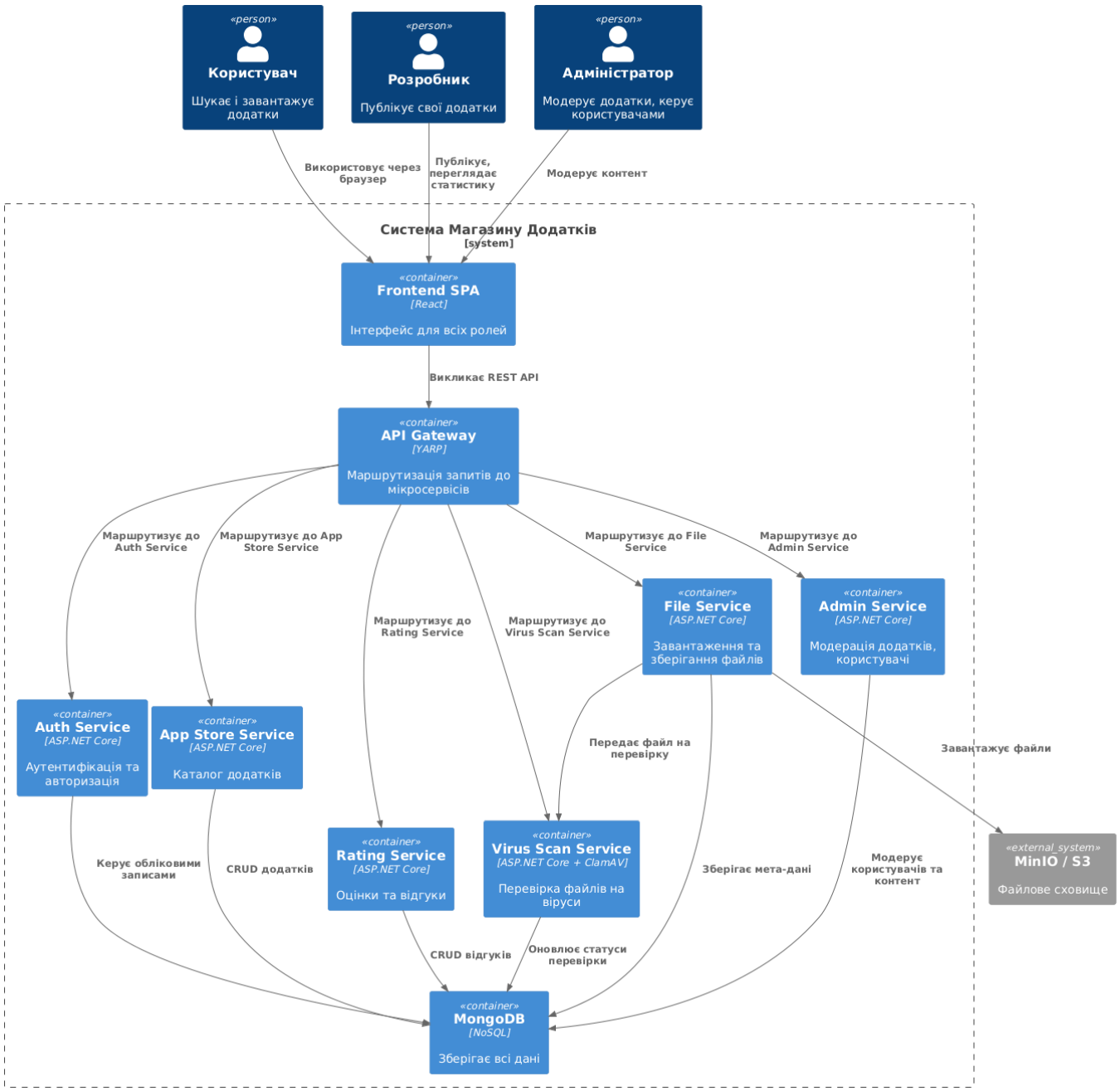

Рисунок 2.2 – Діаграма контейнерів

Діаграма пакетів на рисунку 2.3 демонструє, яким чином функціональність контейнера розподілена між цими частинами, як саме відбувається передача управління та даних між ними, а також які зовнішні залежності (наприклад, бібліотеки, сторонні сервіси чи API) беруть участь у його роботі.

Пакет API Gateway відіграє роль єдиного вхідного пункту для зовнішніх запитів. Він спрямовує HTTP-запити до відповідних контролерів у межах пакетів App Store Service, Admin Service, Auth Service, Rating Service і File Service. Така

централізація забезпечує єдиний рівень аутентифікації, логування та маршрутизації запитів.

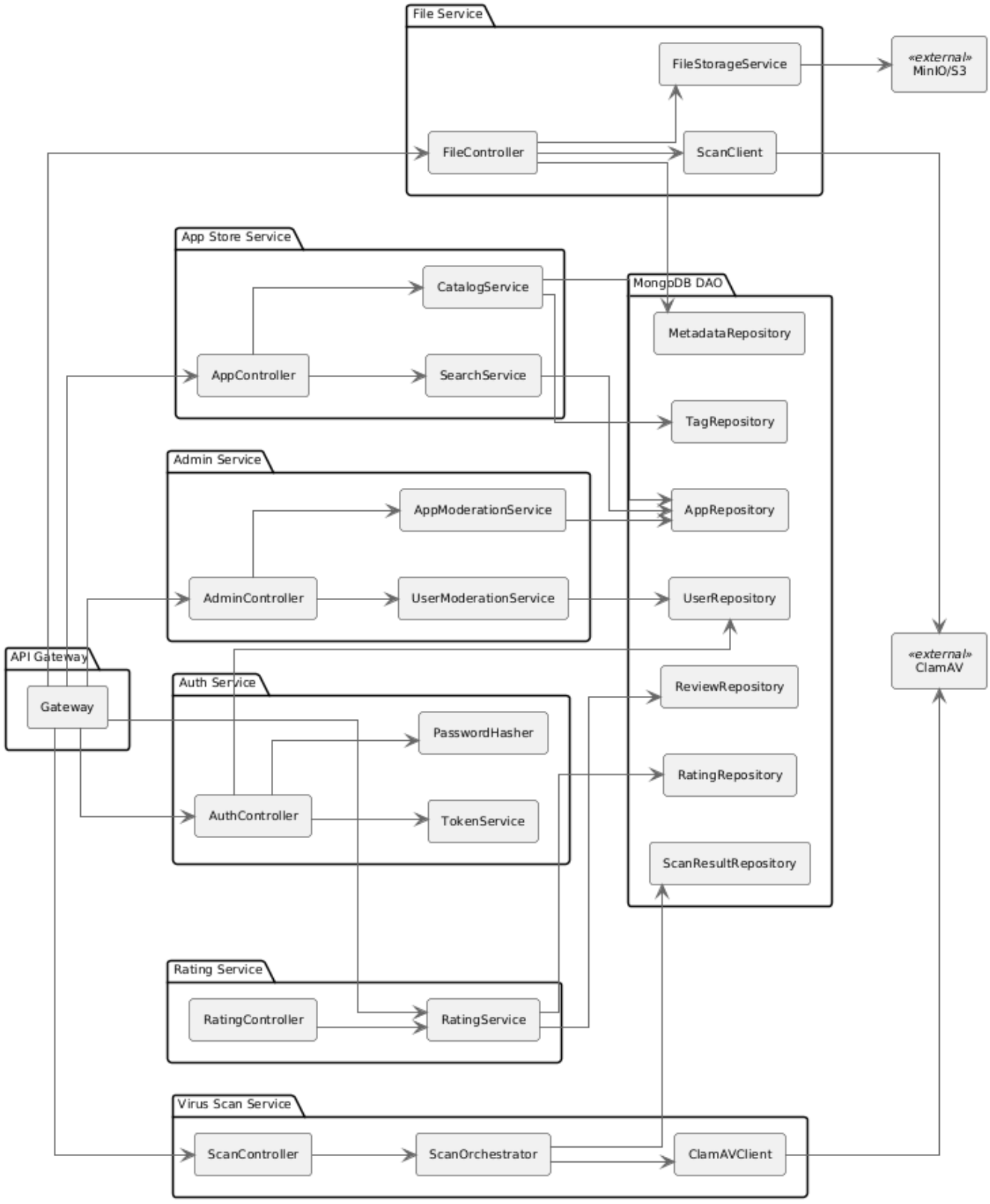

Рисунок 2.3 – Діаграма пакетів

Кожен контролер ініціює виклик внутрішніх сервісів, наприклад, AppController звертається до CatalogService і SearchService для пошуку та каталогізації додатків, RatingController викликає RatingService для керування рейтингами, а FileController у пакеті File Service взаємодіє із ScanClient та FileStorageService для обробки файлів. Сервіси працюють із сховищем даних через набір репозиторіїв у пакеті MongoDB DAO.

Кожен пакет чітко інкапсулює свою відповідальність, виклики до бази даних здійснюються централізовано через рівень DAO, а зовнішні інтеграції сконцентровані в окремих сервісах, що спрощує тестування й заміну компонентів.

Більш докладно внутрішню програмну реалізацію підсистем описують діаграми класів у вигляді класів, їхніх атрибутів, методів та взаємозв’язків. Цей рівень застосовується для демонстрації об’єктно-орієнтованої структури коду та зазвичай одночасно використовується як найнижчий рівень у C4-моделі. Діаграма класів, зображена на рисунку 2.4, репрезентує повну реалізацію системи поширення програмного забезпечення.

Визначена діаграма охоплює ключові аспекти доменної моделі, інфраструктурного шару, сервісного шару та рівня контролерів. Структура побудована за принципами інкапсуляції, інтерфейсної абстракції та впровадження залежностей.

Доменна сутність App містить властивості, що описують ідентифікатори, назву, версію, мінімальну версію API, архітектури, розмір, категорію, опис, кількість завантажень, ідентифікатор видавця та середній рейтинг, що дозволяє зберігати всю необхідну інформацію для демонстрації й пошуку додатків. Publisher, який також входить до доменних сутностей, зберігає дані про видавців: унікальні ідентифікатори, контактну інформацію, назву та опис, забезпечуючи відображення відомостей про автора програмного продукту. Review і Rating моделюють відгуки користувачів і оцінки відповідно: Review містить текст відгуку, посилання на додаток і користувача, а також часову мітку створення; Rating зберігає оцінку у вигляді цілого числа з прив’язкою до додатка та

користувача. ScanResult реєструє результати перевірки файлів на шкідливий код, фіксуючи ідентифікатор файлу, булевий прапорець чистоти, деталі перевірки та часову мітку.

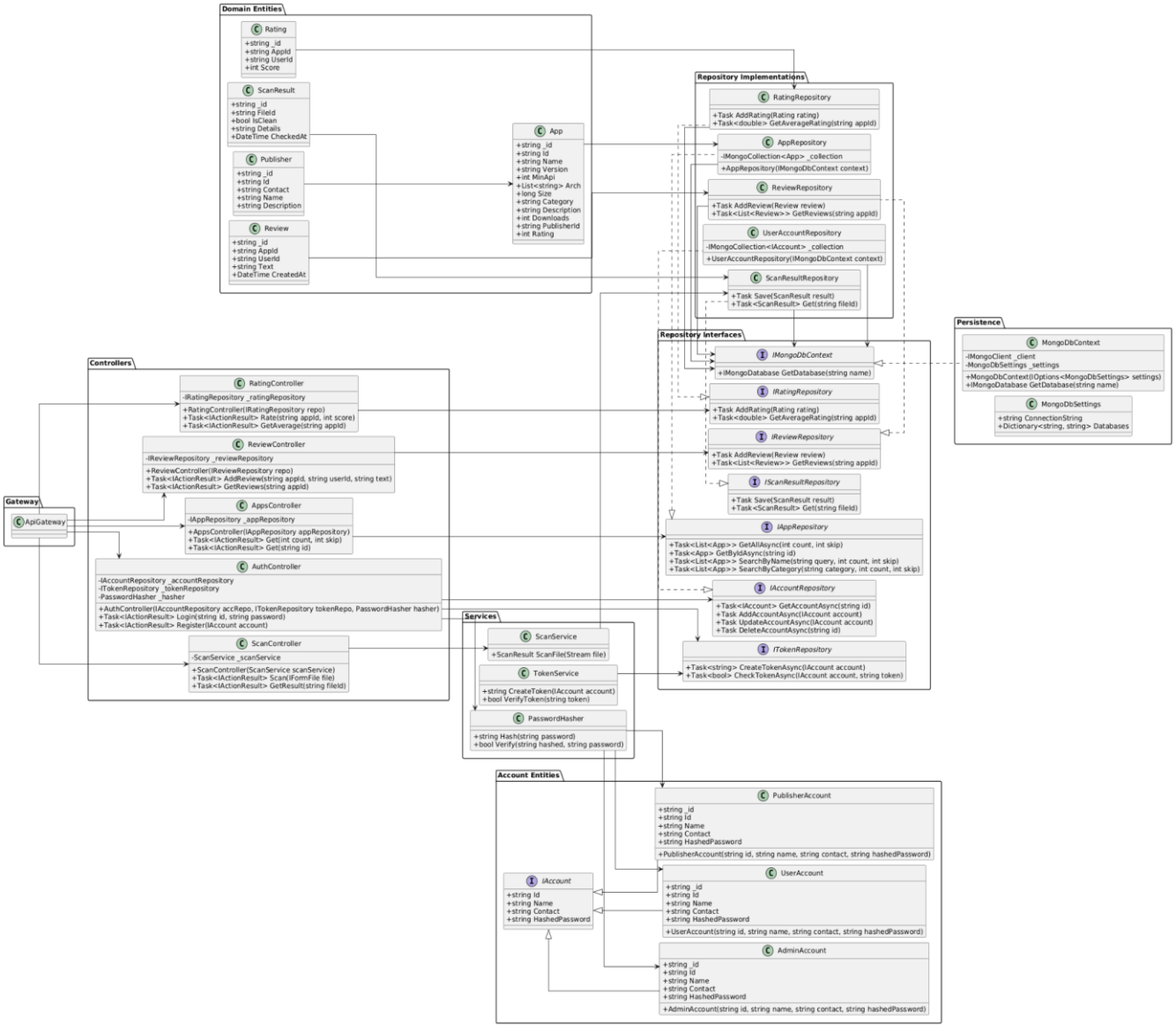

Рисунок 2.4 - Діаграма класів

Реалізація облікових записів здійснюється через інтерфейс IAccount, який задає базові властивості Id, Name, Contact і HashedPassword. Його наслідують класи UserAccount, PublisherAccount та AdminAccount, що реалізують відповідні конструктори для ініціалізації ідентифікаторів, імен, контактів та захешованих паролів. Інтерфейси репозиторіїв визначають CRUD-операції для доменних сутностей: від отримання й пошуку App через IAppRepository до керування

обліковими записами в IAccountRepository, генерації токенів в ITokenRepository, збереження відгуків та рейтингів в IReviewRepository і IRatingRepository, а також збереження та отримання результатів антивірусного сканування через IScanResultRepository. IMongoDbContext описує забезпечення доступу до бази даних MongoDB.

Класи AppRepository, UserAccountRepository, ReviewRepository, RatingRepository та ScanResultRepository реалізують відповідні інтерфейси та інжектують IMongoDbContext для роботи з колекціями MongoDB. Сервісний шар утворюють TokenService (генерує та перевіряє токени), PasswordHasher (виконує хешування та верифікацію паролів для всіх типів облікових записів) та ScanService (перевіряє файли та повертає об’єкт ScanResult). У Persistence реалізовано MongoDbSettings і MongoDbContext для конфігурації з’єднання з базою та отримання екземпляра IMongoDatabase.

Контролери AppsController, AuthController, RatingController, ReviewController і ScanController інжектують відповідні репозиторії та сервіси, забезпечуючи REST-ендпоінти для отримання списків і подробиць додатків, авторизації й реєстрації користувачів, додавання рейтингу, відгуків та запуску антивірусного сканування. Клас ApiGateway виступає єдиною точкою входу, маршрутизуючи HTTP-запити до відповідних контролерів та координуючи взаємодію усіх компонентів системи.

Загалом, архітектура була спроєктована з оглядом на модульність, масштабування та експлуатацію у реальних умовах.

2.4 Реалізація користувацького інтерфейсу

Для реалізації користувацького інтерфейсу було використано бібліотеку React.js, маршрутизація на сторінці відбувається за допомогою React Router. Користувацький інтерфейс запаковано у мікросервіс, який оркеструється інструментами Aspire разом з іншими частинами системи. Код для маршрутизації подано на лістингу 2.1.

Лістинг 2.1. Код для маршрутизації

```
<Routes>
        <Route path="/" element={<HomePage />} />
        <Route path="/apps/:id" element={<AppDetailsPage />} />
        <Route path="/about" element={<AboutPage />} />
        <Route path="/login" element={<LoginPage />} />
      <Route path="*" element={<h1 style={{ padding: '2rem' }}>Page not found</h1>} />
</Routes>
```

На рисунку 2.5 зображено зовнішній вигляд головної сторінки магазину.

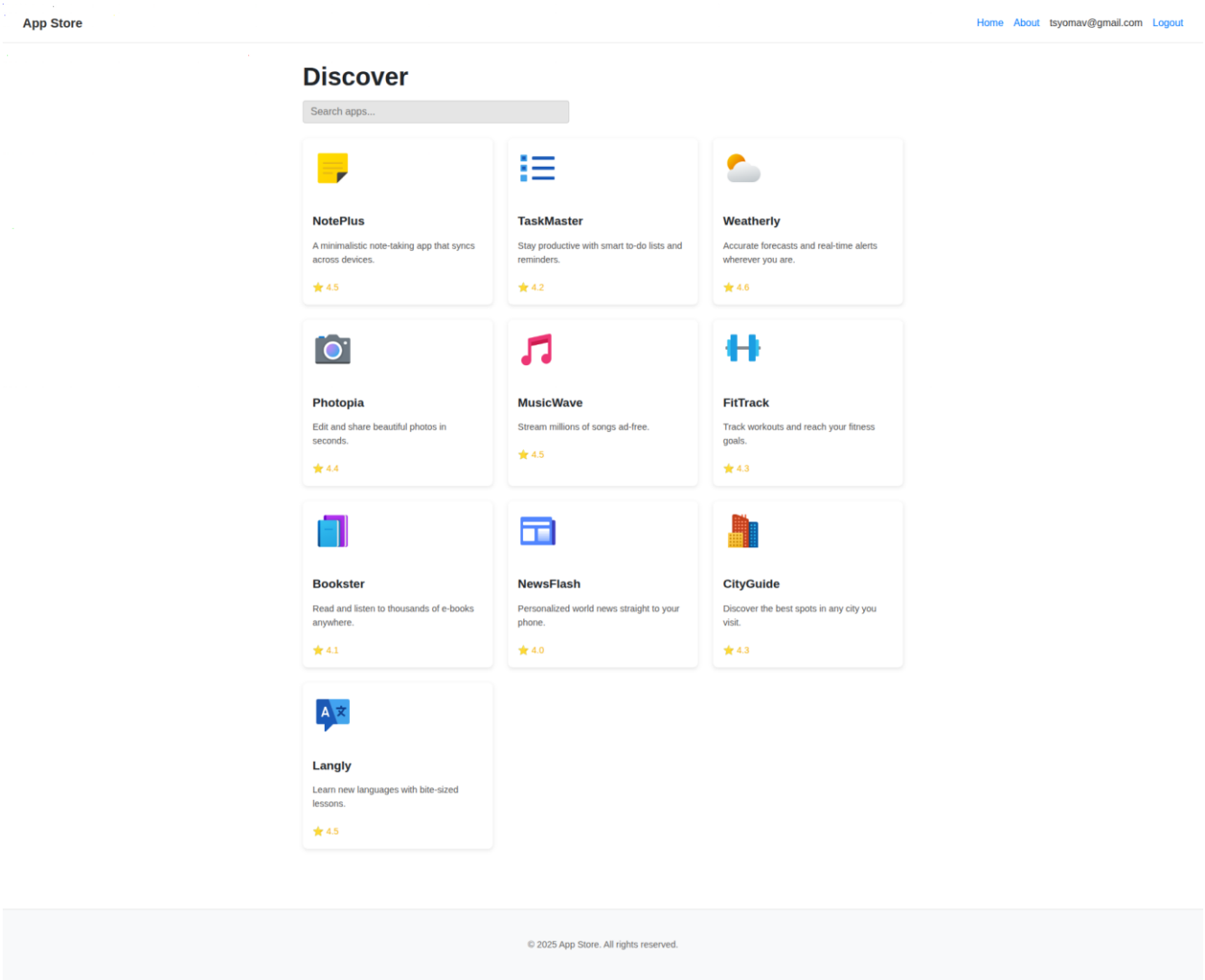

Рисунок 2.5 – Зовнішній вигляд головної сторінки магазину додатків

Користувач має можливість скористатися функціональністю пошуку з метою знаходження релевантних програмних продуктів у каталозі застосунків. Цю функціональність реалізовано у вигляді серверної логіки, що дозволяє ефективно обробляти запити незалежно від клієнтської платформи. Демонстрація функції пошуку подана на рисунку 2.6.

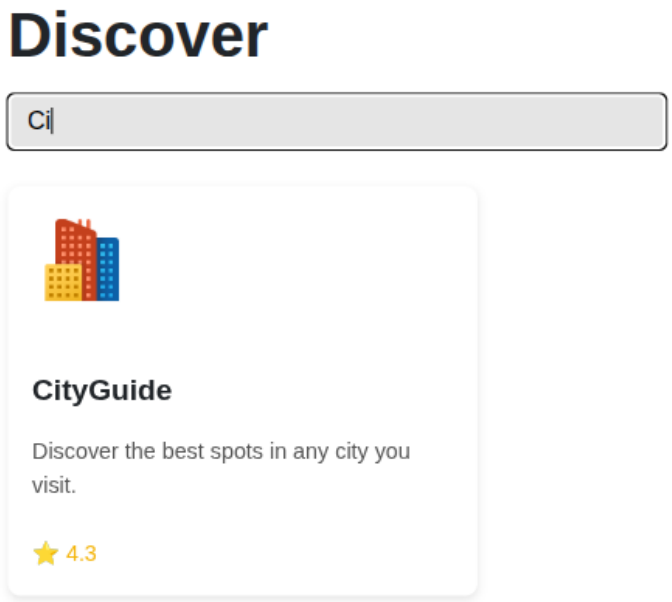

Рисунок 2.6 – Демонстрація функціональності пошуку

Кожний додаток має власну детальну сторінку представлення, яка генерується динамічно на основі інформації, отриманої з бази даних. Ця сторінка агрегує ключові характеристики застосунку для кінцевого користувача. Серед основних її елементів – назва та видавець додатку, середній рейтинг, мультимедійні матеріали (зображення інтерфейсу, функціональності або брендування), стислий опис, список основних функцій, розділ нових змін, а також блок користувацьких оцінок і коментарів. Компонент взаємодії з користувачем включає можливість залишити власний відгук. Наявні відгуки, отримані через API, відображаються в зручному для аналізу вигляді з візуальною діаграмою розподілу оцінок.

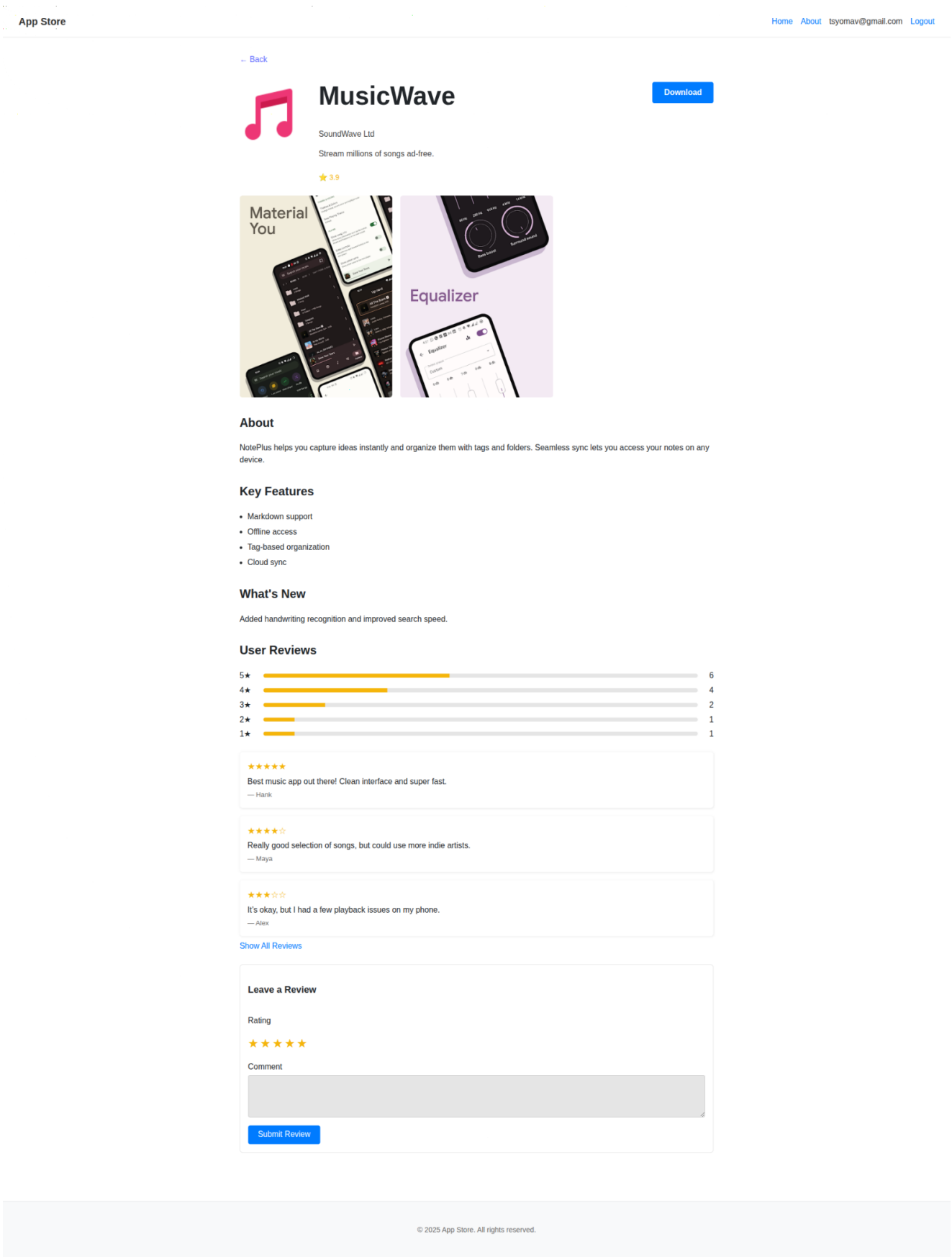

Рисунок 2.7 – Зовнішній вигляд сторінки додатка

Висновки до другого розділу

У другому розділі було проведено аналіз вимог до програмної системи магазину додатків, у результаті якого представлено список функціональних, нефункціональних та технологічних вимог. На основі визначених вимог було спроєктовано архітектуру готової системи, яку було програмно реалізовано у подальшому. Утворений прототип готовий до верифікації працездатності.

РОЗДІЛ 3
ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Вибір методів тестування

У процесі розроблення та впровадження повноцінної програмної системи магазину додатків тестування відіграє ключову роль у забезпеченні стабільності, безпеки та надійності програмного продукту. Для цієї системи доцільно застосувати кілька рівнів тестування, що охоплюють усі аспекти – від окремих одиниць коду до повноцінної взаємодії між сервісами, включаючи нефункціональні характеристики.

Основу тестування становить модульне тестування (unit testing), яке дозволяє перевірити поведінку окремих класів, зокрема сервісів, репозиторіїв та контролерів, в ізоляції. Таке тестування є критично важливим для логіки, яка не залежить від зовнішніх ресурсів, зокрема для класів PasswordHasher, TokenService, ScanService або RatingRepository. Для написання модульних тестів буде застосовано бібліотеку xUnit, яка є де-факто стандартом у середовищі .NET завдяки своїй простоті, гнучкості та глибокій інтеграції з середовищем Visual Studio. У поєднанні з пакетом Moq, популярною бібліотекою для створення мок-об’єктів, можна імітувати зовнішні залежності, такі як IMongoDbContext, не виконуючи реальних запитів до бази даних.

На наступному рівні слід застосувати інтеграційне тестування (integration testing), яке забезпечує перевірку коректної взаємодії між компонентами системи. Наприклад, важливо перевірити, як AppsController у поєднанні з AppRepository взаємодіє з базою даних MongoDB, чи виконується збереження рейтингу в RatingRepository, і чи оновлюється агрегований рейтинг. Для реалізації таких тестів використовуватиметься бібліотека Microsoft.AspNetCore.Mvc.Testing, яка дає змогу піднімати повноцінний тестовий вебсервер із використанням TestServer, забезпечуючи максимально наближені до продуктових умови. Для конфігурації ізольованої бази даних

MongoDB можна застосувати Testcontainers for .NET, яка надає можливість створювати тимчасові контейнери Docker з MongoDB для проведення незалежних і повторюваних тестів.

Функціональне тестування (functional testing) зосереджується на перевірці кінцевої поведінки системи з точки зору користувача. У цьому контексті важливо забезпечити коректність сценаріїв реєстрації користувача, входу до системи, перегляду списку додатків, публікації відгуків, додавання рейтингу, та перевірки файлів на віруси. Для автоматизації таких перевірок доцільно використати інструмент Playwright for .NET, який дозволить емулювати реального користувача в браузері. Також Playwright забезпечує кросбраузерну підтримку, можливість тестування SPA-додатків і зручну інтеграцію з CI/CD-конвеєрами.

Для кожного мікросервісу також передбачається тестування API-інтерфейсів (API testing) з акцентом на правильність обробки запитів і відповідей, перевірку кодів стану, валідацію параметрів і авторизаційні механізми. Для цього буде використано бібліотеку RestSharp у поєднанні з xUnit для програмного тестування REST-інтерфейсів.

Ще одним важливим аспектом є навантажувальне тестування (load testing), що дозволяє оцінити, як система поводиться за умов щільного трафіку, особливо сервіси AuthController, AppsController та ScanController, які очікувано будуть найчастіше використовувані. Для реалізації таких перевірок буде застосовано k6, що надає можливість моделювання тисяч одночасних запитів, генерації звітів про час відповіді, кількість помилок і стабільність системи під навантаженням.

3.2 Формування тест-плану

У межах комплексного підходу до перевірки працездатності та надійності системи магазину додатків сформовано послідовний план тестування, який охоплює всі рівні й аспекти функціонування. План орієнтовано не лише на автоматичну перевірку коду, а й на підтвердження відповідності очікуванням

користувачів і забезпечення безпеки даних. Для реалізації цього плану введено розділення процесу тестування на кілька фаз, кожна з яких передбачає використання відповідних інструментів, підходів і критеріїв завершеності.

Перша фаза – це модульне тестування. Метою є перевірка кожного окремого класу в ізоляції від решти системи. Пріоритет надається перевірці логіки класів, що відповідають за обробку даних, генерацію токенів, хешування паролів і взаємодію з інтерфейсами репозиторіїв. Тестування охоплює сервіси PasswordHasher, TokenService, ScanService, а також репозиторії AppRepository, RatingRepository та інші. Критерієм завершеності виступає охоплення основної логіки не менше ніж на 80% згідно з метриками коду.

Друга фаза – інтеграційне тестування. На цьому перевіряється, як компоненти взаємодіють між собою. Наприклад, чи коректно ReviewController працює з ReviewRepository, чи справно MongoDbContext передає з'єднання до колекцій бази даних. Тестування відбувається з використанням TestServer, Microsoft.AspNetCore.Mvc.Testing, а також Testcontainers, які дають змогу симулювати повноцінне середовище MongoDB у Docker.

Третя фаза – функціональне тестування. Вона імітує поведінку реального користувача, який взаємодіє з системою через REST API або інтерфейс. Тут використовуються сценарії типу: «користувач реєструється – входить у систему – шукає додаток – залишає відгук – додає оцінку – сканує файл». Тестування проводиться як вручну, так і автоматизовано засобами Playwright. Результати оцінюються за відповідністю очікуваній поведінці, HTTP-кодам, коректності формату відповідей.

Четверта фаза – тестування навантаження. Вона передбачає моделювання великої кількості одночасних запитів, особливо до AppsController, AuthController та ScanController. Для цього використовується інструмент k6, який дозволяє генерувати тисячі конкурентних запитів та вимірювати затримки, відсоток помилок, стабільність системи під стресом. Основним критерієм у цьому випадку є витримування системою навантаження в межах запланованих обсягів, без деградації продуктивності та втрати даних.

Повний план представлено в таблиці 3.1, способи тестування для кожного компоненту системи – в таблиці 3.2. У межах таблиці 3.3 описано основні тестові випадки. Зразки тестів TC002, TC016 подано в лістингу 3.1, а TC021 – в лістингу 3.2 відповідно. Повний проєкт розташовано на платформі GitHub за адресою, зазначеною в додатку А.

Таблиця 3.1 – План тестування системи

Етап	Ціль	Інструменти	Критерій успішності
Модульне тестування	Перевірка логіки окремих класів	xUnit, Moq	>80% покриття логіки
Інтеграційне	Перевірка взаємодії компонентів	TestServer, Testcontainers	Успішна взаємодія без помилок
Функціональне	Симуляція поведінки користувача	Playwright	Коректні відповіді на типові запити
Навантажувальне	Перевірка стабільності під щільним трафіком	k6	Система витримує >1000 RPS

Таблиця 3.2 – Способи тестування компонентів системи

Компонент	Типи тестування
PasswordHasher	Модульне
TokenService	Модульне, інтеграційне
ScanService	Модульне, функціональне, навантажувальне
AppRepository	Модульне, інтеграційне
MongoDbContext	Інтеграційне
Controllers	Інтеграційне, функціональне, регресійне
ApiGateway	Функціональне, навантажувальне
UserAccountRepository	Модульне, інтеграційне
ReviewController	Інтеграційне, функціональне
RatingRepository	Модульне, інтеграційне, функціональне

Таблиця 3.3 – Опис тестових випадків

TC ID	Опис тестового випадку	Тип	Очікувана поведінка
TC001	Хешування пароля не повертає вхідне значення	Модульне	Повертається хеш, не рівний паролю
TC002	Перевірка валідного пароля на збіг із хешем	Модульне	Verify повертає true
TC003	Токен створюється без винятків і не є null	Модульне	Створено коректний токен
TC004	Сканування «чистого» файлу повертає IsClean=true	Модульне	Повертається ScanResult з IsClean == true
TC005	Запит до AppRepository повертає не null список	Модульне	GetAllAsync() повертає список
TC006	GetById повертає правильний об’єкт	Інтеграційне	Ідентифікатори збігаються
TC007	RatingRepository зберігає і обчислює середнє значення	Інтеграційне	GetAverageRating повертає правильне число
TC008	MongoDbContext повертає валідну базу	Інтеграційне	GetDatabase() не кидає винятків
TC009	Login працює з правильним паролем	Інтеграційне	Login() повертає токен
TC010	Рейтинг можна отримати через RatingController	Інтеграційне	Відповідь 200 OK
TC011	Register додає нового користувача до бази	Інтеграційне	Користувач з'являється в репозиторії
TC012	ScanResultRepository зберігає об’єкт і дозволяє отримати його	Інтеграційне	Get(fileId) повертає збережене значення
TC013	Сценарій: Реєстрація → Вхід → Пошук → Відгук → Рейтинг	Системне	Усі кроки проходять без помилок
TC014	Авторизація за відсутнього токена повертає 401	Системне	Отримуємо Unauthorized
TC015	Неможливість залишити оцінку двічі	Системне	Повторна оцінка повертає 400
TC016	Сканування зараженого файлу повертає IsClean = false	Системне	Прапорець сканування false
TC017	Пошук за категорією повертає релевантні додатки	Системне	Category усіх додатків збігається
TC018	Некоректний пароль не дає токена	Системне	Відповідь — 401 Unauthorized
TC019	Навантаження 100 RPS протягом 60 секунд (AppsController)	Навантажувальне	< 5% помилок, стабільна відповідь

Продовження таблиці 3.3

ТС020	Навантаження на ScanController із 50 запитами/сек протягом 30 с	Навантажувальне	Немає збоїв у відповіді
ТС021	1000 запитів на авторизацію протягом 20 с	Навантажувальне	Усі запити завершуються протягом <500мс
ТС022	Стабільність при багатопоточному збереженні рейтингів	Навантажувальне	Дані зберігаються, без помилок
ТС023	Безпечна зміна пароля не впливає на інші облікові записи	Системне	Тільки цільовий акаунт змінено
ТС024	Успішна реєстрація нового користувача	Функціональне	Користувача перенаправляє на сторінку входу або підтвердження після успішної реєстрації
ТС025	Переконатися, що пошукова система правильно знаходить додаток за ключовим словом	Функціональне	У списку результатів з’являється додаток з відповідною назвою або її частиною
ТС026	Перевірити, що авторизований користувач може залишити відгук до додатку	Функціональне	Після надсилання форми з'являється новий відгук у списку

Лістинг 3.1 – Програмна реалізація тестів ТС002 та ТС016

```
[Fact]
public void PasswordHasher_Verifies_CorrectPassword()
{
    var hasher = new PasswordHasher();
    var password = "Test1234!";
    var hashed = hasher.Hash(password);
    Assert.True(hasher.Verify(hashed, password));
}


[Fact]
public async Task ScanService_Detects_MaliciousFile()
{
    var stream = File.OpenRead("infected_sample.exe");
    var service = new ScanService(mockScanner);
    var result = await service.ScanFile(stream);
    Assert.False(result.IsClean);
}
```

3.3 Організація комп’ютерних експериментів

Тестування програмного забезпечення здійснювалося відповідно до попередньо визначеного плану у контрольованому локальному середовищі.

Основна мета полягала у виявленні логічних помилок, перевірці коректності обробки запитів та оцінці стійкості системи до навантажень.

Для реалізації модульного тестування було створено окремий проєкт з використанням бібліотеки xUnit, який структуровано за принципами організації тестового середовища. Файлова структура тестового проєкту наведена на рисунку 3.1 та демонструє логічний поділ тестів за функціональними модулями системи. Видно, що для кожного модуля передбачено окремий файл або набір файлів, що значно полегшує підтримку та розширення тестової бази у подальшому.

Рисунок 3.1 – Файлова структура HexStore.Tests

Крім того, для перевірки стійкості до навантажень було реалізовано окремий скрипт, наведений у лістингу 3.2, що використовує бібліотеку k6 для проведення навантажувального тестування REST API-методу авторизації. Скрипт виконує симульовані запити від 50 віртуальних користувачів протягом 20 секунд, відправляючи дані для аутентифікації у форматі JSON. Перевірка результатів здійснюється за двома критеріями: відповідь зі статусом 200 OK та наявність у відповіді JWT-токена.

Результати проходження тестів подано на рисунках 3.2-3.5. На рисунку 3.2 зображено підсумковий результат проходження модульних тестів — усі тести було виконано успішно, що свідчить про коректність логіки на рівні окремих компонентів.

Лістинг 3.2 – Програмна реалізація тесту TC021

```
import http from 'k6/http';
import { check } from 'k6';

export let options = {
  vus: 50,
  duration: '20s',
};

export default function () {
  const payload = JSON.stringify({
    id: "user1",
    password: "correct_password"
  });

  const headers = { 'Content-Type': 'application/json' };
  let res = http.post('https://localhost:5001/api/auth/login', payload, { headers
});
  check(res, {
    'login success': (r) => r.status === 200,
    'token returned': (r) => r.json('token') !== undefined,
  });
}
```

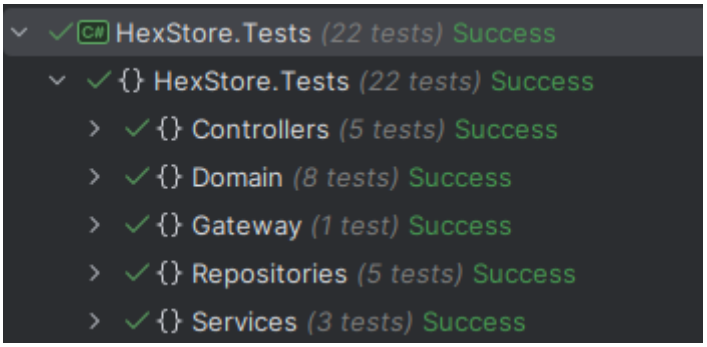

Рисунок 3.2 – Результат проходження модульних тестів

На рисунку 3.3 наведено результати інтеграційного тестування. Тут перевіряється взаємодія між окремими модулями системи, зокрема контролерами, сервісами та репозиторіями. Усі перевірки також пройдено без помилок, що підтверджує коректну інтеграцію компонентів.

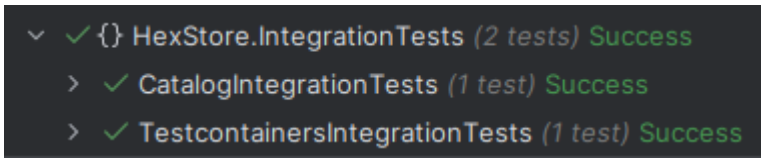

Рисунок 3.3 – Результат проходження інтеграційних тестів

Для перевірки відповідності поведінки користувацького інтерфейсу очікуванням було проведено функціональне тестування за допомогою бібліотеки Microsoft Playwright. Основною метою цього етапу було моделювання взаємодії реального користувача з вебінтерфейсом, а також перевірка правильності навігації, відображення елементів та реакції на введення даних. Результати представлені на рисунку 3.4.

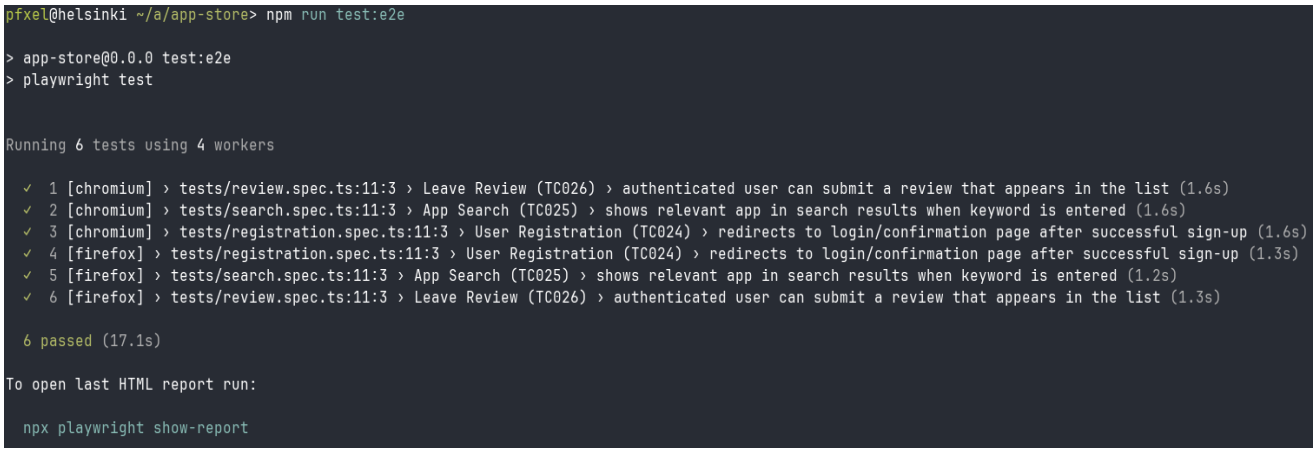

Рисунок 3.4 – Результат проходження функціонального тестування

Останній етап – навантажувальне тестування (рисунок 3.5) – дозволив оцінити продуктивність системи при високому одночасному навантаженні. Результати відповіді системи демонструють стабільну продуктивність без суттєвих затримок або помилок, що є позитивним показником масштабованості та готовності сервісу до експлуатації у реальному середовищі.

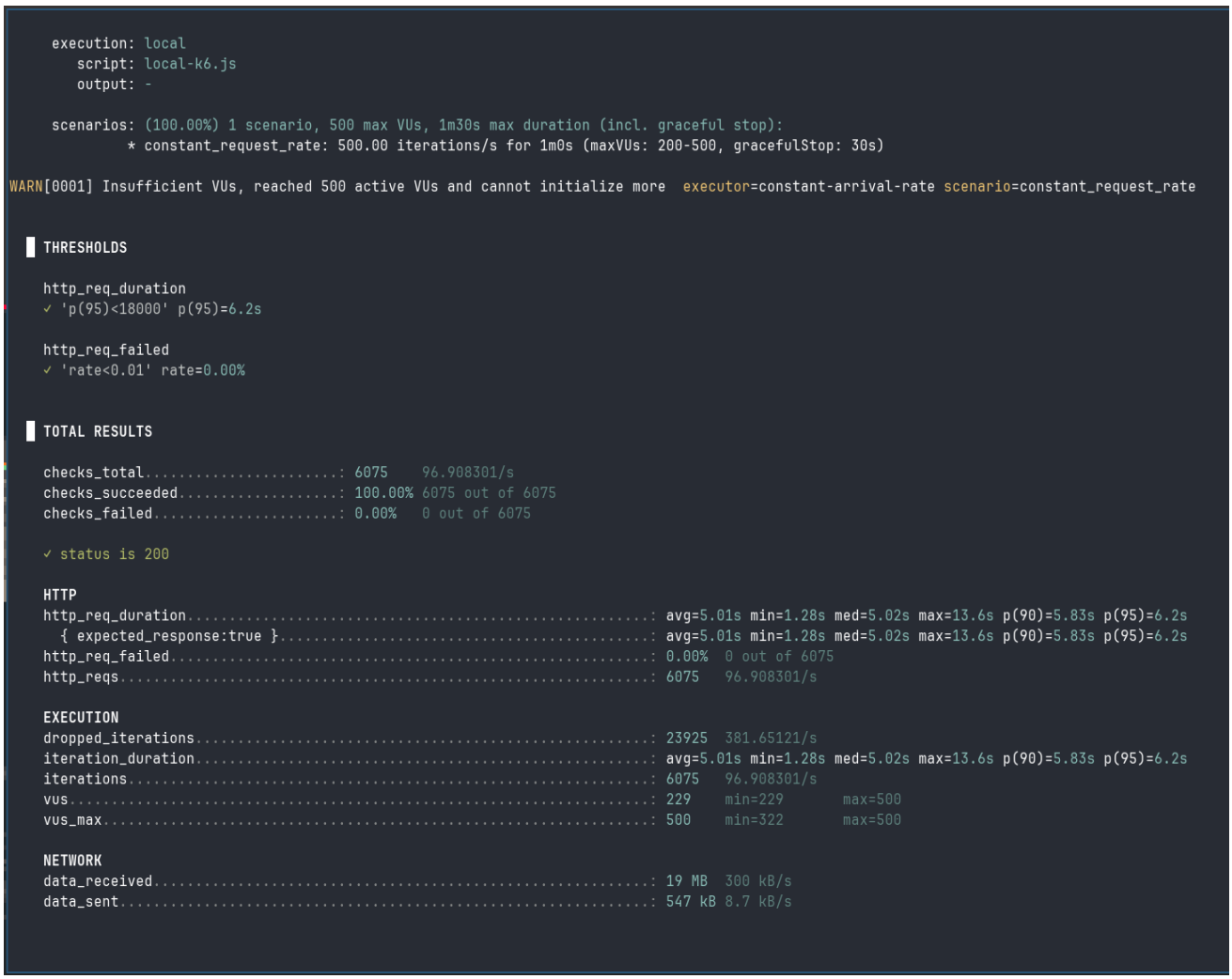

Рисунок 3.5 – Результат проходження навантажувального тесту

Висновки до третього розділу

У третьому розділі було проведено вибір методів тестування з подальшою розробкою тест-плану та описом тест-кейсів. Для тестування системи магазину додатків з мікросервісною архітектурою було проведено вибір інструментів, у результаті якого вуло вирішено обрати xUnit, Moq, TestContainers, Playwright та k6. Проведено верифікацію роботи переважної частини бізнес-логіки відповідно до заданих вимог.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено мікросервісну архітектуру системи магазину додатків, сформовану на основі аналізу сучасних підходів у сфері цифрової дистрибуції та електронної комерції. Створений програмний продукт є повноцінним прикладом прикладного рішення, що має перспективи подальшого практичного використання.

Запропонована система може слугувати як фундамент для розгортання комерційного сервісу з розповсюдження додатків або як прототип технологічної платформи для запуску стартапу з виходом на ринок мобільних чи настільних застосунків. У процесі тестування було налагоджено роботу всіх основних компонентів архітектури, забезпечено перевірку правильності реалізації бізнес-процесів та підтверджено функціональну відповідність вимогам.

Оцінка працездатності системи показала її стабільну роботу, що дозволяє розглядати її як готову до подальшого розгортання в умовах продуктивного середовища. У підсумку реалізоване рішення характеризується високим рівнем масштабованості, структурною цілісністю та потенціалом для розширення функціональності. Отримані результати свідчать про доцільність і актуальність розробки програмних систем подібного типу в умовах зростаючого попиту на цифрову інфраструктуру в e-commerce-сфері.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Statista. Biggest app stores in the world 2024. URL: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/ (date of access: 29.05.2025).
2. Цьома В.С., Марченко С.В. Програмна інженерія розробки магазинів додатків. Матеріали XVII Студентської науково-практичної конференції студентів, аспірантів та молодих вчених за тематикою «Тенденції розвитку ІТ-технологій в Україні». 26-27 березня 2025 р., Черкаси, Україна. Черкаси: Черкаський державний фаховий бізнес-коледж, 2025. С. 116-119.
3. Semerádová T., Weinlich P. The broad and narrow definition of e-commerce. *Achieving business competitiveness in a digital environment* / ed. by T. Semerádová, P. Weinlich. Cham, 2022. URL: https://doi.org/10.1007/978-3-030-93131-5_1.
4. - S. B. Customer relationship management (CRM) and e-commerce: challenges and future opportunities. *International journal for multidisciplinary research*. 2024. Vol. 6, no. 2. URL: https://doi.org/10.36948/ijfmr.2024.v06i02.16534 (date of access: 13.06.2025).
5. International Trade Administration. E-Commerce definition. URL: https://www.trade.gov/ecommerce-definition (date of access: 13.06.2025).
6. A Pattern-Based and Model-Driven Approach for Deriving IT System Functional Models from Annotated Business Models / J. Berrocal et al. *Information System Development* / ed. by M. José Escalona et al. Cham, 2014. URL: https://doi.org/10.1007/978-3-319-07215-9_26.
7. Electronic commerce 2018: a managerial and social networks perspective / E. Turban et al. 2018. URL: https://doi.org/10.1007/978-3-319-58715-8.

8. Set up your developer account - Play Console Help. URL: https://support.google.com/googleplay/android-developer/topic/7072535?hl=en (accessed 31/05/2025).
9. Create and set up your app - Play Console Help. URL: https://support.google.com/googleplay/android-developer/answer/9859152?hl=en (accessed 02/06/2025).
10. Enrollment - Membership - Account - Help - Apple Developer. URL: https://developer.apple.com/help/account/membership/program-enrollment/ (accessed 02/06/2025).
11. Apple Inc. App review guidelines. URL: https://developer.apple.com/app-store/review/guidelines/ (date of access: 02.06.2025).
12. Microsoft Corp. Learn how to publish your Windows apps and games to the Microsoft Store. URL: https://learn.microsoft.com/en-us/windows/apps/publish/ (date of access: 02.06.2025).
13. Zhu W., Others. What is an app store? The software engineering perspective. *Empirical software engineering*. 2024. Vol. 29, no. 1. URL: https://doi.org/10.1007/s10664-023-10362-3.
14. Google Inc. Google play developer API | google for developers. *Google for Developers*. URL: https://developers.google.com/android-publisher (date of access: 01.06.2025).
15. Google Inc. Run tests with firebase test lab for android - firebase help. *Google Help*. URL: https://support.google.com/firebase/answer/6386654?hl=en (date of access: 01.06.2025).
16. Google Inc. Android vitals | App quality | Android Developers. *Android Developers*. URL: https://developer.android.com/topic/performance/vitals (date of access: 16.06.2025).
17. Google Inc. Firebase crashlytics. *Firebase*. URL: https://firebase.google.com/docs/crashlytics (date of access: 01.06.2025).

18. Google Inc. Release app updates with staged rollouts - Play Console Help. *Google Help*. URL: https://support.google.com/googleplay/android-developer/answer/6346149?hl=en (date of access: 01.06.2025).
19. Apple Inc. App store connect API | apple developer documentation. *Apple Developer Documentation*. URL: https://developer.apple.com/documentation/appstoreconnectapi (date of access: 01.06.2025).
20. App Store Deployment - fastlane docs. *fastlane docs*. URL: https://docs.fastlane.tools/getting-started/ios/appstore-deployment/ (date of access: 16.06.2025).
21. Apple Inc. TestFlight overview - test a beta version - app store connect - help - apple developer. *Apple Developer*. URL: https://developer.apple.com/help/app-store-connect/test-a-beta-version/testflight-overview (date of access: 01.06.2025).
22. Apple Inc. Release a version update in phases - update your app - app store connect - help - apple developer. *Apple Developer*. URL: https://developer.apple.com/help/app-store-connect/update-your-app/release-a-version-update-in-phases/ (date of access: 01.06.2025).
23. Microsoft Corp. Create and manage submissions - UWP applications. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/windows/uwp/monetize/create-and-manage-submissions-using-windows-store-services (date of access: 01.06.2025).
24. Microsoft Corp. Health report - partner center. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/partner-center/insights/health-report (date of access: 01.06.2025).
25. Microsoft Corp. Windows SDK - Windows app development. *Microsoft Developer*. URL: https://developer.microsoft.com/en-us/windows/downloads/windows-sdk/ (date of access: 01.06.2025).

26. Microsoft Corp. Windows App Certification Kit - UWP applications. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/windows/uwp/debug-test-perf/windows-app-certification-kit (date of access: 01.06.2025).
27. Microsoft Corp. Beta testing and targeted distribution - Windows apps. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/windows/apps/publish/beta-testing-and-targeted-distribution (date of access: 01.06.2025).
28. Microsoft Corp. Gradual package rollout - Windows apps. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/windows/apps/publish/gradual-package-rollout (дата звернення: 01.06.2025).
29. Red Hat, Inc. What is a REST API?. *Red Hat - We make open source technologies for the enterprise.* URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api (дата звернення: 01.06.2025).
30. Microsoft Corp. Overview of YARP. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/servers/yarp/yarp-overview?view=aspnetcore-9.0 (дата звернення: 01.06.2025).
31. Microsoft Corp. YARP authentication and authorization. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/servers/yarp/authn-authz?view=aspnetcore-9.0 (дата звернення: 01.06.2025).
32. Nardone M. JSON web token (JWT) authentication. *Secure RESTful APIs*. Berkeley, CA, 2025. URL: https://doi.org/10.1007/979-8-8688-1285-9_4.
33. System context diagram. *C4 model*. URL: https://c4model.com/diagrams/system-context (дата звернення: 01.06.2025).
34. Container diagram. *C4 model*. URL: https://c4model.com/diagrams/container (дата звернення: 01.06.2025).

Додаток А – Посилання на репозиторій

Репозиторій з програмною реалізацією, виконаною в межах кваліфікаційної роботи, розташовано за адресою https://github.com/xlspring/hexstore