Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Срібного Данила Микитовича*
(ПІБ)

академічної групи *121-19-1*
(шифр)

спеціальності *121 «Інженерія програмного забезпечення»*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка інтернет-магазину на основі фреймворку ASP.Net з використанням HTML, CSS та C#*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Гуліна І.Г.*			
розділів:				
спеціальний	*доц. Гуліна І.Г.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент	*доц. Шедловський І.А.*			
Нормоконтролер	*ст. викл. Мартиненко А. А.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-19-1* (група) *Срібного Данила Микитовича* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка інтернет-магазину на основі фреймворку ASP.Net з використанням HTML, CSS та C#*

затверджена наказом ректора НТУ «ДП» від *16.05.2023* № *350-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів практик та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2023 р.***

Завдання видав (підпис) *доц. Гуліна І.Г.* (посада, прізвище, ініціали)

Завдання прийняв до виконання (підпис) *Срібний Д.М.* (прізвище, ініціали)

Дата видачі завдання: *14.01.2023 р.*

Термін подання кваліфікаційної роботи до ЕК: *12.06.2023 р.*

РЕФЕРАТ

Пояснювальна записка: 98 с., 14 рис., 3 дод., 24 джерела.

Об'єкт розробки: інтернет-магазин для продажу товарів широкого спектру.

Мета кваліфікаційної роботи: розробка програмного продукту для створення зручної платформи для виставлення виробниками на продаж різних товарів, які покупці зможуть замовити та придбати.

У вступі проаналізовано та описано сучасний стан проблеми, визначено конкретну мету даної кваліфікаційної роботи та галузь, в якій вона може бути застосована. Крім того, обґрунтовується актуальність обраної теми та наведено більш точне визначення поставленої задачі.

У першому розділі проведено аналіз предметної галузі та визначено актуальність завдання та його мету. Також розроблено постановку задачі та визначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проведено аналіз існуючих рішень та обрано платформу для розробки. Проводиться проектування та розробка програми, детально описується алгоритм та структура програмного продукту. Крім того, визначено вхідні та вихідні дані, наведено характеристики технічних засобів, описано процес запуску та завантаження програми, розглянуто її функціонування.

У розділі, присвяченому економічним аспектам, ми визначили трудомісткість розробки програмного продукту. Крім того, було розраховано вартість розробки додатку та оцінено час, необхідний для його створення.

Практичне значення полягає у створенні інтернет-магазину і як наслідок – збільшення економічної активності у сфері торгово-грошових відносин.

Актуальність роботи визначається великим попитом на подібні розробки, що підвищують ефективність обслуговування клієнтів інтернет-магазину.

Оскільки business-to-commerce платформи користуються великим попитом через причини пришвидшення можливостей реалізації товару, розробка інтернет-магазину буде однозначно актуальною.

Список ключових слів: E-COMMERCE, ІНТЕРНЕТ-МАГАЗИН, ПЛАТФОРМА ПРОДАЖУ, ASP.NET, RAZOR, C#, HTML, CSS.

ABSTRACT

Explanatory note: 98 p., 14 figs., 3 app., 24 sources.

The object of development: an online store for the sale of a wide range of goods.

The purpose of the qualification work: is to create a convenient platform for manufacturers to put up for sale various goods that buyers can order and purchase.

The introduction analyzes and describes the current state of the problem, and defines the specific purpose of this qualification work and the industry in which it can be applied. In addition, the relevance of the chosen topic is substantiated and a more precise definition of the task is provided.

The first chapter analyzes the subject area and determines the relevance of the task and its purpose. The task statement is also developed and the requirements for software implementation, technologies, and software tools are determined.

The second section analyzes existing solutions and selects a platform for development. The design and development of the program are carried out, and the algorithm and structure of the subsystem are described in detail. In addition, the input and output data are defined, the characteristics of technical means are given, the process of launching and loading the program is described, and its functioning is considered.

In the section devoted to economic aspects, we have determined the labor intensity of the information subsystem development. In addition, we calculated the cost of developing the application and estimated the time required to create it.

The practical significance is to create an online store and, as a result, to increase economic activity in the field of trade and monetary relations.

The relevance of the work is determined by the great demand for such developments that increase the efficiency of customer service in an online store.

Since business-to-commerce platforms are in great demand due to the reasons for accelerating the ability to sell goods, the development of an online store will be relevant.

Keywords: E-COMMERCE, ONLINE STORE, SALES PLATFORM, ASP.NET, RAZOR, C#, HTML, CSS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – веб-сервер;

ЕОМ – електронно-обчислювальна машина;

ІС – інформаційна система;

ПЗ – програмне забезпечення;

СКБД – система керування базами даних;

ІТ – інформаційні технології;

SQL – мова структурованих запитів;

MVC – архітектурний патерн модель-представлення-контроллер;

JSON – JavaScript об’єктна нотація;

E-COMMERCE – електронна комерція;

DTO – об’єкт передачі даних;

IIS – платформа розгортки веб-серверу;

HTML – гіпертекстова мова розмітки;

CSS – каскадна система стилів;

VPS – віртуальний виділений сервер;

IDE – інтегроване середовище розробки;

B2C – бізнес-до-покупця модель електронної коммерції;

HTTP – гіпертекстовий транспортний протокол;

URL – уніфікований вказник ресурсу.

ВСТУП

Більшість існуючих компаній які займаються роздрібною торгівлею в сучасні часи стикається з проблемою набуття більшої масовості свого товарообороту. Вони намагаються залучити до себе нових клієнтів лише застарілими методами, наприклад за допомогою маркетингу на різних платформах: рекламних брошурах, міських рекламних щитах, короткою рекламою по телевізору. Але, всі ці методи можуть забезпечити лише тимчасову зацікавленість потенційного клієнта, не надаючи йому зручної можливості придбати той, чи інший товар, ця зацікавленість просто сплине - ніяк не принісши користі магазину. Саме цю проблему і вирішує електронна комерція, або ж e-commerce.

E-commerce сприяє розвитку бізнесу, дозволяючи компаніям збільшувати свою аудиторію та здійснювати продажі на віддаленому ринку без необхідності фізично присутніх магазинів. Воно також допомагає компаніям керувати запасами та знижувати витрати на зберігання товарів. Використання інтернет-платформ надає можливість збирати та аналізувати дані про клієнтів, що допомагає підприємствам краще розуміти їх потреби та пропонувати персоналізовані товари та послуги. Це сприяє задоволеності клієнтів та підвищує шанси на повторні покупки.

Електронний підхід до ведення бізнесу дає можливість малим підприємствам та початківцям займатися бізнесом з мінімальними витратами, що стимулює підприємницьку активність та сприяє зростанню нових бізнесів. В цілому, e-commerce надає безліч можливостей для підприємств будь-якого розміру, допомагаючи їм проникнути на глобальний ринок, пропонувати свої товари та послуги в будь-якому куточку світу та взаємодіяти з клієнтами безпосередньо через Інтернет що сприяє економічному зростанню, конкуренції та інноваціям у бізнесі.

Електронна комерція (e-commerce) може бути класифікована на різні види в залежності від характеру учасників та способу здійснення торгівлі. Одним із найпоширеніших типів електронної комерції є бізнес-до-споживача

(B2C), який означає продаж товарів та послуг безпосередньо від підприємства (бізнесу) до кінцевого споживача.

B2C-торгівля включає в себе інтернет-магазини, де споживачі можуть переглядати, вибирати та придбавати товари та послуги через Інтернет. Цей тип торгівлі може охоплювати широкий спектр товарів, від одягу та електроніки до їжі та подорожей.

B2C-торгівля має кілька переваг для підприємств. Вона дозволяє розширити ринок збуту, залучити нових клієнтів та конкурувати на глобальному рівні. Крім того, цей вид електронної комерції дозволяє підприємствам збирати дані про своїх клієнтів, аналізувати їх покупкові звички та персоналізувати пропозиції для кращого задоволення потреб споживачів.

Для кінцевих споживачів B2C-торгівля також має вагомі переваги. Вони мають зручний доступ до широкого асортименту товарів та послуг без необхідності фізично відвідувати магазини. Клієнти можуть порівнювати ціни, читати відгуки, знаходити рекомендації та здійснювати покупки з будь-якого місця та в будь-який час.

Веб-технології та програмне забезпечення є невід'ємною складовою електронної комерції (e-commerce) і мають важливе значення для її розвитку. Веб-сайти та веб-додатки є основними засобами для створення та управління електронними торговими платформами, та зазвичай містять інформацію про товари, послуги, каталоги, корзини покупок та платіжні системи. Вони забезпечують прийом замовлень, підтвердження оплати, відстеження доставки і оновлення інформації про склад товарів.

В цілому, інтернет-торгівля і веб-технології взаємопов'язані і взаємодоповнюють один одного. Веб-технології надають потужний інструментарій для створення, розвитку та оптимізації веб-сайтів та платформ електронної комерції, забезпечують безпеку, зручність та ефективність операцій в інтернет-торгівлі.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Веб-сайт (або просто сайт) - це колекція веб-сторінок, пов'язаних між собою гіперпосиланнями і розміщених на веб-сервері, до яких надається доступ через Інтернет і яка відображається за допомогою веб-браузера на комп'ютерах, смартфонах або інших пристроях. Веб-сайти використовуються для різних цілей, таких як розповсюдження інформації, комунікація, розваги, освіта, електронна комерція та багато іншого. Вони можуть бути статичними (зміст на сторінках не змінюється) або динамічними (змінюються відповідно до запитів користувачів або внутрішньої логіки).

У процесі роботи веб-сайти взаємодіють з клієнтськими пристроями (наприклад, комп'ютерами користувачів) через Інтернет. Користувачі вводять URL-адресу веб-сайту у веб-браузері, і браузер надсилає запит на сервер, де розміщений веб-сайт. Сервер обробляє запит і надсилає відповідь у вигляді HTML-коду, який браузер інтерпретує і відображає на екрані користувача. Крім HTML, веб-сайти можуть використовувати CSS для оформлення сторінок і JavaScript для створення інтерактивності.

Інтернет-магазин (також відомий як електронний магазин або онлайн-магазин) - це веб-сайт, який дозволяє користувачам здійснювати покупки товарів або послуг через Інтернет. Вони дозволяють покупцям переглядати каталог товарів, додавати їх до кошика, оформляти замовлення та здійснювати оплату онлайн, також вони можуть мати різні функціональні можливості, такі як: фільтри товарів, рейтинги, огляди, порівняння товарів, пошук по ключових словах і багато іншого. Ще інтернет-магазини зазвичай мають інтеграцію з

різними платіжними системами, що дозволяє користувачам здійснювати оплату за товари з різних країн та валют.

Розробка як інтернет-магазину (як і інших веб-додатків) - це складний процес, який потребує великої кількості ресурсів і знань. Фреймворки для розробки веб-додатків є інструментами, які допомагають розробникам ефективно і швидко створювати веб-додатки, зменшуючи кількість коду та забезпечуючи готові рішення для важливих задач.

Один з найбільш популярних фреймворків для розробки веб-додатків є ASP.NET. Його розвиває компанія Microsoft і він забезпечує зручну розробку веб-додатків на мові програмування C# або Visual Basic. ASP.NET включає в себе такі готові компоненти як авторизація, аутентифікація, кешування та багато інших, що значно спрощує розробку.

Інтернет-магазини, які розробляються з використанням ASP.NET, мають кілька переваг. По-перше, ASP.NET забезпечує високу продуктивність, що дозволяє обслуговувати велику кількість клієнтів та операцій, завдяки оптимізації і оптимальному використанню ресурсів. По-друге, він надає можливість легко інтегрувати зовнішні сервіси, такі як платіжні шлюзи або системи управління запасами, що полегшує автоматизацію процесів в інтернет-магазині. По-третє, розробка на ASP.NET є зручною та швидкою завдяки великій кількості готових рішень та зручним інструментарієм для розробників.

Нарешті, ASP.NET є масштабованим фреймворком, що дозволяє розробникам легко розширювати та змінювати функціональність інтернет-магазину залежно від зростання бізнесу та потреб клієнтів, за допомогою додавання нових модулів, покращення продуктивності та забезпечення безпеки інтернет магазину.

Ще одним важливим рішенням, після обирання технологій для розробки сайту, є обирання засобів для хостингу веб-сайту.

Хостинг веб-сайту - це процес розміщення веб-сайту на сервері, щоб він був доступний через Інтернет. При хостингу веб-сайту, розміщуються його файли, бази даних, медіа контент і інші ресурси на сервері, який постачає послуги збереження та доступу до цих файлів. Хостинг веб-сайту важливий, оскільки він забезпечує постійну доступність сайту в Інтернеті.

І для цілей хостингу, в Microsoft є якісний інструмент, який називається IIS.

IIS (Internet Information Services) є веб-сервером, розробленим компанією Microsoft, який може бути використаний для хостингу веб-сайтів. IIS має деякі ключові переваги для хостингу інтернет-магазинів, розроблених на базі фреймворку ASP.NET:

– Інтеграція з ASP.NET: IIS відмінно підтримує фреймворк ASP.NET, оскільки обидва розробляються компанією Microsoft. Що призводить до оптимальної продуктивності та сумісністю між сервером і фреймворком, що сприяє ефективній роботі інтернет-магазину.

– Завантаження та продуктивність: IIS є високопродуктивним веб-сервером, який може ефективно обробляти велику кількість запитів. Це важливо для інтернет-магазину, оскільки потрібно забезпечити швидку реакцію на запити користувачів та оптимальну продуктивність при обробці операцій купівлі-продажу.

– Безпека: IIS має вбудовану систему безпеки, яка дозволяє забезпечити захист інтернет-магазину від потенційних загроз.

– Легка масштабованість: IIS дозволяє легко масштабувати інтернет-магазин. Він надає можливість налаштувати балансування навантаження для розподілу трафіку між декількома серверами, що дозволяє забезпечити стабільну роботу навіть при великому обсязі користувачів.

Для роботи інтернет-магазину необхідна база даних з розширеним функціоналом і можливістю обробки великої кількості даних. Вона використовується для зберігання і керування інформацією про товари,

замовлення, клієнтів, оплату, доставку та інші аспекти, пов'язані з функціонуванням магазину.

База даних дозволяє зберігати і керувати даними про товари, включаючи назви, опис, ціни, зображення та характеристики. Це дозволяє ефективно відстежувати наявність товарів, оновлювати їхню інформацію та надавати користувачам актуальну інформацію про товари.

Також база даних дозволяє зберігати дані про замовлення, включаючи інформацію про клієнта, вибрані товари, кількість, ціну та статус замовлення. Це дозволяє відстежувати статуси замовлень, обробляти платежі та забезпечувати ефективну обробку та доставку замовлень.

Управління клієнтами також реалізоване через базу даних, де можна зберігати дані про клієнтів, такі як особисті дані, історію покупок, адреси доставки та інше. Це дає можливість відстежувати покупців, аналізувати їхню поведінку, надавати персоналізовані рекомендації та забезпечувати зручність при оформленні замовлень.

Звертаючи увагу що розробка інтернет магазину ведеться на базі фреймворку ASP.NET та веб-серверу IIS, використання MS SQL може надати декілька переваг порівняно з іншими системами керування базами даних (СКБД):

– Інтеграція з Microsoft-технологіями: MS SQL розроблено компанією Microsoft, тому воно має глибоку інтеграцію з іншими продуктами Microsoft, такими як ASP.NET та IIS. Це дозволяє забезпечити зручну розробку, підтримку та сумісність між цими компонентами, що спрощує процес розробки і підтримки інтернет-магазину.

– Висока продуктивність та масштабованість: MS SQL забезпечує високу продуктивність та швидку обробку запитів, що є важливим для інтернет-магазинів з великим обсягом даних і великою кількістю транзакцій. Він підтримує оптимізовані механізми запитів, індексацію та кешування даних, що сприяє швидкій роботі магазину. Також, він має вбудовані механізми масштабування, такі як розподілені резервні копії та кластеризація,

що дозволяє розширювати функціональність магазину зростаючими потребами.

– Розширені можливості та функціональність: MS SQL надає багато функціональних можливостей, таких як тригери, процедури, функції та індекси, що дозволяють ефективно працювати з базою даних. Він також підтримує різні типи даних, включаючи текстові, числові, дати, геодані та багато інших.

– Надійність та безпека: MS SQL підтримує механізми резервного копіювання та відновлення, які дозволяють забезпечити відновлення даних в разі непередбачуваних ситуацій. Крім того, він має вбудовані механізми безпеки, такі як рівень доступу до даних, шифрування даних та аудит дій користувачів, що забезпечують конфіденційність та цілісність даних магазину.

1.2. Призначення розробки та галузь застосування

Призначенням розробки є створення інтернет-магазину яке полягає в структуруванні різних товарів в якості інтернет-каталогу та реалізації цих товарів за допомогою різних швидких та надійних систем електронних платежів.

Магазин повинен надавати кінцевому користувачу різні зручні інструменти для роботи з електронним каталогом, такі як фільтрація за певними категоріями товару, сортування за ціною або пошук за назвою. Також необхідна наявність деякого віртуального кошику покупок, де будуть зберігатись товари, з яких користувач буде формувати замовлення.

Також інтернет-магазин повинен мати інструменти для адміністрування, який буде дозволяти працівникам інтернет-магазину (далі «адміністратори») змінювати замовлення, здійснювати керування доступом до тих чи інших частин магазину, додавати, видаляти або змінювати товари, здійснювати модерацію відгуків покупців.

Розроблений програмний продукт може бути застосований в галузі роздрібної торгівлі.

1.3. Підстава для розробки

Підставою для розробки кваліфікаційної роботи на тему « Розробка інтернет-магазину на основі фреймворку ASP.Net з використанням HTML, CSS та C#» є наказ ректора по Національному технічному університету «Дніпровська політехніка» № 350-с від 16.05.2023р.;

Освітня програма спеціальності 121 «Інженерія програмного забезпечення».

1.4. Постановка завдання

Завданням до даної кваліфікаційної роботи, є розробка інтернет-магазину для роздрібної торгівлі.

Програмне забезпечення призначене для надання універсального інструменту для відображення контенту, формування замовлень та адміністрування даного інтернет-магазину.

Програма повинна реалізовувати наступні функції:

- формування та заповнення веб-сторінок на основі шаблонів та даних моделей;
- можливість додання, видалення та редагування різних товарів у панелі адміністратора;
- оформлення замовлень;
- розмежування доступу до різних частин веб-додатку між користувачами.

Для досягнення поставленої мети необхідно:

- вивчити предметну галузь розв’язуваної задачі;

- спроектувати архітектуру для веб-додатку;
- написати код для реалізації поставленого завдання;
- створити базу даних, клієнтську та серверну частину програми, що працює з нею.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Для досягнення поставлених цілей - програмне забезпечення, що розробляється, повинно підтримувати наступні функції:

Функціональні вимоги до інтернет-магазину:

- Користувачі повинні мати можливість створити обліковий запис і надати необхідну інформацію для реєстрації.
- Продукт повинен аутентифікувати користувачів для забезпечення безпечного доступу до їхніх облікових записів.
- Інтернет-магазин повинен відображати повний каталог товарів, доступних для придбання.
- Продукти повинні бути класифіковані та доступні для пошуку за різними критеріями, такими як категорія, ціна, назва тощо.
- Користувачі повинні мати можливість додавати товари до кошика для подальшої покупки.
- Кошик повинен дозволяти користувачам змінювати кількість, видаляти товари.
- Користувачі повинні мати можливість перейти до оформлення замовлення, ввести інформацію про доставку та вибрати спосіб оплати.
- Користувачі повинні мати можливість переглядати історію своїх замовлень, включаючи статус замовлення, деталі доставки та інформацію про відстеження.

- Продукт повинен дозволяти користувачам скасовувати або змінювати замовлення протягом певного періоду часу, якщо це можливо.
- Інтернет-магазин повинен відстежувати товарні запаси та оновлювати інформацію про наявність товарів у режимі реального часу.
- Коли товар купується, система повинна віднімати його кількість з наявного запасу.
- Користувачі повинні мати можливість залишати відгуки для товарів
- Система повинна відображати відгуки, щоб допомогти іншим користувачам приймати обґрунтовані рішення.
- Адміністратори повинні мати можливість модерувати відгуки та видаляти неприйнятний контент, якщо це необхідно.
- Користувачі повинні мати можливість відстежувати статус своїх замовлень.
- Інтернет-магазин повинен підтримувати кілька мов і валют, щоб обслуговувати глобальну аудиторію.
- Користувачі повинні мати можливість вибрати бажану мову та валюту для персоналізованого шопінгу.

1.5.2. Вимоги до інформаційної безпеки

Вимоги до безпеки інтернет-магазину:

- Система повинна забезпечувати безпечні методи автентифікації користувачів, такі як надійні паролі.
- Сесії користувачів повинні безпечно управлятися, щоб запобігти несанкціонованому доступу.

- Механізми контролю доступу мають бути реалізовані таким чином, щоб користувачі могли отримати доступ лише до відповідних ресурсів відповідно до їхніх ролей та привілеїв.
- Інтернет-магазин повинен використовувати безпечні протоколи зв'язку, такі як HTTPS, для шифрування передачі даних між браузером користувача та сервером.
- Система повинна реалізовувати заходи для захисту від поширених атак на веб-додатки, таких як міжсайтовий скриптинг (XSS), SQL-ін'єкції та підробка міжсайтових запитів (CSRF).
- Для запобігання ін'єкційним атакам та виконанню шкідливого коду слід застосовувати валідацію вхідних даних.
- Система повинна реалізовувати механізми реєстрації для запису та моніторингу діяльності системи, дій користувачів та подій безпеки.
- Інтернет-магазин повинен мати надійний план аварійного відновлення та процедури резервного копіювання, щоб гарантувати, що критичні дані та послуги можуть бути відновлені в разі катастрофічних подій.

1.5.3. Вимоги до складу та параметрів технічних засобів

Вимоги до апаратного забезпечення для хостингу інтернет-магазину, розробленого на ASP.NET, IIS і Microsoft SQL:

- багатоядерний процесор з тактовою частотою не менше 2 ГГц або вище;
- мінімум 8 ГБ оперативної пам’яті;
- рекомендовано 16 ГБ оперативної пам'яті або більше;
- жорсткий диск (HDD) або твердотільний накопичувач (SSD) з мінімальним обсягом пам’яті 100гб;
- надійне підключення до Інтернету з достатньою пропускною здатністю для обробки очікуваного трафіку.

1.5.4. Вимоги до інформаційної та програмної сумісності

Задля нормальної роботи програми потрібна веб-орієнтована система, яка відповідає наступним вимогам:

- Операційна система Windows (7+), Linux, MacOS;
- Веб-браузер Google Chrome / Firefox / Opera / Safari.

Серверна частина інтернет-магазину має бути реалізована на мові програмування C#, з використанням фреймворку для розробки веб-додатків ASP.NET, та шаблонного типізатору Razor. Робота буде створена за допомогою MVC архітектурного підходу та SOLID принципів.

Клієнтська частина інтернет-магазину буде реалізована за допомогою HTML, CSS та JS.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Функціональне призначення онлайн-магазину полягає в тому, щоб прискорити роздрібну реалізацію того, чи іншого продукту завдяки зручній для користувача та функціональній торгової платформи. Для цього воно повинне реалізовувати наступні речі:

- ведення баз даних для обліку товарів, їх кількість, ціни, замовлення з цими товарами тощо;
- надання адміністративних інструментів для добавлення, редагування або ж видалення товарів;
- розмежовування прав доступу різних користувачів до різних частин програми;
- реалізація можливості використання різних платіжних систем;
- глобалізація інтернет-магазину та надання можливості обрати різні локалізації та мови сайту;
- можливість складання замовлень у інтернет-кошику з покупками.

Для надання можливостей для реалізацій вище-приведених функцій, програмне забезпечення повинно підтримувати виконання таких операцій:

- підтримку роботи з cookie-файлами браузеру задля забезпечення коректної механізмів авторизації;
- можливість обробки JSON-запитів як вхідні данні для користування ресурсами веб-серверу;
- зберігання даних в реляційній та нереляційній базі даних.

2.2. Опис застосованих математичних методів

Оскільки особливості предметної галузі розв'язуваної задачі не передбачають застосування математичних методів, при розробці програмного продукту математичні методи не використовувалися.

2.3. Опис використаної архітектури та шаблонів проектування

При розробці програми було створено та використано багаторівневу структура додатку, в якій кожна логічна частина програми було інкапсульовано в окремому рівні

- рівень доступу до даних: рівень доступу до даних відповідає за взаємодію створеного додатку із зовнішніми ресурсами, які містять данні (інтерфейсі доступу до баз даних, провайдери для роботи з файлами, робота з кешем, тощо);
- рівень бізнес-логіки: цей рівень відповідає з основні бізнес-процеси та правила - які повинен виконувати та яким повинен слідкувати додаток. Бізнес-логікою додатку зазвичай вважається процеси які стосуються саме алгоритмів та обробки даних додатком, наприклад бізнес логікою вважаються: шифрування паролів користувачів, взаємодія з отриманими даними, формування замовлень онлайн магазину, бронювання товарів і та інше;
- рівень презентації: цей рівень відповідає за відображення даних на клієнтській частині додатку. На цьому рівні зберігаються механізми маршрутизації, тощо.

Для передачі моделей поміж рівнями, були використані домені моделі, це класи різних сутностей додатку, які позбавлені будь-яких специфічних полів, колекцій, атрибутів та інших речей – що є специфічними для якогось з рівнів. Перед тим, як заповнена модель буде передана на наступний рівень додатку, вона упаковується моделі узагальненого для всіх рівнів виду.

Задля забезпечення універсальності, рівень доступу до даних був спроектований за шаблоном проектування «Міст» (Bridge), за допомогою узагальнення інтерфейсу доступу до даних та винесення його у абстракцію, що надає нам можливість змінювати нашу реалізацію рівня доступу до даних без внесення змін до інших рівнів.

Також для проектування додатку було використано архітектуру MVC. Модель-Вид-Контролер (MVC) - це популярний архітектурний патерн, який використовується при розробці веб-додатків, а також застосовується в ASP.NET Core. MVC забезпечує структурований підхід до створення веб-додатків, розділяючи завдання маніпулювання даними (модель), рендерингу користувацького інтерфейсу (представлення) та обробки і маршрутизації запитів (контролер). Таке розмежування забезпечує кращу організацію коду, можливість тестування та супроводу.

В ASP.NET Core патерн MVC реалізований через простір імен `Microsoft.AspNetCore.Mvc`, який надає необхідні компоненти та інфраструктуру для розробки веб-додатків з використанням архітектури MVC. Ось короткий огляд ключових компонентів ASP.NET Core MVC:

- Модель: модель представляє дані та бізнес-логіку додатку. Вона інкапсулює операції доступу до даних, валідації та маніпуляції з ними.
- Представлення (View): View відповідає за відображення користувацького інтерфейсу. Він генерує HTML-розмітку або інші формати, які надсилаються до браузера клієнта.
- Контролер: контролер обробляє вхідні запити, обробляє їх і визначає відповідну відповідь. Він взаємодіє з моделлю для отримання або оновлення даних і вибирає відповідне представлення для відображення відповіді.
- Маршрутизація: маршрутизація визначає, як вхідні запити зіставляються з відповідними контролерами та методами дій. ASP.NET Core MVC використовує гнучку і настроювану систему маршрутизації,

яка дозволяє визначати маршрути на основі шаблонів URL і параметрів маршруту.

2.4. Опис використаних технологій та мов програмування

Інтернет-магазин був реалізований на базі фреймворку для розробки веб-додатків ASP.NET Core, мові програмування C#, у якості СКБД був використаний Microsoft SQL Server. Для клієнтської частини були використані такі технології як: шаблоний типізатор Razor, мова гіпертекстової розмітки HTML, каскадні таблиці стилів CSS, та фреймворк для верстки сторінок Bootstrap.

C# - це універсальна і широко використовувана мова програмування, розроблена компанією Microsoft. Вона відома своєю простотою, підтримкою об'єктно-орієнтованого програмування та широким спектром застосувань, включаючи десктопне програмне забезпечення, веб-розробку, мобільні додатки та розробку ігор. C# є об'єктно-орієнтованою мовою, а це означає, що вона підтримує такі поняття, як класи, об'єкти, успадкування, поліморфізм та інкапсуляція. Вона сприяє організації коду та повторному використанню, що полегшує створення складних додатків.

Однією з ключових особливостей C# є безпека типів. Змінні в C# повинні бути оголошені з конкретними типами, що забезпечує кращу надійність коду та зручність його супроводу. C# також включає автоматичне керування пам'яттю за допомогою збирача сміття, що спрощує керування пам'яттю та зменшує ймовірність витоку пам'яті.

Мова надає багату стандартну бібліотеку, таку як .NET Framework або .NET Core Library, яка пропонує широкий спектр класів і функцій для різних завдань програмування. Це включає в себе файловий ввід/вивід, роботу в мережі, доступ до баз даних та багато іншого. C# підтримує асинхронне програмування, що дозволяє розробникам писати ефективні та швидкі додатки. Ключові слова `async` та `await` полегшують написання

асинхронного коду, який не блокує основний потік, що призводить до кращого використання ресурсів.

ASP.NET Core - це кросплатформенна платформа для веб-розробки з відкритим вихідним кодом, розроблена компанією Microsoft. Це сучасний модульний фреймворк, який дозволяє розробникам створювати високопродуктивні, масштабовані та хмарні веб-додатки та API. ASP.NET Core є значним розвитком попереднього фреймворку ASP.NET і пропонує кілька покращень та нових функцій.

Ось деякі ключові аспекти та особливості ASP.NET Core:

1. Висока продуктивність: ASP.NET Core розроблений для забезпечення виняткової продуктивності. Він представляє новий веб-сервер під назвою Kestrel, який є легким та високоефективним. Крім того, ASP.NET Core використовує новітні функції та технології для оптимізації обробки запитів та покращення часу відгуку.
2. Модульність та легкість: ASP.NET Core має модульну архітектуру, що дозволяє розробникам включати в свої додатки лише необхідні компоненти. Це призводить до зменшення розміру додатків і зниження накладних витрат. Фреймворк також забезпечує вбудовану ін'єкцію залежностей для сприяння вільному з'єднанню та зручності обслуговування.
3. Уніфікована модель програмування: ASP.NET Core уніфікує модель програмування для створення веб-інтерфейсів та веб-додатків. Це означає, що розробники можуть використовувати один і той же фреймворк і шаблони для створення обох типів додатків, що дозволяє повторно використовувати код і спрощує розробку.
4. Конвеєр проміжного програмного забезпечення: ASP.NET Core представляє конвеєр проміжного програмного забезпечення, який дозволяє розробникам складати і налаштовувати потік обробки запитів/відповідей. Компоненти проміжного програмного забезпечення можна додавати, видаляти або змінювати порядок для обробки різних

аспектів циклу запиту/відповіді, таких як аутентифікація, маршрутизація, ведення журналів і кешування.

ASP.NET Core набув популярності завдяки своїй гнучкості, продуктивності та кросплатформенності. Він дозволяє розробникам створювати сучасні веб-додатки та API, які є масштабованими, ефективними та легко підтримуваними.

MS SQL, розроблений Microsoft, є потужною системою управління реляційними базами даних. Він забезпечує надійну і масштабовану платформу для зберігання, управління та вилучення даних у різних додатках і корпоративних середовищах.

MS SQL використовує реляційну модель даних, яка організовує дані в таблиці з певними стовпцями та рядками. Ці таблиці можуть бути пов'язані одна з одною, даючи змогу встановлювати зв'язки між даними. MS SQL підтримує стандартну мову запитів SQL, яка дає змогу розробникам виконувати операції читання, запису, оновлення та видалення даних.

Одна з важливих особливостей MS SQL - це підтримка ACID-властивостей (Atomicity, Consistency, Isolation, Durability). ACID-властивості гарантують, що транзакції в базі даних є надійними, а зміни даних або повністю виконуються, або не виконуються зовсім. Також він забезпечує високу продуктивність і масштабованість. СКБД підтримує поділ даних (partitioning), що дає змогу розподіляти дані між кількома файловими групами або серверами. Ще він пропонує оптимізацію запитів, індексування та можливості роботи з пам'яттю, щоб підвищити продуктивність додатків.

Безпека даних є важливим аспектом MS SQL. Він надає механізми автентифікації та авторизації, що дають змогу керувати доступом користувачів і дозволами. MS SQL також підтримує шифрування даних у спокої та під час передачі, забезпечуючи безпеку конфіденційних даних.

HTML, скорочено від Hypertext Markup Language - це стандартна мова розмітки, яка використовується для створення та структурування вмісту веб-сторінок. Вона є основою кожної веб-сторінки в Інтернеті, дозволяючи

представляти та організовувати текст, зображення, мультимедіа та інші елементи. HTML використовує систему тегів для визначення структури і формату вмісту. Теги беруться в кутові дужки (<>) і йдуть парами: відкриваючий тег і закриваючий тег. Вміст веб-сторінки розміщується між цими тегами, визначаючи її структуру та зовнішній вигляд.

Ось базовий приклад структури HTML-документа:

```
<!DOCTYPE html>
<html>
<head>
      <title>My Webpage</title>
</head>
<body>
      <h1>Welcome to My Webpage</h1>
      <p>This is a paragraph of text.</p>
      <img src="image.jpg" alt="An image">
</body>
</html>
```

У цьому прикладі:

- `<!DOCTYPE html>` оголошує тип документа як HTML5;
- `<html>` кореневий елемент, який обгортає весь HTML-вміст;
- `<head>` містить метаінформацію про веб-сторінку, наприклад, заголовок, що відображається в рядку заголовка браузера;
- `<title>` визначає заголовок веб-сторінки;
- `<body>` містить видимий вміст веб-сторінки;
- `<h1>` є тегом заголовка і визначає найбільший заголовок;
- `<p>` тег абзацу, який використовується для визначення абзаців тексту;
- `<img>` тег зображення, який використовується для вбудовування зображень, з атрибутом `src`, що вказує джерело зображення, і атрибутом `alt`, що надає альтернативний текст для доступності.

HTML надає широкий спектр тегів і атрибутів для структурування і форматування веб-вмісту. Він підтримує посилання, списки, таблиці, форми,

мультимедійні елементи тощо. Крім того, HTML можна розширити за допомогою каскадних таблиць стилів (CSS) і JavaScript, щоб додати стилю та інтерактивності веб-сторінкам.

CSS, що розшифровується як Cascading Style Sheets (каскадні таблиці стилів), - це мова таблиць стилів, яка використовується для опису презентації та візуального стилю документів HTML і XML. Вона дозволяє веб-розробникам керувати макетом, кольорами, шрифтами та іншими візуальними аспектами веб-сторінки окремо від базової структури, визначеної HTML. За допомогою CSS можна створювати цілісні та візуально привабливі веб-сторінки, застосовувати різні стилі до різних елементів і легко вносити зміни до презентації, не змінюючи вмісту. CSS працює, пов'язуючи певні правила стилів з елементами HTML за допомогою селекторів та оголошень.

CSS дозволяє використовувати різні типи селекторів, включаючи селектори елементів, класів, ідентифікаторів, атрибутів тощо. Селектори дають змогу вибрати певні елементи або групи елементів для вибіркового застосування стилів. CSS можна включити в HTML-документ за допомогою тегу `<style>` у розділі `<head>`, або ж розмістити у зовнішньому CSS-файлі і зв'язати його з HTML-документом за допомогою тегу `<link>`. Зовнішні CSS-файли зазвичай використовуються для кращої організації та полегшення підтримки стилів на декількох веб-сторінках.

Bootstrap - це популярний фронтенд-фреймворк, який спрощує створення адаптивних і візуально привабливих веб-сайтів та веб-додатків. Він надає набір попередньо розроблених компонентів HTML, CSS та JavaScript, а також систему сітки та утилітарні класи. За допомогою Bootstrap можна легко створювати адаптивні макети, впроваджувати загальні шаблони дизайну та налаштовувати зовнішній вигляд вашого проекту.

Однією з ключових особливостей Bootstrap є його адаптивна система сітки, яка дозволяє створювати гнучкі макети, що адаптуються до різних розмірів екрану. Ця система сітки базується на 12-колоночному макеті і надає можливості для визначення поведінки елементів на різних розмірах екрану.

Bootstrap також пропонує широкий вибір попередньо стилізованих компонентів інтерфейсу, включаючи навігаційні панелі, кнопки, форми, картки, модальні елементи та каруселі. Ці компоненти готові до використання і можуть бути легко налаштовані відповідно до ваших вимог до дизайну. Використовуючи ці компоненти, можна заощадити час і забезпечити узгодженість стилю впродовж усього проекту. Крім того, Bootstrap постачається з файлом CSS, що містить попередньо визначені стилі та класи. Ці стилі можна застосувати до ваших елементів, щоб досягти послідовного та візуально привабливого вигляду. Фреймворк також надає систему тем, що дозволяє налаштовувати зовнішній вигляд проекту, змінюючи змінні або створюючи власні теми.

Кросбраузерна сумісність - ще одна перевага Bootstrap. Він розроблений так, щоб добре працювати в різних браузерах, забезпечуючи узгодженість зовнішнього вигляду і функціональності.

Щоб використовувати Bootstrap, вам потрібно включити файли Bootstrap CSS і JavaScript у ваш HTML-документ. Можна завантажити ці файли або скористатися посиланням на розміщену версію. Після включення можна почати використовувати класи і компоненти Bootstrap для структурування і стилізації вашого веб-контенту.

JavaScript (JS) - це динамічна мова програмування, яка широко використовується у веб-розробці. Вона забезпечує інтерактивну поведінку на веб-сайтах, покращуючи користувацький досвід. JS знаходить застосування в серверній розробці, розробці мобільних і десктопних додатків.

JavaScript полегшує написання сценаріїв на стороні клієнта, виконуючись у веб-браузерах. Він взаємодіє з HTML і CSS, дозволяючи створювати динамічний контент, валідацію форм, зворотній зв'язок у реальному часі та асинхронні запити. Мова підтримує об'єктно-орієнтовані принципи програмування, що дозволяє розробникам створювати багаторазовий і модульний код, використовуючи об'єкти з властивостями і методами.

JavaScript слідує моделі, керованій подіями, реагуючи на дії користувача, такі як кліки та натискання клавіш, підвищуючи інтерактивність веб-додатків. Він сумісний з різними платформами, працює в основних веб-браузерах і різних операційних системах, що дозволяє розробникам створювати крос-платформні додатки.

Асинхронне програмування є важливою особливістю JavaScript, що дозволяє ефективно обробляти трудомісткі операції. Воно допомагає створювати AJAX-запити, отримувати дані та виконувати фонові обчислення, не блокуючи основне виконання.

2.5. Опис структури програмного продукту та алгоритмів її функціонування

Програмний продукт було під час розробки розбито на різні рівні.

1. Data Access Layer: це рівень доступу до даних, в ньому зібрані моделі, поля яких репрезентують собою колонки таблиць (або документів, якщо це NoSQL-БД). Ці моделі заповнюються за допомогою різних репозиторіїв, які у свою чергу діляться на 3 види: реляційні (уособлюють у собі лише CRUD-функціонал додатку до SQL-БД), нереляційні (містять у собі Read/Update операції та реляційний механізм) та загальні репозиторії (є узагальненим інтерфейсом для виконання CRUD-операцій з даними, керує роботою обох репозиторіїв).
2. Abstract Data Access Layer: це рівень який надає абстрактний узагальнений інтерфейс, який не містить самої реалізації та лише надає можливість звертатись до інтерфейсів репозиторіїв. Таке рішення проектування надає можливість змінювати деталі реалізації доступу до даних без необхідності змінювати бізнес-логіку. Також інтерфейси цього рівня повертають доменні моделі.

3. Business Logic Layer: на цьому рівні знаходяться різноманітні сервіси додатку, які містять у собі бізнес-правила для різних алгоритмів, від яких залежить спосіб обробки даних. На цьому рівні знаходяться різні файли налаштувань, класи сервісів платіжних систем, методи розширення, користувацькі виключення. Також на цьому рівні збираються замовлення покупця, виконується бронювання різних продуктів, які користувач додає до замовлення, та виконується виконання його замовлення. На цьому рівні, вся бізнес-логіка здійснює обробку саме над доменними моделями, оскільки саме вимоги бізнес-логіки є критичними для коректної та очікуваної роботи програми.
4. Presentation Layer: на цьому рівні, доменні моделі над якими було виконано обробку згідно бізнес-правил додатку спочатку перетворюються до DTO-моделей, сериалізуються у JSON-моделі, потім підставляються шаблонізатором Razor на HTML сторінки і відправляються покупцеві. На цьому рівні ключовими об’єктами є класи контролерів, які виконують перетворення з доменних моделей у DTO-моделі, з яких потім за необхідності (якщо модель, необхідна для сторінки є комплексною) модель контексту представлення за допомогою різних комплектувачів моделей, та які потім надсилаються як HTML сторінки до браузерів користувачів.

Нижче наведена схема структури додатку (рис. 2.1.):

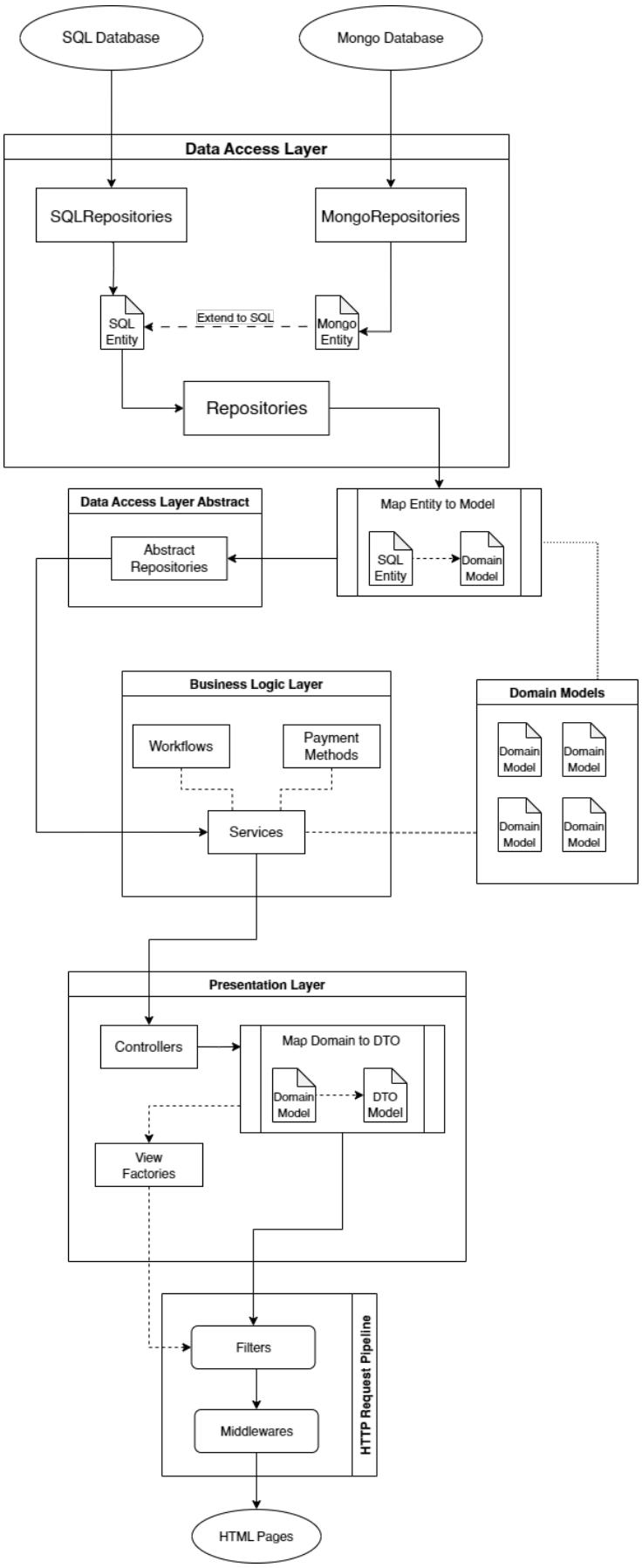

Рис. 2.1. UML-діаграма структури додатку

База даних містить у собі дуже багато таблиць, які має сенс умовно розділити за сутностями до яких вони відносяться:

- Продукти та ігри магазину: Games (таблиця з продуктами та іграми магазину), Comments (відгуки користувачів магазину), LegacyKeys (старі ключі для формування посилання на продукт за його ім’ям на випадок, якщо назва продукту було змінено), (платформи різних ігор), PlatformTypes (платформи на яких були видані ігри), Genres (таблиця яка містить інформацію про категорії продуктів), Publishers (таблиця з поширювачами товару та видавництвами ігор).
- Локалізація інтерфейсу: Localizations (таблиця з наявними у магазині мовами), GenreLocalizations (таблиця з перекладами даних про категорії), PlatformTypeLocalizations (таблиця з перекладами платформ), GamesLocalizations (таблиця з перекладами даних про продукти та ігри).
- Користувацькі данні: Users (таблиця з даними про користувачів), Publishers (таблиця з дистрибьюторами та видавництвами, аккаунт яких можуть використовувати співробітники компанії для керуванням товарами), Orders (замовлення користувачів), OrderDetails (деталі замовлення користувачів).
- NoSQL таблиці відносин: GoodsProducts (таблиця в якій зберігається інформація про продукти які мігрували з NoSQL БД до SQLБД продуктів), GenreCategory (таблиця в якій зберігається представлення нереляційних категорій до реляційних жанрів), PublisherSupplier (таблиця в якій зберігається відношення нереляційних постачальників до реляційних видавництв).

Нижче наведено UML-діаграму структури таблиць програмного продукту (рис. 2.2.):

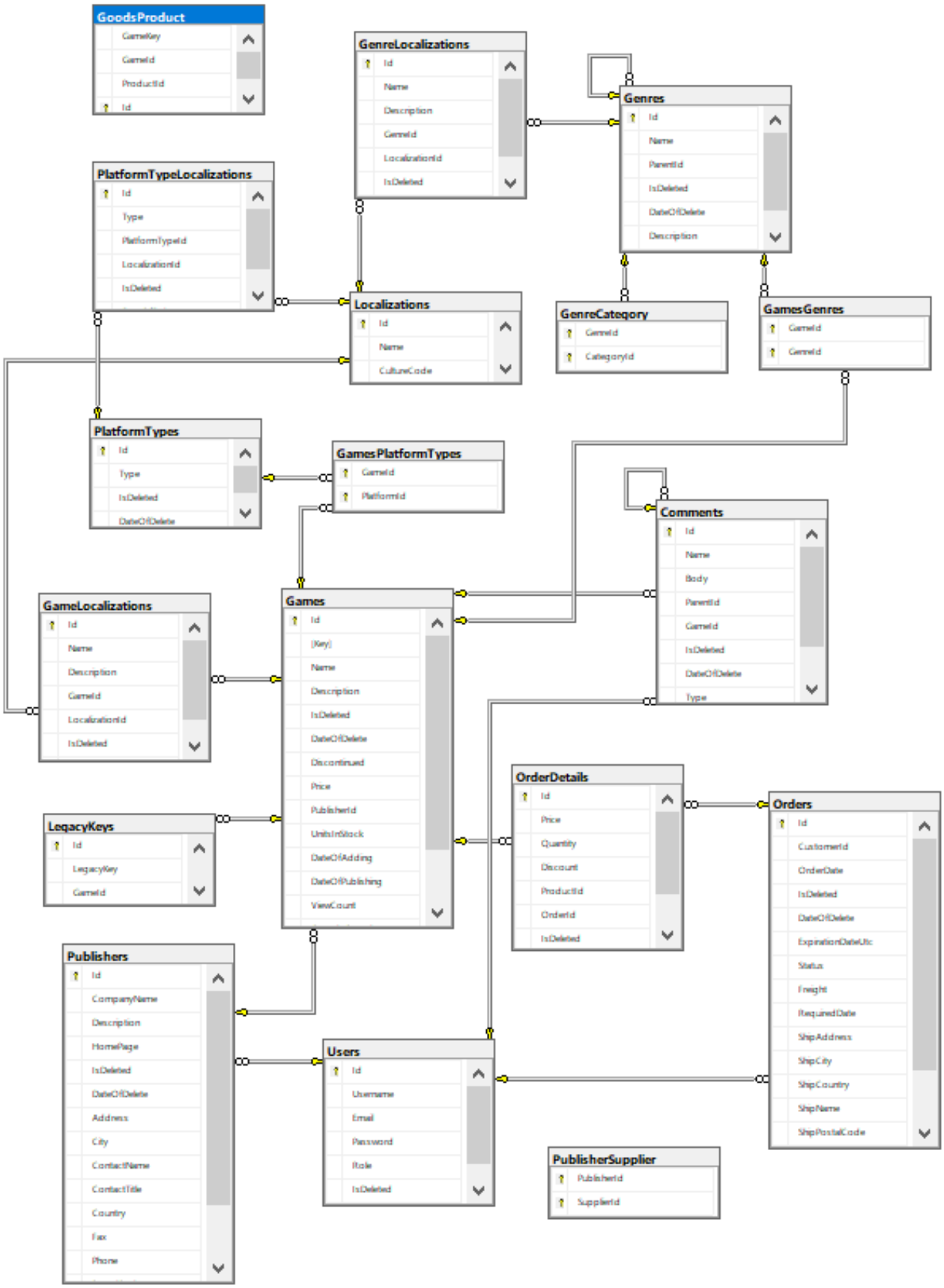

Рис. 2.2. UML-діаграма структури БД

2.6. Обґрунтування та організація вхідних та вихідних даних програми

Вхідними даними інтернет-магазину є завантажені з БД дані, в яких зберігається майже уся інформація про різні сутності додатку, ресурсні файли, в яких зберігається локалізовані версії елементів інтерфейсу, cookie-файли користувача, кеш додатку.

Типовими для додатку є данні, які зберігаються:

- дані про користувачів;
- дані товарів;
- інформація про замовлення користувачів;
- супровідні для товарів дані (постачальники, категорії);
- переклади інформації про товари.

В ресурсних файлах додатку зберігаються словники, які у форматі «ключ-значення» містять переклади для різних текстових елементів інтерфейсу сайту, як приклад можна привести переклади наступних елементів:

- «Order» – «Замовити»;
- «Buy» – «Купити»;
- «Login» – «Увійти»;
- «Registration» – «Зареєструватися».

Згенеровані веб-додатком cookie-файли містять зазвичай файли, які є необхідними для авторизації, збережені неоформлені до кінця товари у кошику, інформація про обрану мову додатку.

2.7. Опис роботи розробленого програмного продукту

2.7.1. Використані технічні засоби

Задля безперебійної роботи програми протягом усієї доби, рекомендується наступна конфігурація серверних технічних засобів:

- процесор класу AMD Ryzen 5 5700x 3.4Ghz/32MB;
- материнська плата Gygabite B550 PRIME;
- модулі пам’яті Kingston 16GB 3666Mhz;
- блок живлення Cheiftec ECO 750W;
- SSD диск Samsung EVO 860 500GB;
- монітор діагоналлю не менше 15 дюймів;
- доступ до мережі Internet не менше 1 мб/с;
- клавіатура;
- комп’ютерна миша.

Технічні характеристики, які були наведені вище, є рекомендованими. Це означає, що для забезпечення відповідності вимогам щодо надійності, швидкості обробки даних і безпеки, встановлені технічні засоби повинні мати як мінімум зазначені параметри.

2.7.2. Використані програмні засоби

Розробка інтернет магазину велася на мові програмування C#, та платформі розробки .NET. Написання коду відбувалося у Visual Studio Community. Visual Studio Community - це безкоштовне, повнофункціональне інтегроване середовище розробки (IDE) від Microsoft. Воно призначене для індивідуальних розробників, проектів з відкритим вихідним кодом, академічних досліджень, освіти та невеликих професійних команд. Visual Studio Community пропонує широкий спектр інструментів та функцій для полегшення розробки програмного забезпечення на різних платформах та мовах програмування.

У якості СКБД було обрано MS SQL Server для роботи з якою було використано Microsoft SQL Server Management Studio, яка є інтегрованим середовищем для управління та адміністрування баз даних Microsoft SQL Server, це інструмент з графічним інтерфейсом користувача (GUI), який надає повний набір функцій і можливостей для взаємодії з SQL Server.

Задля полегшення розробки додатку було використано фреймворк для розробки веб-додатків - ASP.NET, який у якості мови програмування використовує мову програмування C#. Для тестування додатку було застосовано додаток для надіслання веб-запитів до веб-додатків Postman, який є інструментом розробки, що використовується розробниками для спрощення процесу створення, тестування та документування API та який надає інтуїтивно зрозумілий графічний інтерфейс для створення запитів до API, організації та керування запитами, а також аналізу відповідей.

2.7.3. Виклик та завантаження програми

Щоб розгорнути додаток ASP.NET на сервері VPS з окремим сервером бази даних MS SQL, необхідно виконати наступні кроки:

1. Підготувати VPS сервер:

– Налаштувати VPS-сервер із операційною системою Windows або будь-якою Linux-подібною системою.

– Встановити IIS на VPS-сервері для розміщення ASP.NET-додатку.

2. Налаштувати VPS-сервер:

– Налаштувати IIS на підтримку додатків ASP.NET, увімкнувши необхідні функції та розширення.

– Створити веб-сайт в IIS для розміщення ASP.NET-додатку, вказавши відповідні прив'язки (наприклад, доменне ім'я або IP-адресу, номер порту).

3. Опублікувати ASP.NET додаток:

– Створити та опублікувати ASP.NET додаток за допомогою Visual Studio або інструментів командного рядка, таких як dotnet CLI.

– Скопіюйте опубліковані файли програми (наприклад, DLL, конфігураційні файли, статичні файли) у відповідний каталог на сервері VPS, де IIS налаштовано для обслуговування програми.

4. Встановити і налаштувати сервер баз даних MS SQL:

– Встановити окремий сервер бази даних MS SQL на тому ж VPS або на іншому сервері, дотримуючись інструкцій з установки, наданих Microsoft.

– Створити нову базу даних на сервері MS SQL, якщо її ще немає, для використання додатком ASP.NET.

– Змінити рядок підключення, щоб він вказував на окремий сервер бази даних MS SQL, вказавши IP-адресу або ім'я хоста сервера, ім'я бази даних та дані для автентифікації.

5. Протестувати розгортання:

– Отримати доступ до інтернет-магазину через веб-браузер, використовуючи налаштоване доменне ім'я або IP-адресу.

2.7.4. Опис інтерфейсу користувача

Після завантаження сторінки веб-магазину, користувач опиняється на головній сторінці сайту – на сторінці каталогу, де в нього є можливість перейти до різних частин магазину, наприклад:

- сторінка з наявними постачальниками в магазину, аби переглянути каталог їх товарів;
- сторінка з різними категоріями товарів, де він може дізнатись про наявні в цій категорії товари;
- також користувач може увійти до свого акаунту.

На наступному рисунку наведено зовнішній вигляд головного каталогу сайту (рис. 2.3.):

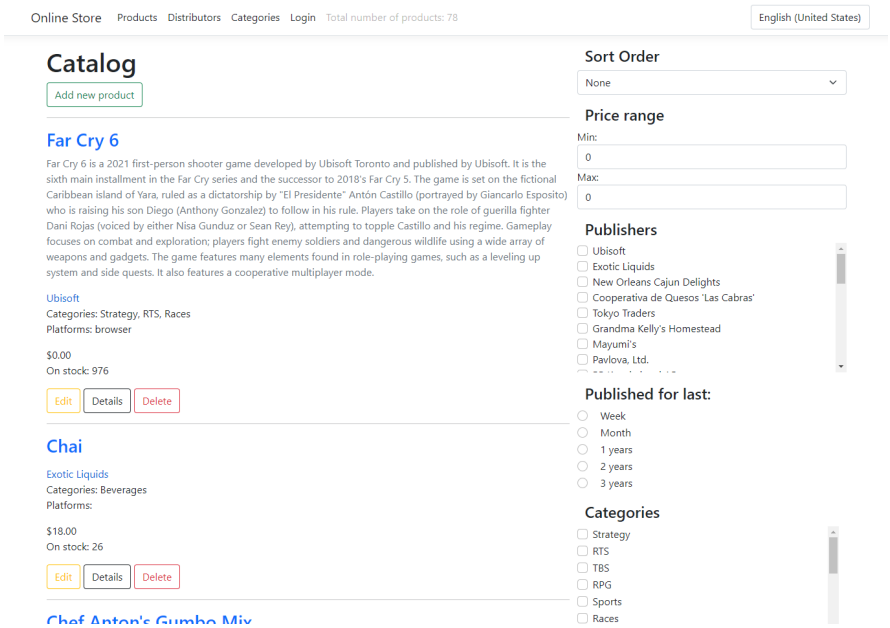

Рис. 2.3. Головна сторінка сайту

Каталог з продуктами надає можливість користувачу застосувати механізми фільтрації та сортування товарів, аби передивитись тільки ті товари, які його могли зацікавити (рис. 2.4.):

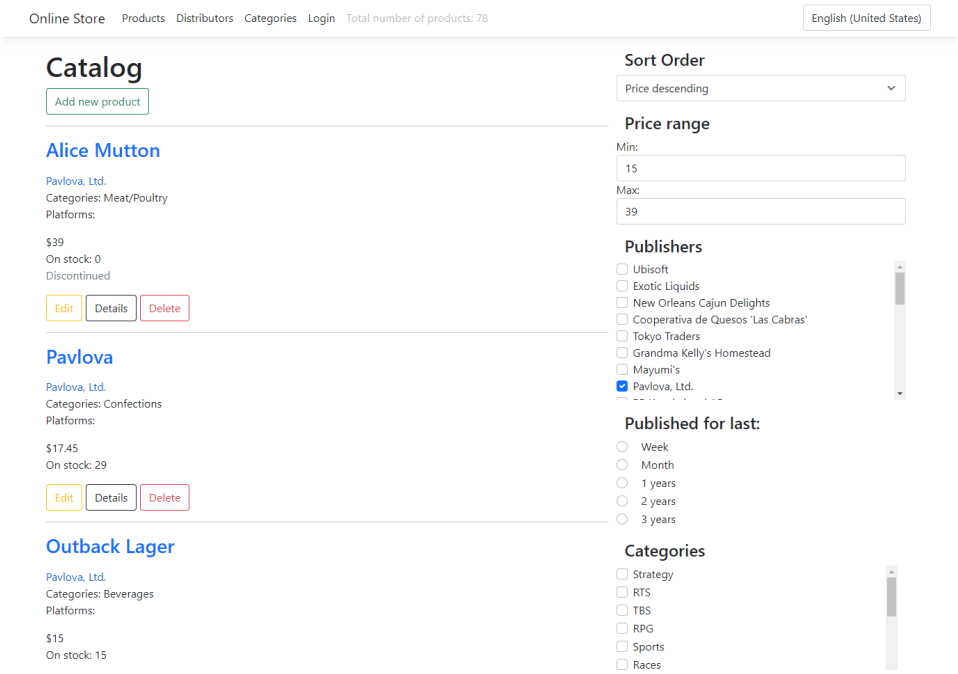

Рис. 2.4. Головна сторінка сайту з включеними фільтрами

Користувач може увійти до свого облікового запису, використовуючи спеціальну форму, в даному випадку ми увійдемо під обліковим записом адміністратора (рис. 2.5.):

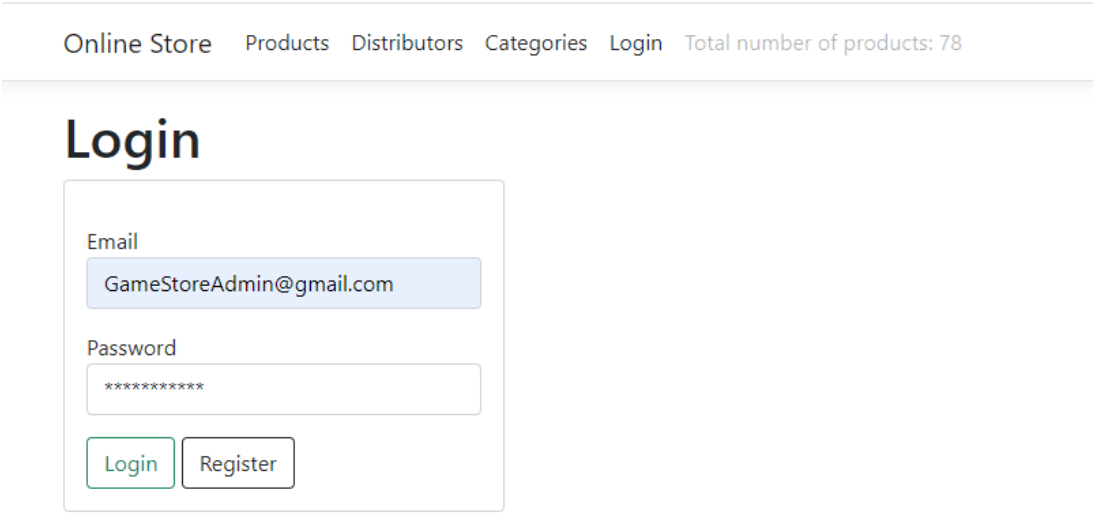

Рис. 2.5. Сторінка логіну

Залежно від типу облікового запису, стають доступними різні частини, в цьому прикладі оскільки ми увійшли як адміністратор магазину – нам доступний увесь функціонал магазину, у тому числі адміністраторські сторінки: сторінка із замовленнями користувачів та сторінка адміністратора (рис. 2.6.).

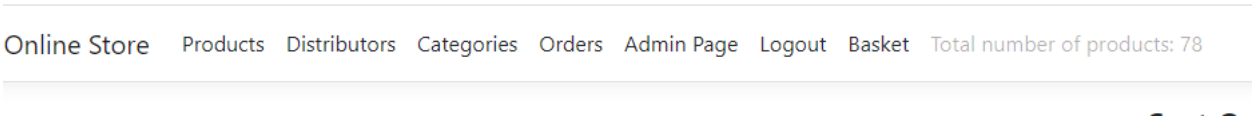

Рис. 2.6. Навігаційна панель з правами адміністратора

До речі сторінка із замовленнями користувачів є доступною не тільки для адміністраторів, а й для користувачів, які мають обліковий запис менеджеру.

Адміністраторська сторінка (рис. 2.7.) дозволяє переглянути користувачів магазину та керувати їх правами у використанні магазину. Також можна змінювати дані користувачів, переглядати детальну інформацію про них та ще акаунті користувачів можна видаляти.

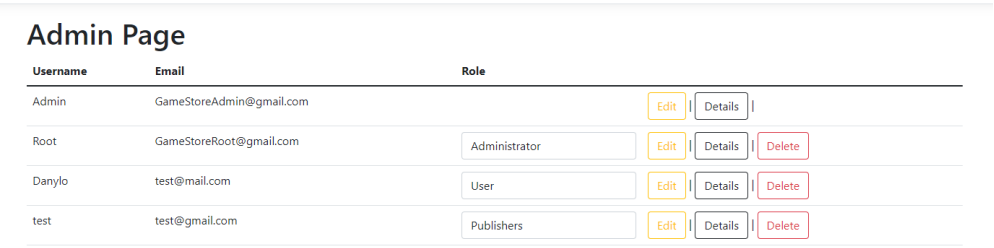

Рис. 2.7. Адміністративна панель

Зайдемо на сторінку з випадковим продуктом та додамо його до свого кошику (рис. 2.8.):

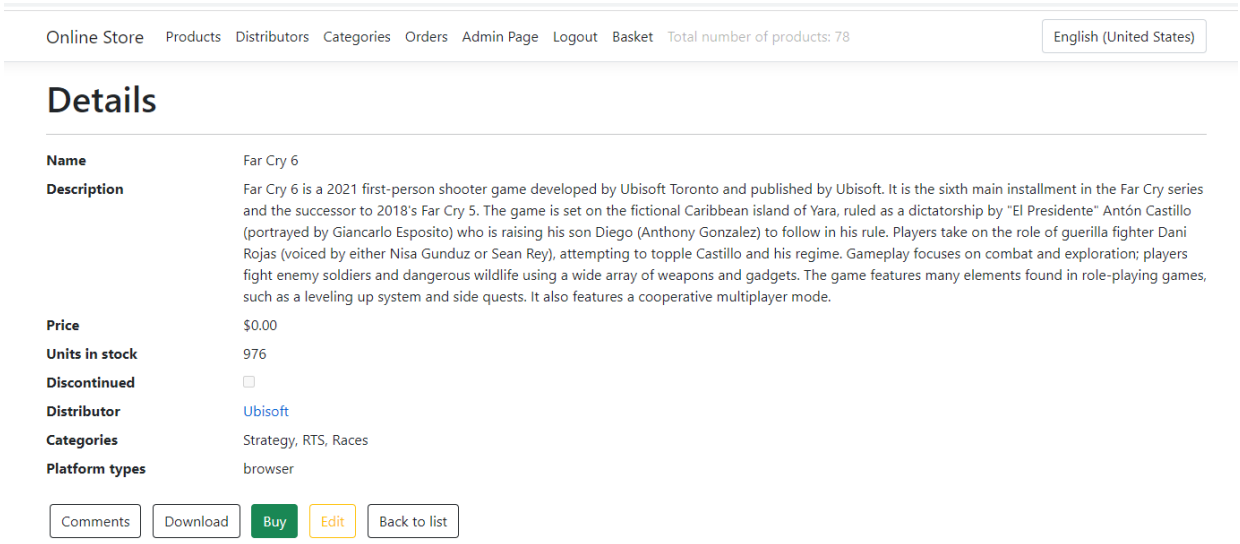

Рис. 2.8. Сторінка деталей продукту

Після натискання на кнопку покупки товару користувач потрапляє на сторінку із замовленнями (для більшої автентичності були куплені додаткові продукти) (рис.2.9.):

Рис. 2.9. Сторінка кошику із замовленням

На наступному кроці нам необхідно заповнити платіжну інформацію та інформацію необхідну для доставки і обрати спосіб доставки (рис. 2.10.):

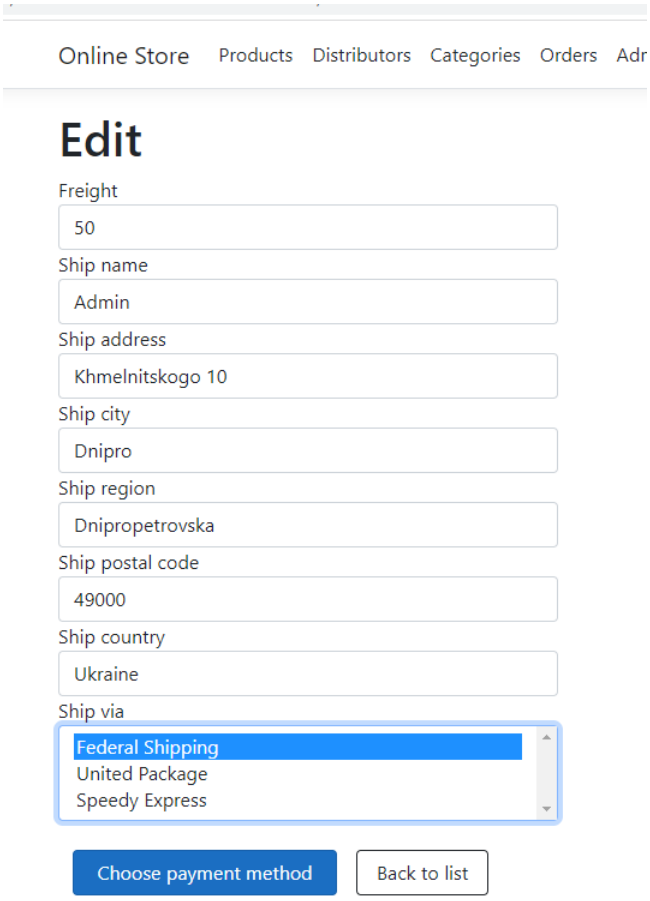

Рис. 2.10. Сторінка для заповнення інформації для доставки замовлення

Після підтвердження адреси її можна зберегти для зручності заповнення у майбутньому (рис. 2.11.):

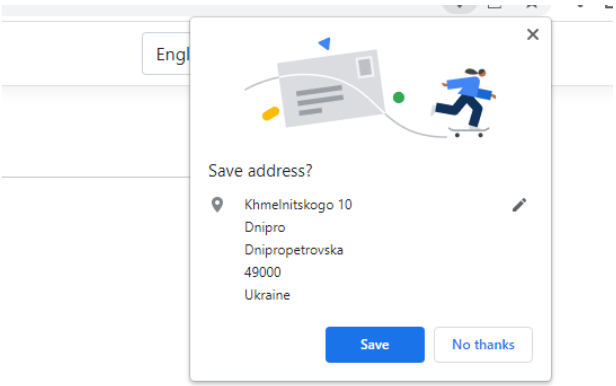

Рис. 2.11. Вікно збереження паролю

На сторінці після заповнення попередньої інформації буде відображено інформацію про замовлення (рис. 2.12.): перелік товарів у замовленні, платіжна інформація, інформація для доставки. На цій сторінці також можна обрати платіжну систему, якою буде сплачено замовлення.

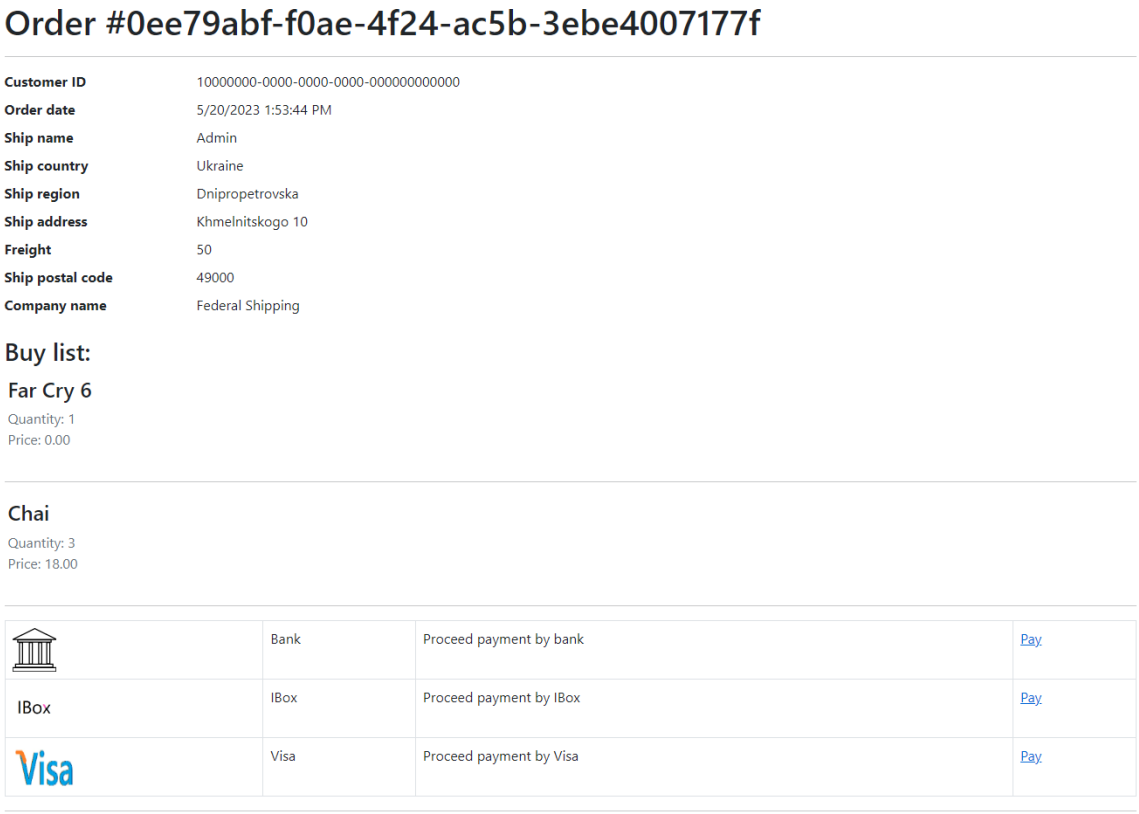

Рис. 2.12. Сторінка перегляду інформації про сформоване замовлення

На даний момент, кількість товарів, які були перелічені у замовленні були зарезервовані для даного покупця і якщо повернутись на сторінку із списком замовлень – можна буде змінювати різні деталі замовлень.

Також, було реалізовано систему відгуків товарів, якщо натиснути на кнопку коментарів, можна буде перейти на сторінку коментарів цього товару (рис. 2.13.). На цій сторінці можна написати новий коментар, написати відповідь на коментар іншого користувача, цитувати якогось користувача (рис. 2.14.), видалити коментар (якщо поточний користувач – адміністратор), заблокувати користувача, та редагувати його коментар.

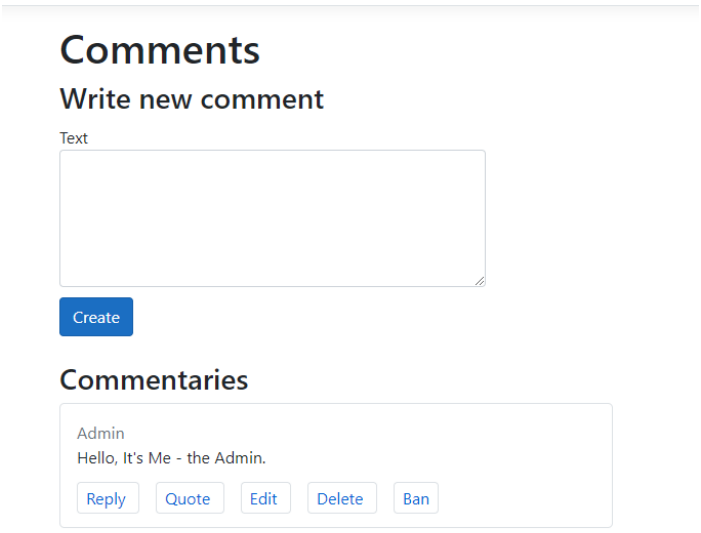

Рис. 2.13. Сторінка з коментарями

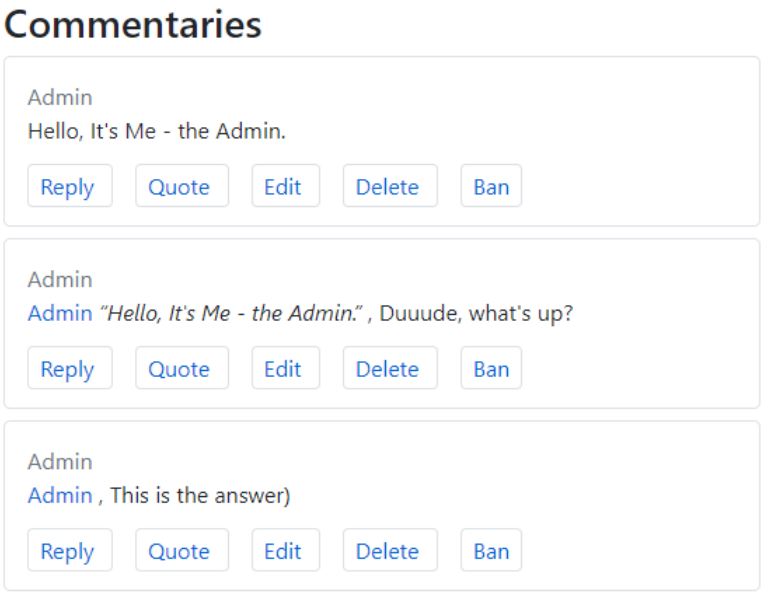

Рис. 2.14. Сторінка із коментарями різного типу

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 1450;
2. коефіцієнт корекції програми в ході її розробки – 0,05;
3. коефіцієнт складності програми – 1,7;
4. годинна заробітна плата програміста– 163 грн/год.

Виходячи з даних «Developers of Ukraine (DOU)» - було вирахувано середню годинну зарплату програміста. Станом на кінець 2022 року заробітна плата Junior Full-Stack .NET розробника простягається від 575$ до 1000$. Вирахувавши середню заробітну плату програміста маємо плату 787,5$. При курсів валют Національного Банку України на початок травня 2023 року один американський долар коштуватиме 36.5 гривень, тому середня заробітна плата дорівнює 28743 грн. При стандартному графіку (176 годин/місяць) зарплата за годину буде становити близько 163 грн.

5. коефіцієнт збільшення витрат праці в наслідок недостатнього опису задачі – 1,4;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,4;
7. вартість машино-години ЕОМ – 22 грн/год.

Оскільки для цього проекту потрібна значна обчислювальна потужність комп'ютера для розміщення бекенду та обробки великого обсягу даних на локальному сервері, рекомендовано взяти комп'ютер в оренду протягом розробки додатку. Вартість оренди комп'ютера складає 1260 грн на місяць за монітор та 2650 грн на місяць за системний блок. Загальна вартість оренди на

місяць становить 3910 грн. При стандартному графіку роботи (176 годин на місяць), витрати на годину роботи комп'ютера складатимуть 22 грн. Ця вартість включає ремонт за гарантією та базовий комплект аксесуарів, таких як клавіатура та миша.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин, (3.1)}$$

де t_o- витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a- витрати праці на розробку блок-схеми алгоритму;

t_n-витрати праці на програмування по готовій блок-схемі;

$t_{отл}$-витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p),$$

де q - передбачуване число операторів (1450);

C - коефіцієнт складності програми (1,7);

p - коефіцієнт корекції програми в ході її розробки (0,05).

Звідси умовне число операторів в програмі:

$$Q = 1{,}7 \cdot 1450 \cdot (1 + 0{,}05) = 2588{,}25$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k} \text{ , людино-годин,}$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи від 5 до 8 років він складає 1,4.

Приймемо збільшення витрат праці внаслідок недостатнього опису завдання не більше 50% (B = 1,2). З урахуванням коефіцієнта кваліфікації k = 1,4, отримуємо витрати праці на вивчення опису завдання:

$$t_u = (2588{,}25 \cdot 1{,}2) / (75 \cdot 1{,}4) = 29{,}58 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20 \ldots 25) \cdot k} \text{ , людино-годин, (3.2)}$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.2), отримаємо:

$$t_a = 2588{,}25 / (20 \cdot 1{,}4) = 92{,}43 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20 \ldots 25) \cdot k} \text{ , людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = 2588{,}25 / (25 \cdot 1{,}4) = 73{,}95 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5) \cdot k} \text{ , людино-годин.}$$

$$t_{отл} = 2588{,}25 / (5 \cdot 1{,}4) = 369{,}75 \text{ людино-годин.}$$

– за умови комплексного налагодження завдання:

$$t^k_{отл} = 1{,}5 \cdot t_{отл} \text{ , людино-годин.}$$

$$t^k_{отл} = 1{,}5 \cdot 369{,}75 = 554{,}6 \text{ людино-годин.}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_д = t_{др} + t_{до}\ ,\ \text{людино-годин},$$

де $t_{др}$-трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15..20) \cdot k}\ ,\ \text{людино-годин},$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 \cdot t_{др}\ ,\ \text{людино-годин}.$$

Підставляючи відповідні значення, отримаємо:

$t_{др}$ = 2588,25 / (18 · 1,4) = 102,7 людино-годин.

$t_{до}$ = 0,75 · 102,7= 77 людино-годин.

$t_д$ = 102,7 + 77 = 179,7 людино-годин.

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

t = 50 + 29,58 + 92,43 + 73,95 + 369,75 + 179,7 = 795,41 людино-години.

3.2. Рахунок витрат на створення програми

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн.}$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,}$$

де: t - загальна трудомісткість, людино-годин;

$С_{ПР}$ - середня годинна заробітна плата програміста, грн/година

З урахуванням того, що середня годинна зарплата програміста становить 163 грн / год, отримуємо:

$$З_{ЗП} = 795{,}41 \cdot 163 = 129651 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мв} = t_{отл} \cdot С_{мч}, \text{ грн, (3.3)}$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$С_{мч}$ - вартість машино-години ЕОМ, грн/год (22 грн/год).

Підставивши в формулу (3.3) відповідні значення, визначимо вартість необхідного для налагодження машинного часу:

$$З_{мв} = 369{,}75 \cdot 22 = 8\ 134{,}5 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 129\ 651 + 8\ 134{,}5 = 137\ 785{,}5 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс.}$$

де B_k - число виконавців (дорівнює 1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин).

Період створення ПЗ:

$$T = 795{,}41 / 1 \cdot 176 \approx 4{,}5 \text{ міс.}$$

Висновок: розроблене програмне забезпечення є інтернет-магазином, спеціалізованим на роздрібній торгівлі. Його основною метою є надання зручних та ефективних можливостей для замовлення та придбання товарів через Інтернет. Вартість цього програмного забезпечення становить 137 785,5 грн. Планується, що розробка займе близько 4,5 місяців. Цей термін обумовлений великою кількістю завдань, включаючи дослідження та розробку алгоритму для досягнення поставленої мети, програмування на основі цього алгоритму, відладку програми та підготовку необхідної документації.

ВИСНОВКИ

Під час цієї кваліфікаційної роботи, було спроектовано та розроблено інтернет-магазин згідно поставленого завдання.

Дане програмне забезпечення призначене для надання зручної платформи для постановки та реалізації торгівельних відносин бізнесу та покупців у вигляді e-commerce та моделі B2C.

Практичне призначення даного додатку полягає в надання можливості відвідувачам інтернет-магазину комфортного доступу до каталогу товарів, які можна фільтрувати в залежності від своїх потреб, для підприємців – інтернет-магазин надає прості та швидкі методи для додавання до каталогу нових товарів та можливість надання глобалізації для своїх продуктів за допомогою написання перекладів для описів товару.

Структура програми являє собою багаторівневу реалізацію архітектури MVC, тобто це клієнт-серверний додаток, який в собі містить механізми як низького рівня (взаємодія з базами даних), різні сервіси з бізнес-логікою, так і механізми необхідні для роботи клієнту додатку в браузері. Наприклад: механізм маршрутизації, шаблони представлень, засоби отримання і серіалізації даних від користувача, механізми прив’язки отриманих даних та абстрактних моделей об’єктно-орієнтованої парадигми програмування і тощо.

Інтернет-магазин був розроблений на базі фреймворку ASP.NET та мови програмування високого рівня C#. У якості системи керування баз даних було використано СКБД Microsoft SQL Server і для взаємодії з нею на абстрактному рівні додатку було застосовано бібліотеку для об’єктно-реляційного проектування, яка дозволила проектування дані з таблиць БД на об’єкті моделей C#. Для роботи з клієнтською частиною було використано двигун представлень Razor, HTML, Bootstrap, CSS та JS.

В «Економічному розділі» було також визначено трудомісткість розробки програмного забезпечення (795 чол-год), підраховані витрати на

створення програмного забезпечення (137 785 грн) і очікуваний період розробки (4.5 місяці).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Microsoft: ASP.NET overview | URL: https://learn.microsoft.com/en-us/aspnet/overview (дата звернення: 18.01.23)

2. Kevin Hoffman: Building Microservices with ASP.NET Core 1st Edition 2017. 232 c.

3. Microsoft: Getting started with ASP.NET MVC 5 | URL: https://learn.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/getting-started (дата звернення: 19.01.23)

4. Juan Gonzalez: Getting started with ASP.NET Razor | URL: https://medium.com/@gonzalezjuan.ab/getting-started-with-asp-net-razor-32d15a9cf93d (дата звернення 20.01.23)

5. Joseph Albahari: C# 10 in a Nutshell. The Definitive Reference 2022. 1058c.

6. Stephen Clearly: Concurrency in C# Cookbook: Asynchronous, Parallel, and Multithreaded Programming 2nd Edition 2019, 251 p.

7. Jon P Smith: Entity Framework Core in Action 2018, 520 p.

8. Alan Beaulieu: Learning SQL: Master SQL Fundamentals 3rd Edition, 2020 380c.

9. Toby J. Teorey: Database Design: Know It All, 2008, 368c.

10. Microsoft: Explicitly Tracking Entities | URL: https://learn.microsoft.com/en-us/ef/core/change-tracking/explicit-tracking (дата звернення 21.01.23)

11. Microsoft: Simple Logging | URL: https://learn.microsoft.com/en-us/ef/core/logging-events-diagnostics/simple-logging (дата звернення 25.01.23)

12. Microsoft: Tutorial: Update related data – ASP.NET MVC with EF Core | URL: https://learn.microsoft.com/en-us/aspnet/core/data/ef-mvc/update-related-data?view=aspnetcore-7.0 (дата звернення 28.01.23)

13. Microsoft: Make an ASP.NET Core app’s content localizable | URL: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/localization/make-content-localizable?view=aspnetcore-7.0 (дата звернення 29.01.23)

14. Jennifer Niederst Robbins: Web Design in Nutshell, 2009, 830c.

15. Keith Grant: CSS in Depth, 2018, 472 c.

16. Jake Spurlock: Bootstrap: Responsive Web Development, 2013, 128 c.

17. Bootstrap: Quick start | URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/#quick-start (дата звернення 30.01.23)

18. Довідник по загальним HTML атрибутам | URL: https://css.in.ua/html/global/attributes (дата звернення 01.02.23)

19. Microsoft: SQL tools overview | URL: https://learn.microsoft.com/en-us/sql/tools/overview-sql-tools?view=sql-server-ver16 (дата звернення 03.02.23)

20. xUnit: Samples | URL: https://github.com/xunit/samples.xunit (дата звернення 04.02.23)

21. Nicklas Millard: Get a Basic Understanding of Authentication | URL: https://nmillard.medium.com/brief-overview-of-asp-net-core-authentication-451e630bc42d (дата звернення 06.02.23)

22. Oskar Berggren: Returning a file to View/Download in ASP.NET MVC | URL: https://stackoverflow.com/questions/5826649/returning-a-file-to-view-download-in-asp-net-mvc (дата звернення 08.02.23)

23. Serilog: Configuration Basics | URL: https://github.com/serilog/serilog/wiki/Configuration-Basics (дата звернення 10.02.23)

24. Serilog: Formatting Output | URL: https://github.com/serilog/serilog/wiki/Formatting-Output (дата звернення 13.02.23)

КОД ПРОГРАМИ

Startup.cs

```
using Autofac;
using GameStore.BLL.MappingProfiles;
using GameStore.DAL.Context;
using GameStore.DAL.MappingProfiles;
using GameStore.PL.Configurations;
using GameStore.PL.DependencyInjections;
using GameStore.PL.Extensions;
using GameStore.PL.HostedServices;
using GameStore.PL.MappingProfiles;
using GameStore.PL.Middlewares;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Localization;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using System.Reflection;

namespace GameStore.PL
{
    public class Startup
    {
        public IConfiguration Configuration { get; }

        public IServiceCollection Services { get; set; }

        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public void ConfigureServices(IServiceCollection services)
        {
            var appSettings = Configuration.Get<ApplicationSettings>();

            services.AddSingleton(appSettings.DownloadingFileSettings);
            services.AddSingleton(appSettings.MongoIntegrationSettings);
            services.AddSingleton(appSettings.GuestCookieSettings);

            services.Configure<ApplicationSettings>(Configuration);

            services
                .AddMongoDb(appSettings.MongoDbSettings)
                .ConfigureNorthwindDatabase();

            services
                .AddDbContext<GameStoreContext>(options =>
options.UseSqlServer(appSettings.ConnectionStrings.GameStoreSQLEXPRESS))
                .AddScoped<DbContext, GameStoreContext>();

            services.AddAutoMapper(
                typeof(EntityDomainMapperProfile),
                typeof(PresentationLayerMapperProfile),
                typeof(BusinessMappingProfile));
```

```
            services
                .AddHostedService<LaunchCacheService>()
                .AddHostedService<PaymentTimeOutService>();

            services
                .AddMemoryCache()
                .AddResponseCaching();


services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
                .AddCookie(options =>
                {
                    options.LoginPath = new
Microsoft.AspNetCore.Http.PathString("/User/Login");
                    options.AccessDeniedPath = new
Microsoft.AspNetCore.Http.PathString("/User/access-denied");
                });

            services.AddLocalization(options => options.ResourcesPath = "Resources");

            services.Configure<RequestLocalizationOptions>(options =>
            {
                options.SetDefaultCulture("en-US");

                options.AddSupportedCultures("uk-UA", "ru-RU", "en-US");
                options.AddSupportedUICultures("uk-UA", "ru-RU", "en-US");
                options.FallBackToParentUICultures = true;

                options
                    .RequestCultureProviders
                    .Remove(typeof(AcceptLanguageHeaderRequestCultureProvider));
            });

            services.AddMvc()
                .AddRazorRuntimeCompilation()
                .AddDataAnnotationsLocalization()
                .AddViewLocalization()
                .AddNewtonsoftJson(options =>
                {
                    options.SerializerSettings.ReferenceLoopHandling =
Newtonsoft.Json.ReferenceLoopHandling.Ignore;
                });

            Services = services;
        }

        public void ConfigureContainer(ContainerBuilder builder)
        {
            Assembly[] gameStoreAssemblies =
GameStoreAssembliesCollector.CollectAssemblies();
            DependencyInjectionConfigurator.RegisterDependencies(builder,
gameStoreAssemblies, Services);
        }

        // This method gets called by the runtime. Use this method to configure the
HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
```

```
            }
            else
            {
                app.UseExceptionHandler("/Shared/Errors/Error");
                app.UseHsts();
            }

            app.UseRequestLocalization();

            app
                .UseMiddleware<RequestLocalizationCookiesMiddleware>()
                .UseMiddleware<RequestLoggingMiddleware>()
                .UseMiddleware<ErrorHandlingMiddleware>();

            app.UseHttpsRedirection();
            app.UseStaticFiles();

            app.UseRouting();

            app.UseResponseCaching();

            app.UseAuthentication();
            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                       name: "default",
                       pattern: "{controller=Games}/{action=GetAllGames}/{id?}");
            });
        }
    }
}
```

Program.cs

```
using Autofac.Extensions.DependencyInjection;
using GameStore.PL.Configurations;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace GameStore.PL
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .UseServiceProviderFactory(new AutofacServiceProviderFactory())
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                })
                .ConfigureLogging((context, logging) =>
                {
                    logging.ClearProviders();
                    logging.AddGenericLogging(context);
                });
```

```
    }
}
```

Index.cshtml

```
@model GameStore.PL.ViewContexts.GamesViewsContext

@using GameStore.PL.App_Code
@using GameStore.PL.DTOs
@using GameStore.PL.Configurations
@using GameStore.DomainModels.Enums
@using GameStore.PL.Util
@using GameStore.PL.Util.Localizers
@using Microsoft.AspNetCore.Mvc.Localization
@using Microsoft.Extensions.Options

@inject IOptions<ApplicationSettings> ApplicationSettings

@inject IHtmlLocalizer<SharedLocalizer> SharedLocalizer
@inject IViewLocalizer Localizer

@{
    ViewData["Title"] = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
    var publishingDateOptions =
ApplicationSettings.Value.DateOfPublishingFilterSettings.OldestPublishingDateFilterOp
tions.OrderBy(x => x.Value);
    var itemsPerPageOptions =
ApplicationSettings.Value.GamesPerPageSettings.ItemsPerPageOptions.OrderBy(x =>
x.Value);
}

<div class="row container">
    <div class="col-8 container">
        <div>
            <h1>@Localizer["Catalog"]</h1>
            <p>
                <a asp-action="GetCreationView" class="btn btn-outline-
success">@Localizer["CreateNewGame"]</a>
                <hr />
            </p>
        </div>

        <div class="games-list">
            @foreach (var game in Model.Games)
            {
                <div class="gameContainer">
                    <h3 class="text-primary">
                        @(FieldLocalizer.GetLocalizedField<GoodsDTO>(g => g.Name,
game))
                    </h3>
                    <div>
                        <p class="text-secondary">
                            @(FieldLocalizer.GetLocalizedField<GoodsDTO>(g =>
g.Description, game))
                        </p>
                    </div>
                    <div>
                        @if (string.IsNullOrEmpty(game.Distributor))
                        {
```

```
                            <span class="text-muted">@Localizer["Uknown"]</span>
                        }
                        else
                        {
                            <a asp-controller="publisher" asp-action="Details" asp-
route-name="@game.Distributor" class="text-decoration-none">@game.Distributor</a>
                        }
                        <p>
                            @Localizer["Genres"]:
                            @if (game.Genres.Any())
                            {
                                @ListHelper.JoinListIntoOneString(Html, game.Genres,
genre => genre.Name, ", ")
                            }
                            else
                            {
                                <span class="text-muted">@Localizer["Other"]</span>
                            }
                            <br>
                            @Localizer["Platforms"]:
@ListHelper.JoinListIntoOneString(Html, game.PlatformTypes, type => type.Type, ", ")
                        </p>
                        <p>
                            <span>$@game.Price</span> <br>
                            @Localizer["KeysLeft"]: @game.UnitsInStock <br>
                            @if (@game.Discontinued == true)
                            {
                                <span class="text-
muted">@Localizer["Discontinued"]</span>
                            }
                        </p>
                    </div>

                    <div>
                        <a asp-action="GetEditingView" asp-route-key="@game.Key"
class="btn btn-outline-warning">@SharedLocalizer["Edit"]</a>
                        <a asp-controller="game" asp-action="DetailsByKey" asp-route-
key="@game.Key" class="btn btn-outline-dark">@SharedLocalizer["Details"]</a>
                        <a asp-action="GetDeletingView" asp-route-key="@game.Key"
class="btn btn-outline-danger">@SharedLocalizer["Delete"]</a>
                    </div>
                    <hr />
                </div>
            }

            @if (Model.Filters.ItemsPerPage != 0)
            {
                <pagination></pagination>
            }

        </div>
    </div>

    <div class="col container filterForm">
        <form asp-action="Index" name="filterForm" onsubmit="setupCurrentPageValue()"
method="get">
            <div>
                <h4>@Localizer["SortOrder"]</h4>
                <div class="row">
                    <select name="SortBy" class="form-select">
                        @foreach (GamesSortType sort in
Enum.GetValues(typeof(GamesSortType)))
```

```
                        {
                            @if (Model.Filters.SortBy == sort)
                            {
                                <option selected
value="@sort">@Localizer[sort.ToString()]</option>
                            }
                            else
                            {
                                <option
value="@sort">@Localizer[sort.ToString()]</option>
                            }
                        }
                    </select>
                </div>

                <div class="mt-3">
                    <h4>@Localizer["PriceRange"]</h4>
                    <div class="row ">
                        @Localizer["Min"]:
                        <input type="number" min="0" class="form-control"
name="MinPrice" id="MinPrice" value="@Model.Filters.MinPrice" />

                        @Localizer["Max"]:
                        <input type="number" min="0" class="form-control"
name="MaxPrice" id="MaxPrice" value="@Model.Filters.MaxPrice" />
                    </div>
                </div>

                <div class="row mt-3">
                    <h4>@Localizer["Publishers"]</h4>

                    <div class="form-check overflow-auto" style="height:200px">
                        @foreach (var publisherFilter in
Model.PublishersFilterOptions)
                        {
                            <div class=col>
                                @if (Model.Filters.Distributors.Any(x => x ==
publisherFilter.Id))
                                {
                                    <input type="checkbox"
id="@publisherFilter.CompanyName" class="form-check-input" name="Distributors"
checked="checked" value="@publisherFilter.Id" />
                                }
                                else
                                {
                                    <input type="checkbox"
id="@publisherFilter.CompanyName" class="form-check-input" name="Distributors"
value="@publisherFilter.Id" />
                                }
                                <label class="form-check-label"
for="Publishers">@publisherFilter.CompanyName</label>
                            </div>
                        }
                    </div>
                </div>

                <div class="row mt-3">
                    <h4>@Localizer["PublishedFor"]:</h4>

                    <div>
                        @foreach (var date in publishingDateOptions)
                        {
```

```
                            <div class="row form-check">
                                @if (Model.Filters.CountOfDaysBeforePublishingDate ==
date.Value)
                                {
                                    <input class="p-0 form-check-input"
name="CountOfDaysBeforePublishingDate" type="radio" id="@date.Key"
value="@date.Value" checked="checked" />
                                }
                                else
                                {
                                    <input class="p-0 form-check-input"
name="CountOfDaysBeforePublishingDate" type="radio" id="@date.Key"
value="@date.Value" />
                                }
                                <label class="mx-0 form-check-label"
for="@date.Key">@Localizer[date.Key.ToString()]</label>
                            </div>
                        }
                    </div>
                </div>

                <div class="row mt-3">
                    <h4>@Localizer["Genres"]</h4>

                    <div class="row justify-content-center row-cols-1 overflow-auto"
style="height:200px">
                        @foreach (var genreFilter in Model.GenresFilterOptions)
                        {
                            <div class="col form-check">
                                @if (Model.Filters.Genres.Any(x => x ==
genreFilter.Id))
                                {
                                    <input class="col-1 form-check-input"
                                   type="checkbox"
                                   id="@genreFilter.Name"
                                   name="Genres"
                                   value="@genreFilter.Id"
                                   checked="checked" />
                                }
                                else
                                {
                                    <input class="col-1 form-check-input"
type="checkbox" id="@genreFilter.Name" name="Genres" value="@genreFilter.Id" />
                                }
                                <label class="col-5 form-check-label"
for="Genres">@(FieldLocalizer.GetLocalizedField(x => x.Name, genreFilter))</label>
                            </div>
                        }
                    </div>
                </div>

                <div class="row mt-3">
                    <h4>@Localizer["Platforms"]</h4>

                    <div class="overflow-auto" style="height:200px">
                        @foreach (var platformFilter in Model.PlatformsFilterOptions)
                        {
                            <div class=col form-check>
                                @if (Model.Filters.PlatformTypes.Any(x => x ==
platformFilter.Id))
                                {
```

```
                                    <input type="checkbox" class="form-check-input"
id="@platformFilter.Type" name="PlatformTypes" checked="checked"
value="@platformFilter.Id" />
                                }
                                else
                                {
                                    <input type="checkbox" class="form-check-input"
id="@platformFilter.Type" name="PlatformTypes" value="@platformFilter.Id" />
                                }
                                <label class="form-check-label"
for="PlatformTypes">@(FieldLocalizer.GetLocalizedField(x => x.Type,
platformFilter))</label>
                            </div>
                        }
                    </div>
                </div>

                <div>
                    <h4>@Localizer["ItemsPerPage"]</h4>
                    <div class="row mt-3">
                        <select class="form-select" name="ItemsPerPage">
                            @foreach (var itemsPerPageOption in itemsPerPageOptions)
                            {
                                @if (Model.Filters.ItemsPerPage ==
itemsPerPageOption.Value)
                                {
                                    <option selected
value="@itemsPerPageOption.Value">@Localizer[@itemsPerPageOption.Key]</option>
                                }
                                else
                                {
                                    <option
value="@itemsPerPageOption.Value">@Localizer[@itemsPerPageOption.Key]</option>
                                }
                            }
                        </select>
                    </div>
                </div>

                <div class="row mt-3">
                    <h4>@Localizer["SearchByName"]</h4>
                    <input class="form-control" type="text" name="Name" id="Name"
value="@Model.Filters.Name" />
                </div>

                <input hidden name="CurrentPage" value="@Model.Filters.CurrentPage"
/>

                <div class="row mt-3">
                    <input class="btn btn-outline-secondary col-3 mx-3" type="reset"
value="@SharedLocalizer["Clear"]" />
                    <input class="btn btn-outline-success col-3" type="submit"
value="@SharedLocalizer["Apply"]" />
                </div>
            </div>
        </form>
    </div>
</div>
```

GamesController.cs

```
using AutoMapper;
using GameStore.BLL.Interfaces;
```

```
using GameStore.DomainModels.Enums;
using GameStore.DomainModels.Exceptions;
using GameStore.DomainModels.Models;
using GameStore.DomainModels.Models.CreateModels;
using GameStore.DomainModels.Models.EditModels;
using GameStore.PL.DTOs;
using GameStore.PL.DTOs.EditDTOs;
using GameStore.PL.Factories.Interfaces;
using GameStore.PL.Filters;
using GameStore.PL.Util.Authorization;
using GameStore.PL.ViewContexts;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Threading.Tasks;

namespace GameStore.PL.Controllers
{
    [Route("[controller]")]
    public class GamesController : Controller
    {
        private readonly IGoodsService _gameService;

        private readonly IMapper _mapper;
        private readonly ILogger<GamesController> _logger;
        private readonly IGameContextFactory _contextFactory;

        private string CurrentLanguage => CultureInfo.CurrentCulture.Name;

        public GamesController(
            IGoodsService gameService,
            IMapper mapper,
            ILogger<GamesController> logger,
            IGameContextFactory contextFactory)
        {
            _gameService = gameService;
            _mapper = mapper;
            _logger = logger;
            _contextFactory = contextFactory;
        }

        [HttpGet]
        public async Task<IActionResult> IndexAsync([FromQuery] GoodsSearchRequestDTO
filters)
        {
            GoodsSearchRequest gamefilters =
_mapper.Map<GoodsSearchRequest>(filters);

            var filteredGames = await _gameService.GetFilteredGoodsAsync(gamefilters,
CurrentLanguage);

            List<GoodsDTO> filteredGamesDto =
_mapper.Map<List<GoodsDTO>>(filteredGames);
            var context = await
_contextFactory.BuildGamesViewContextAsync(filteredGamesDto, CurrentLanguage);

            context.Filters = filters;

            _logger.LogDebug("Returned game list: {@GamesDTOList}", context.Games);
```

```
            return View("Index", context);
        }

        [PermissionLevel(UserRoles.Manager)]
        [HttpGet("new")]
        public async Task<IActionResult> GetCreationViewAsync()
        {
            GameCreationViewContext context = await
_contextFactory.BuildGameCreationContextAsync();
            return View("Create", context);
        }

        [PermissionLevel(UserRoles.Manager)]
        [HttpPost("new")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> CreateAsync(GameCreationViewContext
gameToCreate)
        {
            CreateGoodsRequest createRequest =
_mapper.Map<CreateGoodsRequest>(gameToCreate.GameCreateDTO);

            string gameKey;
            try
            {
                gameKey = await _gameService.CreateGoodsAsync(createRequest);
            }
            catch (NotUniqueException ex)
            {
                ModelState.AddModelError("Key", ex.Message);
                gameToCreate = await _contextFactory.BuildGameCreationContextAsync();
                return View("Create", gameToCreate);
            }

            _logger.LogDebug("Created game: {@CreatedBySaveDTO}.", gameToCreate);

            return RedirectToAction("DetailsByKey", "Game", new { key = gameKey });
        }

        [PermissionLevel(UserRoles.Publishers)]
        [HttpGet("update/{key}")]
        [ArgumentDecoderFilter("key")]
        public async Task<IActionResult> GetEditingViewAsync(string key)
        {
            Goods foundGame = await
_gameService.GetGoodsByKeyIncludeAllLocalizationsAsync(key);

            Guid? publisherId = null;
            if (foundGame.Distributor != null)
            {
                publisherId = foundGame.Distributor.UserId;
            }

            if (!HierarchicalValidator.IsUserManagerOrOwner(publisherId, User))
            {
                return Forbid();
            }

            var gameToEdit = _mapper.Map<EditGoodsRequestDTO>(foundGame);
            GameEditingViewContext gameEditContext = await
_contextFactory.BuildGameEditingContextAsync(gameToEdit);

            return View("Edit", gameEditContext);
```

```
        }

        [PermissionLevel(UserRoles.Publishers)]
        [HttpPost("update")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> EditAsync(GameEditingViewContext context)
        {
            if
(!HierarchicalValidator.IsUserManagerOrOwner(context.DistributorUserId, User))
            {
                return Forbid();
            }

            EditGoodsRequest editRequest =
_mapper.Map<EditGoodsRequest>(context.GameEditDTO);
            try
            {
                await _gameService.EditGoodsAsync(editRequest);
            }
            catch (NotUniqueException ex)
            {
                ModelState.AddModelError("Key", ex.Message);
                context = await
_contextFactory.BuildGameEditingContextAsync(context.GameEditDTO);
                return View("Edit", context);
            }

            return RedirectToAction("Index");
        }

        [PermissionLevel(UserRoles.Manager)]
        [HttpGet("remove/{key}")]
        [ArgumentDecoderFilter("key")]
        public async Task<IActionResult> GetDeletingViewAsync(string key)
        {
            Goods foundedGame = await _gameService.GetGoodsByKeyAsync(key);

            GoodsDTO gameToDelete = _mapper.Map<GoodsDTO>(foundedGame);
            return View("Delete", gameToDelete);
        }

        [PermissionLevel(UserRoles.Manager)]
        [HttpPost("remove")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteAsync(string id)
        {
            await _gameService.SoftDeleteGoodsAsync(id);
            return RedirectToAction("Index");
        }
    }
}
```

Goods.cs

```
using GameStore.DomainModels.Models.Localizations;
using System;
using System.Collections.Generic;

namespace GameStore.DomainModels.Models
{
    public class Goods
    {
```

```
        public string Id { get; set; }

        public string Key { get; set; }

        public string Name { get; set; }

        public string Description { get; set; }

        public decimal Price { get; set; }

        public short UnitsInStock { get; set; }

        public bool Discontinued { get; set; }

        public string QuantityPerUnit { get; set; }

        public int UnitsInOrder { get; set; }

        public int ReorderLevel { get; set; }

        public DateTime? DateOfPublishing { get; set; }

        public DateTime DateOfAdding { get; set; }

        public long ViewCount { get; set; }

        public Distributor Distributor { get; set; }

        public string DistributorId { get; set; }

        public List<Comment> Comments { get; set; } = new List<Comment>();

        public List<PlatformType> PlatformTypes { get; set; } = new
List<PlatformType>();

        public List<Genre> Genres { get; set; } = new List<Genre>();

        public List<GoodsLocalization> Localizations { get; set; }
    }
}
```

IGoodsProductRepository.cs

```
using GameStore.DomainModels.Models;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace GameStore.DAL.Abstractions.Interfaces
{
    public interface IGoodsProductRepository
    {
        Task DeleteRangeAsync(List<GoodsProductMapping> goodsProducts);

        Task<GoodsProductMapping> GetByGameIdAsync(Guid id);

        Task<GoodsProductMapping> GetByGameKeyAsync(string gameKey);

        Task<List<GoodsProductMapping>> GetAllAsync();

        Task UpdateAsync(GoodsProductMapping mapping);

        Task CreateAsync(GoodsProductMapping newMapping);
```

```
    }
}
```

GoodsRepository.cs

```
using AutoMapper;
using GameStore.DAL.Abstractions.Interfaces;
using GameStore.DAL.Entities;
using GameStore.DAL.Entities.Localizations;
using GameStore.DAL.Entities.MongoEntities;
using GameStore.DAL.Repositories.MongoDbRepositories.Interfaces;
using GameStore.DAL.Repositories.MsSqlDdRepositories.Interfaces;
using GameStore.DAL.SearchPipelines.GoodsSearchPipeline.Interfaces;
using GameStore.DAL.Settings;
using GameStore.DAL.Util.CacheManagers.Interfaces;
using GameStore.DAL.Workflows.Interfaces;
using GameStore.DomainModels.Exceptions;
using GameStore.DomainModels.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace GameStore.DAL.Repositories
{
    public class GoodsRepository : IGoodsRepository
    {
        private readonly IGameRepository _gameRepository;
        private readonly IProductMongoRepository _productRepository;
        private readonly ISupplierMongoRepository _supplierMongoRepository;
        private readonly ILegacyKeyRepository _legacyKeyRepository;
        private readonly IPublisherRepository _publisherRepository;

        private readonly IPublisherSupplierMappingWorkflow
_publisherSupplierMappingWorkflow;

        private readonly IMapper _mapper;
        private readonly IGoodsSearchPipeline _goodsSearchPipeline;
        private readonly ICacheManager _cacheManager;

        private readonly MongoIntegrationSettings _mongoIntegrationSettings;

        public GoodsRepository(
            IGameRepository gameRepository,
            IProductMongoRepository productRepository,
            ISupplierMongoRepository supplierMongoRepository,
            ILegacyKeyRepository legacyKeyRepository,
            IPublisherRepository publisherRepository,
            IPublisherSupplierMappingWorkflow publisherSupplierMappingWorkflow,
            IMapper mapper,
            IGoodsSearchPipeline goodsSearchPipeline,
            ICacheManager cacheManager,
            MongoIntegrationSettings mongoIntegrationSettings)
        {
            _gameRepository=gameRepository;
            _productRepository=productRepository;
            _supplierMongoRepository=supplierMongoRepository;
            _legacyKeyRepository=legacyKeyRepository;
            _publisherRepository=publisherRepository;
            _publisherSupplierMappingWorkflow=publisherSupplierMappingWorkflow;
            _mapper=mapper;
            _goodsSearchPipeline=goodsSearchPipeline;
```

```
            _cacheManager=cacheManager;
            _mongoIntegrationSettings=mongoIntegrationSettings;
        }

        public async Task<bool> IsGoodsUnique(Goods goods)
        {
            GameEntity game = new GameEntity() { Key = goods.Key, Name = goods.Name
};
            ProductMongoEntity product = new ProductMongoEntity() { ProductName =
goods.Name, ProductKey = goods.Key };

            if (Guid.TryParse(goods.Id, out Guid gameId))
            {
                game.Id = gameId;

                var mappings = await
_cacheManager.GetGoodsProductMappingsCacheAsync();

                var mapping = mappings.FirstOrDefault(x => x.GameKey == goods.Key);
                if (mapping != null)
                {
                    product.ProductId = mapping.ProductId;
                }
            }
            else if (int.TryParse(goods.Id, out int productId))
            {
                product.ProductId = productId;
            }

            Task<bool> isGameUniqueTask = _gameRepository.IsGameUnique(game);
            Task<bool> isProductUniqueTask =
_productRepository.IsProductUnique(product);

            bool[] isGoodsUniqueTaskResult = await Task.WhenAll(isGameUniqueTask,
isProductUniqueTask);

            foreach (bool isGoodsUnique in isGoodsUniqueTaskResult)
            {
                if (!isGoodsUnique)
                {
                    return false;
                }
            }

            return true;
        }

        public Task<List<Goods>> GetAllAsync(string localizationCulture = null, bool
isIncludeDeleted = false)
        {
            var getGamesTask =
_gameRepository.GetAllEntitiesIncludeLocalization(localizationCulture,
isIncludeDeleted);
            var getProductsTask = _productRepository.GetAllAsync();

            return GetMergedGoodsAsync(getGamesTask, getProductsTask);
        }

        public async Task<List<Goods>> GetWithFiltersAsync(GoodsSearchRequest
request, string localizationCulture = null, bool isIncludeDeleted = false)
        {
            var gamesSearchRequest = _mapper.Map<GamesSearchRequest>(request);
```

```
            var productsSearchRequest = _mapper.Map<ProductSearchRequest>(request);
            await FillExcludedProductKeysAsync(productsSearchRequest);

            await SetupGameSearchForTemporalPublishersAsync(request.Distributors,
gamesSearchRequest);
            await SetupProductSearchForCategoriesAsync(request.Genres,
productsSearchRequest);

            Task<List<GameEntity>> getSqlWithFilters =
_gameRepository.GetWithFiltersAsync(gamesSearchRequest, localizationCulture);
            Task<List<ProductMongoEntity>> getMongoWithFilters =
_productRepository.GetWithFiltersAsync(productsSearchRequest);

            List<Goods> goods = await GetMergedGoodsAsync(getSqlWithFilters,
getMongoWithFilters);

            goods = _goodsSearchPipeline.Execute(goods, request);

            return goods;
        }

        public async Task<Goods> FindByKeyAsync(string key, string
localizationCultureCode = null, bool isIncludeDeleted = false)
        {
            var goodsProductMappingsList = await
_cacheManager.GetGoodsProductMappingsCacheAsync();
            if (goodsProductMappingsList.Any(x => x.GameKey == key))
            {
                var gameEntity = await _gameRepository.FindByKeyAsync(key,
localizationCultureCode, isIncludeDeleted);

                return await SetupMigratedGameEntity(gameEntity);
            }

            var entity = await _gameRepository.FindByKeyAsync(key,
localizationCultureCode, isIncludeDeleted);
            if (entity is null)
            {
                Guid gameId = await _legacyKeyRepository.FindGameIdAsync(key);
                if (gameId != Guid.Empty)
                {
                    entity = await _gameRepository.FindEntityByIdAsync(gameId,
isIncludeDeleted);
                }
            }

            if (entity != null)
            {
                return await FillGamesWithSupplierAsync(entity);
            }

            var product = await _productRepository.FindByKeyAsync(key);
            return await SetupProductIncludes(product);
        }

        public async Task<Goods> FindByKeyIncludeAllLocalizationsAsync(string key,
bool includeDeletedRows = false)
        {
            var goodsProductMappingsList = await
_cacheManager.GetGoodsProductMappingsCacheAsync();
            if (goodsProductMappingsList.Any(x => x.GameKey == key))
            {
```

```
                var gameEntity = await
_gameRepository.FindByKeyIncludeAllLocalizationsAsync(key, includeDeletedRows);

                return await SetupMigratedGameEntity(gameEntity);
            }

            var entity = await
_gameRepository.FindByKeyIncludeAllLocalizationsAsync(key, includeDeletedRows);
            if (entity is null)
            {
                Guid gameId = await _legacyKeyRepository.FindGameIdAsync(key);
                if (gameId != Guid.Empty)
                {
                    entity = await
_gameRepository.FindByIdIncludeAllLocalizationsAsync(gameId, includeDeletedRows);
                }
            }

            if (entity != null)
            {
                return await FillGamesWithSupplierAsync(entity);
            }

            var product = await _productRepository.FindByKeyAsync(key);
            return await SetupProductIncludes(product);
        }

        public async Task<Goods> FindAsync(string id, bool isIncludeDeleted = false,
bool asNoTracking = false)
        {
            if (Guid.TryParse(id, out Guid gameId))
            {
                GameEntity game;
                if (asNoTracking)
                {
                    game = await
_gameRepository.FindEntityByIdAsNoTrackingAsync(gameId, isIncludeDeleted);
                }
                else
                {
                    game = await _gameRepository.FindEntityByIdAsync(gameId,
isIncludeDeleted);
                }

                var goodsProductMappingsList = await
_cacheManager.GetGoodsProductMappingsCacheAsync();

                if (goodsProductMappingsList.Any(x => x.GameId == gameId))
                {
                    return await SetupMigratedGameEntity(game);
                }

                return await FillGamesWithSupplierAsync(game);
            }

            if (!int.TryParse(id, out int productId))
            {
                throw new ArgumentException("Not correct id format");
            }

            var product = await _productRepository.FindByIdAsync(productId);
```

```
            return await SetupProductIncludes(product);
        }

        public async Task<string> FindKeyByGoodsIdAsync(string id, bool
isIncludeDeleted = false)
        {
            if (Guid.TryParse(id, out Guid gameId))
            {
                return await _gameRepository.FindKeyByGameIdAsync(gameId,
isIncludeDeleted);
            }

            if (!int.TryParse(id, out int productId))
            {
                throw new ArgumentException("Not correct id format");
            }

            var product = await _productRepository.FindByIdAsync(productId);
            if (product is null)
            {
                throw new NotFoundException("Product with that key wasn't found");
            }

            return product.ProductKey;
        }

        public async Task<Guid> FindGameIdByKeyAsync(string key, bool
isIncludeDeleted = false)
        {
            return await _gameRepository.FindGameIdByKeyAsync(key, isIncludeDeleted);
        }

        public async Task<List<Goods>> FindByGenreAsync(Guid genreId, string
localizationCultureCode, bool isIncludeDeleted = false)
        {
            var genreCategoryList = await
_cacheManager.GetGenreCategoryMappingCacheAsync();
            var mapping = genreCategoryList.FirstOrDefault(x => x.GenreId ==
genreId);
            if (mapping == null)
            {
                var games = await _gameRepository.FindByGenreAsync(genreId,
localizationCultureCode, isIncludeDeleted);

                return _mapper.Map<List<Goods>>(games);
            }

            var findByCategoryTask =
_productRepository.FindByCategoryAsync(mapping.CategoryId);
            var findByGenreTask = _gameRepository.FindByGenreAsync(genreId,
isIncludeDeleted: isIncludeDeleted);

            return await GetMergedGoodsAsync(findByGenreTask, findByCategoryTask);
        }

        public async Task<List<Goods>> FindByPlatformTypeAsync(Guid platformId,
string localizationCultureCode = null, bool isIncludeDeleted = false)
        {
            var games = await _gameRepository.FindByPlatformTypeAsync(platformId,
localizationCultureCode, isIncludeDeleted);

            return _mapper.Map<List<Goods>>(games);
```

```
        }

        public async Task BatchUpdateUnitsInStockAsync(List<Goods> games)
        {
            var gamesList = new List<GameEntity>();
            var productsList = new List<ProductMongoEntity>();

            await SplitGoodsToGamesAndProducts(games, gamesList, productsList);

            var gameBatchUpdateTask =
_gameRepository.BatchUpdateUnitsInStockAsync(gamesList);
            var productBatchUpdateTask =
_productRepository.BatchUpdateUnitsInStockAsync(productsList);

            await Task.WhenAll(gameBatchUpdateTask, productBatchUpdateTask);
        }

        public Task IncreaseViewCountAsync(string id)
        {
            if (Guid.TryParse(id, out Guid gameId))
            {
                return _gameRepository.IncreaseViewCountAsync(gameId);
            }

            if (!int.TryParse(id, out int productId))
            {
                throw new ArgumentException("Not correct id format");
            }

            return _productRepository.IncreaseViewCountAsync(productId);
        }

        public Task<List<Goods>> FindByIdsAsync(List<string> ids, bool
isIncludeDeleted = false, bool asNoTracking = false)
        {
            var productIds = new List<int?>();
            var gamesIds = new List<Guid>();

            foreach (var id in ids)
            {
                if (Guid.TryParse(id, out Guid gameId))
                {
                    gamesIds.Add(gameId);
                }

                if (int.TryParse(id, out int productId))
                {
                    productIds.Add(productId);
                }
            }

            Task<List<GameEntity>> getGamesByIdsTask;
            if (asNoTracking)
            {
                getGamesByIdsTask =
_gameRepository.FindByIdsAsNoTrackingAsync(gamesIds);
            }
            else
            {
                getGamesByIdsTask = _gameRepository.FindEntitiesByIdsAsync(gamesIds,
isIncludeDeleted);
            }
```

```
            var getProductsByIdsTask = _productRepository.FindByIdsAsync(productIds);

            return GetMergedGoodsAsync(getGamesByIdsTask, getProductsByIdsTask);
        }

        public async Task CreateAsync(Goods item)
        {
            var entity = _mapper.Map<GameEntity>(item);

            await FillGamePublisherAsync(item, entity);

            await _gameRepository.CreateEntityAsync(entity);
        }

        private async Task FillGamePublisherAsync(Goods item, GameEntity entity)
        {
            if (Guid.TryParse(item.DistributorId, out Guid publisherId))
            {
                entity.PublisherId = publisherId;
            }
            else if (int.TryParse(item.DistributorId, out int supplierId))
            {
                await SetupPublisherFromMappings(entity, supplierId);
            }
        }

        public Task UpdateAsync(Goods item)
        {
            if (Guid.TryParse(item.Id, out Guid gameId))
            {
                return UpdateGameAsync(item, gameId);
            }
            else if (int.TryParse(item.Id, out int productId))
            {
                return StartProductMigrationAsync(item, productId);
            }

            throw new ArgumentException("Not correct id format");
        }

        public async Task UpdateLocalizationAsync(Goods item, Guid
chosenLocalization)
        {
            var localization = item.Localizations.First(x => x.LocalizationId ==
chosenLocalization);

            if (Guid.TryParse(item.Id, out Guid gameId))
            {
                localization.GoodsId = gameId;
            }
            else if (int.TryParse(item.Id, out int productId))
            {
                var mappings = await
_cacheManager.GetGoodsProductMappingsCacheAsync();
                var mapping = mappings.FirstOrDefault(x => x.ProductId == productId);

                var product = await FindByKeyAsync(item.Key);

                await StartProductMigrationAsync(product, productId);

                Guid migratedProductId = await
_gameRepository.FindGameIdByKeyAsync(product.Key);
```

```
                if (mapping != null && string.IsNullOrEmpty(mapping.GameKey))
                {
                    await SetupGenreForPartialMigratedProduct(item.Genres,
migratedProductId);
                }

                localization.GoodsId = migratedProductId;
            }
            else
            {
                throw new ArgumentException("Not correct id format");
            }

            var localizationEntity =
_mapper.Map<GameLocalizationEntity>(localization);
            await _gameRepository.UpdateLocalizationAsync(localizationEntity);
        }

        public Task RecoverAsync(Guid id)
        {
            return _gameRepository.RecoverAsync(id);
        }

        public Task SoftDeleteAsync(Guid id)
        {
            return _gameRepository.SoftDeleteAsync(id);
        }

        public async Task SoftDeleteAsync(string id)
        {
            if (Guid.TryParse(id, out Guid gameId))
            {
                await _gameRepository.SoftDeleteAsync(gameId);
            }
            else if (!int.TryParse(id, out int productId))
            {
                throw new ArgumentException("Not correct id format");
            }
            else
            {
                var product = await _productRepository.FindByIdAsync(productId);
                var productToMigrate = SetupProductForMigration(product);

                productToMigrate.IsDeleted = true;

                await _gameRepository.CreateEntityAsync(productToMigrate);
                await UpdateGoodsProductRelationAsync(product, productToMigrate.Id);
            }
        }

        public Task DeleteAsync(Guid id)
        {
            return _gameRepository.DeleteByIdAsync(id);
        }

        public Task DeleteAsync(Goods good)
        {
            if (Guid.TryParse(good.Id, out Guid id))
            {
                var game = _mapper.Map<GameEntity>(good);
```

```
                return _gameRepository.DeleteEntityAsync(game);
            }

            if (int.TryParse(good.Id, out int productId))
            {
                throw new InvalidOperationException("Can't delete product from
NorthwindOnlineStore, use a soft delete instead");
            }

            throw new ArgumentException("Not correct id format");
        }

        public async Task<int> GetCountOfGoods(bool isIncludeDeleted = false)
        {
            int productsCount = await _productRepository.GetCountOfProductsAsync();
            int gamesCount = await
_gameRepository.GetCountOfGamesAsync(isIncludeDeleted);

            return productsCount + gamesCount;
        }

        public async Task<List<GameGenre>> GetGamesGenresByGameIdAsync(Guid gameId)
        {
            List<GameGenreEntity> gameGenreEntities = await
_gameRepository.GetGameGenresByGameIdAsync(gameId);

            return _mapper.Map<List<GameGenre>>(gameGenreEntities);
        }

        public async Task<List<GameGenre>> GetGamesGenresByGenreIdAsync(Guid genreId)
        {
            List<GameGenreEntity> gameGenreEntities = await
_gameRepository.GetGameGenresByGenreIdAsync(genreId);

            return _mapper.Map<List<GameGenre>>(gameGenreEntities);
        }

        public Task DeleteRangeGamesGenresAsync(List<GameGenre> gameGenresToDelete)
        {
            var gameGenreEntities =
_mapper.Map<List<GameGenreEntity>>(gameGenresToDelete);

            return _gameRepository.DeleteRangeGamesGenresAsync(gameGenreEntities);
        }

        public async Task<List<GamePlatformType>> GetGamePlatformByGameIdAsync(Guid
gameId)
        {
            var gamePlatformTypeEntities = await
_gameRepository.GetGamesPlatformsGameIdAsync(gameId);

            return _mapper.Map<List<GamePlatformType>>(gamePlatformTypeEntities);
        }

        public async Task<List<GamePlatformType>>
GetGamePlatformByPlatformTypeIdAsync(Guid platformTypeId)
        {
            var gamePlatformTypeEntities = await
_gameRepository.GetGamesPlatformsByPlatformTypeIdAsync(platformTypeId);

            return _mapper.Map<List<GamePlatformType>>(gamePlatformTypeEntities);
        }
```

```
        public Task DeleteRangeGamesPlatformTypesAsync(List<GamePlatformType>
gameGenresToDelete)
        {
            var gameGenreEntities =
_mapper.Map<List<GamePlatformTypeEntity>>(gameGenresToDelete);

            return _gameRepository.DeleteRangeGamesPlatformsAsync(gameGenreEntities);
        }

        public Task CreateRangeGamesGenresAsync(List<GameGenre> newGameGanreList)
        {
            var gameGenreEntities =
_mapper.Map<List<GameGenreEntity>>(newGameGanreList);

            return _gameRepository.CreateRangeGamesGenresAsync(gameGenreEntities);
        }

        public Task CreateRangeGamesPlatformTypesAsync(List<GamePlatformType>
newGamePlatformTypeList)
        {
            var gamesPlatformTypesEntities =
_mapper.Map<List<GamePlatformTypeEntity>>(newGamePlatformTypeList);

            return
_gameRepository.CreateRangeGamesPlatformTypesAsync(gamesPlatformTypesEntities);
        }

        private async Task<Goods> FillGamesWithSupplierAsync(GameEntity entity)
        {
            var goods = _mapper.Map<Goods>(entity);
            var publishersSuppliersMappingsList = await
_cacheManager.GetPublisherSupplierMappingsCacheAsync();

            var mapping = publishersSuppliersMappingsList.FirstOrDefault(x =>
x.PublisherId == entity.PublisherId);
            if (mapping != null)
            {
                goods.DistributorId = mapping.SupplierId.ToString();
                await SetupProductSupplier(mapping.SupplierId, goods);
            }

            return goods;
        }

        private async Task UpdateGoodsProductRelationAsync(ProductMongoEntity
product, Guid id)
        {
            List<GoodsProductMapping> goodsProductMappingsList = await
_cacheManager.GetGoodsProductMappingsCacheAsync();
            GoodsProductMapping mapping = goodsProductMappingsList.FirstOrDefault(m
=> m.GameId == id);

            if (mapping == null)
            {
                mapping = _mapper.Map<GoodsProductMapping>(product);
                mapping.GameId = id;
                mapping.GameKey = product.ProductKey;

                await _cacheManager.AddGoodsProductMappingCache(mapping);
            }
            else
```

```
            {
                Guid mappingId = mapping.Id;

                mapping = _mapper.Map<GoodsProductMapping>(product);
                mapping.GameId = id;
                mapping.GameKey = product.ProductKey;

                mapping.Id = mappingId;
                await _cacheManager.UpdateGoodsProductMappingAsync(mapping);
            }
        }

        private async Task SplitGoodsToGamesAndProducts(List<Goods> games,
List<GameEntity> gamesList, List<ProductMongoEntity> productsList)
        {
            foreach (var goods in games)
            {
                if (Guid.TryParse(goods.Id, out Guid gameId))
                {
                    await HandleMigratedProducts(productsList, goods, gameId);

                    var game = _mapper.Map<GameEntity>(goods);

                    game.Id = gameId;

                    gamesList.Add(game);
                }
                else if (int.TryParse(goods.Id, out int productId))
                {
                    var product = _mapper.Map<ProductMongoEntity>(goods);

                    product.ProductId = productId;

                    productsList.Add(product);
                }
            }
        }

        private async Task HandleMigratedProducts(List<ProductMongoEntity>
productsList, Goods goods, Guid gameId)
        {
            var goodsProductMappingsList = await
_cacheManager.GetGoodsProductMappingsCacheAsync();
            var mapping = goodsProductMappingsList.FirstOrDefault(x => x.GameId ==
gameId && x.IsFullyMigrated);

            if (mapping != null)
            {
                var product = _mapper.Map<ProductMongoEntity>(goods);
                product.ProductId = mapping.ProductId;
                productsList.Add(product);
            }
        }

        private async Task StartProductMigrationAsync(Goods item, int productId)
        {
            var product = await _productRepository.FindByIdAsync(productId);
            var productToMigrate = SetupProductForMigration(product, item);

            await SetupDistributorForMigratedProduct(item.DistributorId, product,
productToMigrate);
```

```
            await FinishProductMigrationAsync(productId, productToMigrate);

            await UpdateGoodsProductRelationAsync(product, productToMigrate.Id);
        }

        private async Task FinishProductMigrationAsync(int productId, GameEntity
productToMigrate)
        {
            var goodsProductMappingsList = await
_cacheManager.GetGoodsProductMappingsCacheAsync();
            var mapping = goodsProductMappingsList.FirstOrDefault(m => m.ProductId ==
productId);

            if (mapping == null)
            {
                await _gameRepository.CreateEntityAsync(productToMigrate);
            }
            else
            {
                productToMigrate.Id = mapping.GameId;
                await _gameRepository.UpdateEntityAsync(productToMigrate);
            }
        }

        private async Task SetupDistributorForMigratedProduct(string distributorId,
ProductMongoEntity product, GameEntity productToMigrate)
        {
            if (int.TryParse(distributorId, out int supplierId))
            {
                var mapping = await
_publisherSupplierMappingWorkflow.GetMappingAsync(supplierId);

                if (mapping is null)
                {
                    mapping = await
_publisherSupplierMappingWorkflow.CreateMappingAsync(supplierId);
                }

                productToMigrate.PublisherId = mapping.PublisherId;
            }
            else if (Guid.TryParse(distributorId, out Guid publisherId))
            {
                productToMigrate.PublisherId = publisherId;
            }
            else
            {
                product.SupplierId = null;
            }
        }

        private async Task UpdateGameAsync(Goods item, Guid gameId)
        {
            string legacyKey = await FindKeyByGoodsIdAsync(item.Id);

            var entity = _mapper.Map<GameEntity>(item);
            entity.Id = gameId;

            await SetupDistributorForGameAsync(item, entity);

            await _gameRepository.UpdateEntityAsync(entity);

            if (legacyKey == item.Key)
```

```
            {
                return;
            }

            await _legacyKeyRepository.CreateAsync(legacyKey, gameId);

            var mappings = await _cacheManager.GetGoodsProductMappingsCacheAsync();

            var mapping = mappings.FirstOrDefault(x => x.GameKey == legacyKey);
            if (mapping != null)
            {
                mapping.GameKey = entity.Key;
                await _cacheManager.UpdateGoodsProductMappingAsync(mapping);
            }
        }

        private async Task SetupDistributorForGameAsync(Goods item, GameEntity
entity)
        {
            if (int.TryParse(item.DistributorId, out int supplierId))
            {
                await SetupPublisherFromMappings(entity, supplierId);
            }
            else if (Guid.TryParse(item.DistributorId, out Guid publisherId))
            {
                entity.PublisherId = publisherId;
            }
        }

        private async Task SetupPublisherFromMappings(GameEntity entity, int
supplierId)
        {
            var mapping = await
_publisherSupplierMappingWorkflow.GetMappingAsync(supplierId);
            if (mapping == null)
            {
                mapping = await
_publisherSupplierMappingWorkflow.CreateMappingAsync(supplierId);
            }

            entity.PublisherId = mapping.PublisherId;
        }

        private async Task<List<Goods>> GetMergedGoodsAsync(Task<List<GameEntity>>
sqlGetTask, Task<List<ProductMongoEntity>> MongoGetTask)
        {
            await Task.WhenAll(sqlGetTask, MongoGetTask);

            var games = await sqlGetTask;
            var products = await MongoGetTask;

            List<Goods> goodsList = await
SetupFillGamesWithMongoEntitiesAsync(games);

            var mongoGoodsList = await SetupProductsIncludes(products);

            goodsList.AddRange(mongoGoodsList);
            return goodsList;
        }

        private async Task<List<Goods>>
SetupFillGamesWithMongoEntitiesAsync(List<GameEntity> games)
```

```
        {
            var goodsProductMappingsList = await
_cacheManager.GetGoodsProductMappingsCacheAsync();
            var migratedGames = games.Where(x => goodsProductMappingsList.Any(c =>
c.GameKey == x.Key)).ToList();

            games.RemoveAll(g => migratedGames.Any(mg => mg.Key == g.Key));

            var goodsList = new List<Goods>();
            foreach (var game in games)
            {
                var goods = await FillGamesWithSupplierAsync(game);
                goodsList.Add(goods);
            }

            foreach (var item in migratedGames)
            {
                var goods = await SetupMigratedGameEntity(item);
                goodsList.Add(goods);
            }

            return goodsList;
        }

        private async Task<List<Goods>>
SetupProductsIncludes(List<ProductMongoEntity> products)
        {
            List<Goods> goodsList = new List<Goods>();
            foreach (var product in products)
            {
                var goods = await SetupProductIncludes(product);
                goodsList.Add(goods);
            }

            return goodsList;
        }

        private async Task<Goods> SetupMigratedGameEntity(GameEntity game)
        {
            var goods = _mapper.Map<Goods>(game);

            var goodsProductMappingsList = await
_cacheManager.GetGoodsProductMappingsCacheAsync();
            var mappings = goodsProductMappingsList.First(x => x.GameKey ==
game.Key);

            if (mappings.ProductId.HasValue)
            {
                await SetupProductUnitsInStock(mappings.ProductId.Value, goods);
            }

            var publishersSuppliersMappingsList = await
_cacheManager.GetPublisherSupplierMappingsCacheAsync();
            var pubSubMapping = publishersSuppliersMappingsList.FirstOrDefault(x =>
x.PublisherId == game.PublisherId);

            if (pubSubMapping != null)
            {
                await SetupProductSupplier(pubSubMapping.SupplierId, goods);
            }

            return goods;
```

```
        }

        private async Task SetupProductUnitsInStock(int productId, Goods goods)
        {
            var product = await _productRepository.FindByIdAsync(productId);
            if (product != null && product.UnitsInStock.HasValue)
            {
                goods.UnitsInStock = (short)product.UnitsInStock.Value;
            }
        }

        private async Task<Goods> SetupProductIncludes(ProductMongoEntity product)
        {
            if (product is null)
            {
                return null;
            }

            var goods = _mapper.Map<Goods>(product);
            goods.DateOfAdding =
_mongoIntegrationSettings.NorthwindOnlineStoreDbConnectionDate;

            if (product.CategoryId.HasValue)
            {
                await SetupProductCategory(product.CategoryId.Value, goods);
            }

            if (product.SupplierId.HasValue)
            {
                await SetupProductSupplier(product.SupplierId.Value, goods);
            }

            return goods;
        }

        private async Task SetupProductSupplier(int supplierId, Goods goods)
        {
            var supplier = await _supplierMongoRepository.FindByIdAsync(supplierId);
            goods.Distributor = _mapper.Map<Distributor>(supplier);
            goods.DistributorId = supplierId.ToString();

            var mapping = await
_publisherSupplierMappingWorkflow.GetMappingAsync(supplierId);
            if (mapping != null)
            {
                var publisher = await
_publisherRepository.FindEntityByIdAsync(mapping.PublisherId);
                goods.Distributor.UserId = publisher.UserId;
            }
        }

        private async Task SetupProductCategory(int categoryId, Goods goods)
        {
            var genreCategoryList = await
_cacheManager.GetGenreCategoryMappingCacheAsync();

            var genreCategory = genreCategoryList.FirstOrDefault(x => x.CategoryId ==
categoryId);
            if (genreCategory == null)
            {
                return;
            }
```

```
            goods.Genres.Add(genreCategory.Genre);
        }

        private async Task SetupGenreForPartialMigratedProduct(List<Genre> genres,
Guid migratedProductId)
        {
            var gameGenre = new List<GameGenreEntity>();
            foreach (var genre in genres)
            {
                gameGenre.Add(new GameGenreEntity
                {
                    GameId = migratedProductId,
                    GenreId = genre.Id
                });
            }

            await _gameRepository.CreateRangeGamesGenresAsync(gameGenre);
        }

        private GameEntity SetupProductForMigration(ProductMongoEntity product, Goods
goods = null)
        {
            GameEntity game;
            Guid newGuid = Guid.NewGuid();
            if (goods is null)
            {
                game = _mapper.Map<GameEntity>(product);
                game.Id = newGuid;
            }
            else
            {
                goods.Id = newGuid.ToString();
                game = _mapper.Map<GameEntity>(goods);
            }

            return game;
        }

        private async Task FillExcludedProductKeysAsync(ProductSearchRequest
productsSearchRequest)
        {
            var mappings = await _cacheManager.GetGoodsProductMappingsCacheAsync();
            productsSearchRequest.ExcludedProductKeys = mappings.Select(x =>
x.GameKey).ToList();
        }

        private async Task SetupProductSearchForCategoriesAsync(List<Guid>
chosedGenres, ProductSearchRequest productsSearchRequest)
        {
            foreach (var genreId in chosedGenres)
            {
                var genreCategoryList = await
_cacheManager.GetGenreCategoryMappingCacheAsync();
                var mapping = genreCategoryList.FirstOrDefault(x => x.GenreId ==
genreId);

                if (mapping != null)
                {

productsSearchRequest.Categories.Add(mapping.CategoryId.ToString());
                }
```

```
            }
        }

        private async Task SetupGameSearchForTemporalPublishersAsync(List<string>
chosedDistributors,

GamesSearchRequest gamesSearchRequest)
        {
            foreach (var item in chosedDistributors)
            {
                if (int.TryParse(item, out int supplierId))
                {
                    var publishersSuppliersMappingsList = await
_cacheManager.GetPublisherSupplierMappingsCacheAsync();
                    var mapping = publishersSuppliersMappingsList.FirstOrDefault(x =>
x.SupplierId == supplierId);

                    if (mapping != null)
                    {

gamesSearchRequest.Publishers.Add(mapping.PublisherId.ToString());
                    }
                }
            }
        }
    }
}
```

GameStoreContext.cs

```
using GameStore.DAL.Entities;
using GameStore.DAL.Entities.Localizations;
using GameStore.DomainModels.Enums;
using Microsoft.EntityFrameworkCore;

namespace GameStore.DAL.Context
{
    public class GameStoreContext : DbContext
    {
        public DbSet<GameEntity> Games { get; set; }

        public DbSet<CommentEntity> Comments { get; set; }

        public DbSet<GenreEntity> Genres { get; set; }

        public DbSet<PlatformTypeEntity> PlatformTypes { get; set; }

        public DbSet<PublisherEntity> Publishers { get; set; }

        public DbSet<OrderDetailsEntity> OrderDetails { get; set; }

        public DbSet<OrderEntity> Orders { get; set; }

        public DbSet<LegacyKeyEntity> LegacyKeys { get; set; }

        public DbSet<UserEntity> Users { get; set; }

        public DbSet<GenreCategoryMappingEntity> GenreCategory { get; set; }

        public DbSet<GoodsProductMappingEntity> GoodsProduct { get; set; }

        public DbSet<PublisherSupplierMappingEntity> PublisherSupplier { get; set; }
```

```
        public DbSet<LocalizationEntity> Localizations { get; set; }

        public DbSet<GameLocalizationEntity> GameLocalizations { get; set; }

        public DbSet<PlatformTypeLocalizaitionEntity> PlatformTypeLocalizations {
get; set; }

        public DbSet<GenreLocalizationEntity> GenreLocalizations { get; set; }

        public DbSet<GameGenreEntity> GamesGenres { get; set; }

        public DbSet<GamePlatformTypeEntity> GamesPlatformTypes { get; set; }

        public GameStoreContext()
        {
        }

        public GameStoreContext(DbContextOptions<GameStoreContext> options) :
base(options)
        {
        }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.EnableSensitiveDataLogging();
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<UserEntity>(entity =>
            {
                entity.HasKey(x => x.Id);
                entity.HasIndex(x => x.Username)
                    .IsUnique();
                entity.HasIndex(x => x.Email)
                    .IsUnique();
                entity.Property(x => x.Role)
                    .HasDefaultValue(UserRoles.User)
                    .HasConversion<string>();

                entity.HasMany(x => x.Comments)
                    .WithOne(x => x.Author)
                    .HasForeignKey(x => x.AuthorId);
                entity.HasMany(x => x.Orders)
                    .WithOne(x => x.Customer)
                    .HasForeignKey(x => x.CustomerId);
            });

            modelBuilder.Entity<LegacyKeyEntity>(entity =>
            {
                entity.HasKey(x => x.Id);
                entity.HasIndex(x => x.LegacyKey)
                    .IsUnique();
            });

            modelBuilder.Entity<PublisherEntity>(entity =>
            {
                entity.HasKey(x => x.Id);
                entity.HasIndex(x => x.CompanyName)
                    .IsUnique();
                entity.HasMany(x => x.PublishedGames)
```

```
        .WithOne(x => x.Publisher)
        .HasForeignKey(x => x.PublisherId);
});

modelBuilder.Entity<OrderEntity>(entity =>
{
    entity.HasKey(x => x.Id);
    entity.HasMany(x => x.OrderDetails)
        .WithOne(x => x.Order)
        .HasForeignKey(x => x.OrderId);
    entity.Property(p => p.Status)
        .HasDefaultValue(OrderStatus.Open)
        .HasConversion<string>();
});

modelBuilder.Entity<OrderDetailsEntity>(entity =>
{
    entity.HasKey(x => x.Id);
});

modelBuilder.Entity<GenreEntity>(entity =>
{
    entity.HasKey(x => x.Id);
    entity.HasIndex(x => x.Name)
        .IsUnique();
    entity.HasOne(x => x.Parent)
        .WithMany(x => x.SubGenres)
        .HasForeignKey(x => x.ParentId)
        .IsRequired(false)
        .OnDelete(DeleteBehavior.Restrict);
});

modelBuilder.Entity<CommentEntity>(entity =>
{
    entity.HasKey(x => x.Id);
    entity.Property(x => x.Name);
    entity.Property(x => x.Body);
    entity.HasOne(x => x.Parent)
        .WithMany(x => x.Replies)
        .HasForeignKey(x => x.ParentId)
        .IsRequired(false)
        .OnDelete(DeleteBehavior.Restrict);
    entity.Property(p => p.Type)
        .HasDefaultValue(CommentType.Standalone)
        .HasConversion<string>();
});

modelBuilder.Entity<GameEntity>(entity =>
{
    entity.HasKey(x => x.Id);
    entity.HasIndex(x => x.Key)
        .IsUnique();
    entity.HasMany(x => x.Comments)
        .WithOne(x => x.Game)
        .HasForeignKey(x => x.GameId);
});

modelBuilder.Entity<GameGenreEntity>(entity =>
{
    entity.HasKey(x => new { x.GameId, x.GenreId });

    entity.HasOne(x => x.Game)
```

```
                .WithMany(x => x.GameGenres)
                .HasForeignKey(x => x.GameId);

            entity.HasOne(x => x.Genre)
                .WithMany(x => x.GameGenres)
                .HasForeignKey(x => x.GenreId);
        });

        modelBuilder.Entity<GamePlatformTypeEntity>(entity =>
        {
            entity.HasKey(x => new { x.GameId, x.PlatformId });

            entity.HasOne(x => x.Game)
                .WithMany(x => x.GamesPlatformTypes)
                .HasForeignKey(x => x.GameId);

            entity.HasOne(x => x.PlatformType)
                .WithMany(x => x.GamesPlatformTypes)
                .HasForeignKey(x => x.PlatformId);
        });

        modelBuilder.Entity<PlatformTypeEntity>(entity =>
        {
            entity.HasKey(x => x.Id);

            entity.HasIndex(x => x.Type)
                .IsUnique();
        });

        modelBuilder.Entity<PublisherSupplierMappingEntity>(entity =>
        {
            entity.HasKey(x => new { x.PublisherId, x.SupplierId });
        });

        modelBuilder.Entity<GenreCategoryMappingEntity>(entity =>
        {
            entity.HasKey(x => new { x.GenreId, x.CategoryId });
        });

        modelBuilder.Entity<GoodsProductMappingEntity>(entity =>
        {
            entity.HasKey(x => x.Id);
        });

        modelBuilder.Entity<GenreEntity>()
            .HasData(DataSeed.GetGenreSeed());

        modelBuilder.Entity<PlatformTypeEntity>()
            .HasData(DataSeed.GetPlatformTypeSeed());

        modelBuilder.Entity<UserEntity>()
            .HasData(DataSeed.GetUserSeed());

        modelBuilder.Entity<GenreCategoryMappingEntity>()
            .HasData(DataSeed.GetGenreCategoryMappingSeed());

        modelBuilder.Entity<LocalizationEntity>()
            .HasData(DataSeed.GetLocalizationsSeed());
    }
  }
}
```

GoodsService.cs

```
using AutoMapper;
using GameStore.BLL.Interfaces;
using GameStore.BLL.Settings;
using GameStore.DAL.Abstractions.Interfaces;
using GameStore.DomainModels.Exceptions;
using GameStore.DomainModels.Models;
using GameStore.DomainModels.Models.CreateModels;
using GameStore.DomainModels.Models.EditModels;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;

namespace GameStore.BLL.Services
{
    public class GoodsService : IGoodsService
    {
        private readonly IGoodsRepository _goodsRepository;
        private readonly IMapper _mapper;
        private readonly DownloadingFileSettings _downloadingFileConfiguration;

        public GoodsService(
            IGoodsRepository gameRepository,
            DownloadingFileSettings downloadingFileConfiguration,
            IMapper mapper)
        {
            _goodsRepository = gameRepository;
            _downloadingFileConfiguration = downloadingFileConfiguration;
            _mapper = mapper;
        }

        public async Task<Goods> GetGoodsByIdAsync(Guid id)
        {
            Goods foundGame = await _goodsRepository.FindAsync(id.ToString());
            if (foundGame is null)
            {
                throw new NotFoundException($"Game with id {id} wasn't found");
            }

            return foundGame;
        }

        public Task<Goods> GetGoodsByKeyAsync(string key, string
localizationCultureCode = null, bool includeDeletedRows = false)
        {
            return _goodsRepository.FindByKeyAsync(key, localizationCultureCode,
includeDeletedRows);
        }

        public Task<Goods> GetGoodsByKeyIncludeAllLocalizationsAsync(string key, bool
includeDeletedRows = false)
        {
            return _goodsRepository.FindByKeyIncludeAllLocalizationsAsync(key,
includeDeletedRows);
        }

        public async Task<string> CreateGoodsAsync(CreateGoodsRequest gameToCreate)
        {
            gameToCreate.Id = Guid.NewGuid();
```

```
            gameToCreate.DateOfAdding = DateTime.UtcNow;

            Goods newGame = _mapper.Map<Goods>(gameToCreate);

            if (string.IsNullOrEmpty(gameToCreate.Key))
            {
                newGame.Key = GenerateKey(newGame.Name);
            }

            bool isUnique = await _goodsRepository.IsGoodsUnique(newGame);
            if (!isUnique)
            {
                throw new NotUniqueException("Game with that key is already exist");
            }

            await _goodsRepository.CreateAsync(newGame);
            return newGame.Key;
        }

        public Task EditGoodsAsync(EditGoodsRequest gameToEdit)
        {
            Goods changedGame = _mapper.Map<Goods>(gameToEdit);
            if (gameToEdit.ChosedLocalization.HasValue)
            {
                return _goodsRepository.UpdateLocalizationAsync(changedGame,
gameToEdit.ChosedLocalization.Value);
            }

            return UpdateOriginalGoodsAsync(changedGame);
        }

        public async Task RecoverGoodsAsync(Guid id)
        {
            try
            {
                await _goodsRepository.RecoverAsync(id);
            }
            catch (NotFoundException)
            {
                throw new NotFoundException($"Game with id {id} wasn't found");
            }
        }

        public async Task SoftDeleteGoodsAsync(string id)
        {
            try
            {
                await _goodsRepository.SoftDeleteAsync(id);
            }
            catch (NotFoundException)
            {
                throw new NotFoundException($"Game with id {id} wasn't found");
            }
        }

        public async Task HardDeleteGoodsAsync(Guid id)
        {
            try
            {
                await _goodsRepository.DeleteAsync(id);
            }
            catch (NotFoundException)
```

```
            {
                throw new NotFoundException($"Game with id {id} wasn't found");
            }
        }

        public async Task IncreaseViewCountAsync(string gameId)
        {
            await _goodsRepository.IncreaseViewCountAsync(gameId);
        }

        public Task<List<Goods>> GetGoodsAsync(string localizationCultureCode = null)
        {
            return _goodsRepository.GetAllAsync(localizationCultureCode);
        }

        public Task<List<Goods>> GetFilteredGoodsAsync(GoodsSearchRequest filters,
string localizationCultureCode = null)
        {
            return _goodsRepository.GetWithFiltersAsync(filters,
localizationCultureCode);
        }

        public Task AddGenreToGoodsAsync(Guid gameId, List<Genre> genres)
        {
            ValidateChangeForGame(gameId, genres);

            List<GameGenre> newGameGanreList = new List<GameGenre>();

            foreach (var g in genres)
            {
                GameGenre gameGenre = new GameGenre()
                {
                    GameId = gameId,
                    GenreId = g.Id
                };
                newGameGanreList.Add(gameGenre);
            }

            return _goodsRepository.CreateRangeGamesGenresAsync(newGameGanreList);
        }

        public async Task UpdateGenreForGameAsync(Guid gameId, List<Genre> genres)
        {
            List<GameGenre> oldGameGenres = await
_goodsRepository.GetGamesGenresByGameIdAsync(gameId);
            List<Genre> newGenres = genres.Where(g => oldGameGenres.All(gg =>
gg.GenreId != g.Id)).ToList();
            List<GameGenre> removedGenres = oldGameGenres.Where(gg => genres.All(g =>
g.Id != gg.GenreId)).ToList();

            if (newGenres.Count != 0)
            {
                await AddGenreToGoodsAsync(gameId, newGenres);
            }

            if (removedGenres.Count != 0)
            {
                await _goodsRepository.DeleteRangeGamesGenresAsync(removedGenres);
            }
        }
```

```
        public Task RemoveGenreFromGoodsAsync(Guid gameId, List<Genre>
genresToRemove)
        {
            ValidateChangeForGame(gameId, genresToRemove);

            List<GameGenre> removeGameGanreList = new List<GameGenre>();

            foreach (var g in genresToRemove)
            {
                GameGenre gameGenre = new GameGenre()
                {
                    GameId = gameId,
                    GenreId = g.Id
                };
                removeGameGanreList.Add(gameGenre);
            }

            return _goodsRepository.DeleteRangeGamesGenresAsync(removeGameGanreList);
        }

        public Task AddPlatformTypeToGoodsAsync(Guid gameId, List<PlatformType>
platformTypesToAdd)
        {
            ValidateChangeForGame(gameId, platformTypesToAdd);

            List<GamePlatformType> newGamePlatformTypeList = new
List<GamePlatformType>();

            foreach (var p in platformTypesToAdd)
            {
                GamePlatformType gamelatformType = new GamePlatformType()
                {
                    GameId = gameId,
                    PlatformId = p.Id
                };
                newGamePlatformTypeList.Add(gamelatformType);
            }

            return
_goodsRepository.CreateRangeGamesPlatformTypesAsync(newGamePlatformTypeList);
        }

        public async Task UpdatePlatformTypeForGameAsync(Guid gameId,
List<PlatformType> platformTypes)
        {
            List<GamePlatformType> oldGamePlatformType = await
_goodsRepository.GetGamePlatformByGameIdAsync(gameId);
            List<PlatformType> newPlatformTypes = platformTypes.Where(g =>
oldGamePlatformType.All(gp => gp.PlatformId != g.Id)).ToList();
            List<GamePlatformType> removedPlatformTypes =
oldGamePlatformType.Where(gp => platformTypes.All(g => g.Id !=
gp.PlatformId)).ToList();

            if (newPlatformTypes.Count != 0)
            {
                await AddPlatformTypeToGoodsAsync(gameId, newPlatformTypes);
            }

            if (removedPlatformTypes.Count != 0)
            {
                await
_goodsRepository.DeleteRangeGamesPlatformTypesAsync(removedPlatformTypes);
```

```
            }
        }

        public Task RemovePlatformTypeFromGoodsAsync(Guid gameId, List<PlatformType>
platformTypesToRemove)
        {
            ValidateChangeForGame(gameId, platformTypesToRemove);

            List<GamePlatformType> removeGamePlatformTypeList = new
List<GamePlatformType>();

            foreach (var p in platformTypesToRemove)
            {
                GamePlatformType gamePlatformType = new GamePlatformType()
                {
                    GameId = gameId,
                    PlatformId = p.Id
                };
                removeGamePlatformTypeList.Add(gamePlatformType);
            }

            return
_goodsRepository.DeleteRangeGamesPlatformTypesAsync(removeGamePlatformTypeList);
        }

        public Task<int> GetCountOfGoodsAsync()
        {
            return _goodsRepository.GetCountOfGoods();
        }

        public Task<byte[]> DownloadAsync(string key)
        {
            string baseDirectory = AppContext.BaseDirectory;
            string newPath = Path.GetFullPath(
                Path.Combine(baseDirectory, _downloadingFileConfiguration.FolderPath
+ key + _downloadingFileConfiguration.FileExtension));

            if (!File.Exists(newPath))
            {
                throw new NotFoundException($"Download file for game {key} was'nt
found");
            }

            return File.ReadAllBytesAsync(newPath);
        }

        private string GenerateKey(string name)
        {
            if (string.IsNullOrEmpty(name))
            {
                throw new ArgumentException("Can't create a game without a name");
            }

            List<char> removeCharsList = new List<char>() { '@', '_', ',', '.', ' '
};

            return string.Concat(name.Split(removeCharsList.ToArray())).ToLower();
        }

        private void ValidateChangeForGame<T>(Guid gameId, List<T> changeList)
        {
            if (gameId == Guid.Empty)
```

```
            {
                throw new ArgumentException("No game id was given");
            }

            if (changeList.Count == 0)
            {
                throw new ArgumentException($"list of {typeof(T).Name} is empty");
            }
        }

        private async Task UpdateOriginalGoodsAsync(Goods changedGame)
        {
            bool isUnique = await _goodsRepository.IsGoodsUnique(changedGame);
            if (!isUnique)
            {
                throw new NotUniqueException("Game with that key is already exist");
            }

            await _goodsRepository.UpdateAsync(changedGame);

            Guid goodId = await
_goodsRepository.FindGameIdByKeyAsync(changedGame.Key);

            await UpdateGenreForGameAsync(goodId, changedGame.Genres);
            await UpdatePlatformTypeForGameAsync(goodId, changedGame.PlatformTypes);
        }
    }
}
```

У цьому додатку були вказали файли з кожного розділу файлової системи. Повний перелік файлів зі змістом знаходяться на електронному носієві.

ДОДАТОК Б

ВІДГУК
керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему:
«Розробка інтернет-магазину на основі фреймворку ASP.Net з використанням HTLM, CSS та C#»
студента групи 121-19-1 Срібного Данила Микитовича

Керівник економічного розділу
доцент каф. ПЕП та ПУ, к.е.н. **Л. В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом Срібний.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом Срібний.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
diplom.zip	Архів. Містить коди програми.
Презентація	
Презентація Срібний.ppt	Презентація кваліфікаційної роботи.