ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-додатку для управління персональною колекцією з використанням технологій ASP.NET»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Владислав БАШИЧ
(підпис)

Виконав: здобувач(ка) вищої освіти групи ПД-44

Владислав БАШИЧ

Керівник: Володимир САДОВЕНКО
к.ф.м.н., доцент

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Башичу Владиславу Андрійовичу

1. Тема кваліфікаційної роботи: Розробка web-застосунку для управління персональною колекцією плюшевих іграшок.
керівник кваліфікаційної роботи к.ф.м.н., професор Володимир САДОВЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи розробки web-застосунків, опис роботи web-додатків, документація фреймворків ASP.NET Core та Entity Framework Core.

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

1. Огляд застосунку для управління персональною колекцією плюшевих іграшок та аналіз існуючих аналогів, визначення вимог до програмного забезпечення.
2. Огляд та аналіз засобів реалізації web-додатку для управління персональною колекцією плюшевих іграшок.

3. Проектування архітектури застосунку для управління персональною колекцією плюшевих іграшок.
4. Програмне втілення та тестування застосунку для управління персональною колекцією плюшевих іграшок.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до додатку.
3. Програмні засоби реалізації.
4. Граф діалогу та діаграма діяльності.
5. Діаграма класів та ER-діаграма.
6. Екранні форми.
7. Апробація результатів дослідження

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд та аналіз існуючих рішень з управління плюшевою колекцією іграшок	14.03-22.03.2025	
4	Проектування вимог до програмного забезпечення	23.03-31.03.2025	
5	Огляд засобів реалізації програмного забезпечення для управління персональною колекцією плюшевих іграшок	01.04-09.04.2025	
6	Проектування та розробка програмного забезпечення для управління персональною колекцією плюшевих іграшок	10.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Владислав БАШИЧ
(підпис)

Керівник
кваліфікаційної роботи _____ Володимир САДОВЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 1 табл., 15 рис., 22 джерел.

Мета роботи – спростити взаємодію користувачів для обміну елементами персональних колекцій плюшевих іграшок

Об'єкт дослідження – процес обміну інформацією про персональні колекції користувачів.

Предмет дослідження – web-застосунок для взаємодії користувачів з об'єктами персональних колекцій.

Короткий зміст роботи: В роботі проведено ретельний аналіз засобів цифрової організації колекційної інформації у web-середовищах, значну увагу сконцентровано на побудові інтерфейсу користувача та механізмах взаємодії між користувачами. Проаналізовано інструментальні засоби існуючих рішень різного типу колекцій: Libib, Sortly, Collector System, Pinterest. Розроблено web-додаток та програмно реалізовані ключові функціональні можливості, насамперед: автентифікація та аутентифікація, можливість створення, видалення, редагування та поширювання колекцій та іграшок, система модерації спільних публікацій, конвертування об'єктів у PDF-формат для зручності експорту поза межами застосунку. В роботі використано сучасні технологічні підходи до проектування програмного забезпечення, фреймворк ASP.NET Core MVC реалізує серверну частину, для доступу до бази даних використано Entity Framework Core.

Сферою використання застосунку є спільноти та індивідуальні особи зацікавлені в загальнодоступності їх персональних колекцій плюшевих іграшок.

КЛЮЧОВІ СЛОВА: РОЗРОБКА, АРХІТЕКТУРА, УПРАВЛІННЯ КОЛЛЕКЦІЯМИ, SQL, ASP.NET CORE, C#, ПРОЕКТУВАННЯ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CRUD – Create, Read, Update, Delete

QR Code – Quick Response Code

ID – Identifier

HTML – HyperText Markup Language

AJAX – Asynchronous JavaScript and XML

UI/UX – User Interface / User Expirience

HTTP – HeperText Transfer Protocol

JSON – JavaScript Object Notation

API – Application Programming Interface

WCAG – Web Content Accessibiility Guidelines

TLS – Transport Layer Security

SQL – Structured Query Language

CSRF – Cross-Site Request Forgery

SPA - Single Page Application

ВСТУП

Актуальність: У сучасному цифровому світі все більше користувачів прагнуть організувати свої особисті речі, зокрема колекції, в зручний та ефективний спосіб. Потрібність розробки вебзастосунку для управління персональною колекцією іграшок обумовлена зростанням потреби у систематизації особистих даних, збереженні інформації про колекційні предмети, а також можливості їх цифрового представлення, зберігання й обміну. Особливо це стосується людей, які серйозно займаються колекціонуванням плюшевих іграшок і хочуть вести облік своїх експонатів, фіксувати їх характеристики, унікальні властивості, історію походження тощо.

Об'єктом дослідження є процес обміну інформацією про персональні колекції користувачів.

Предметом дослідження є web-застосунок з взаємодії користувачів з об’єктами персональних колекцій.

Метою роботи спростити взаємодію користувачів для обміну елементами персональних колекцій плюшевих іграшок.

Методи дослідження аналіз предметної області, вивчення аналогічного програмного забезпечення, порівняльний аналіз функціональних можливостей, проектування бази даних та архітектури системи, реалізація за допомогою технологій ASP.NET Core MVC та Entity Framework Core, а також тестування створеного рішення на основі реальних сценаріїв використання.

Наукова новизна роботи полягає у створенні універсального механізму для шаблонного представлення даних колекцій з можливістю їх публікації, модерації та обміну у форматі, наближеному до професійного обліку. Це забезпечує баланс між простотою використання для звичайного користувача та розширеною функціональністю для досвідченого колекціонера або адміністратора спільноти.

Практична значущість результатів полягає у можливості застосування розробленої системи не лише для ведення особистої колекції іграшок, а й для інших

типів колекційних предметів, що робить застосунок універсальним інструментом у сфері цифрового колекціонування. Крім того, зручність інтерфейсу, наявність ролей користувачів, функції перевірки, спільного доступу та централізованого управління дозволяють інтегрувати систему у ширші організаційні або освітні процеси.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Особливості управління особистою колекцією

Управління особистою колекцією плюшевих іграшок являє собою задачу систематизації та класифікації елементів особистої колекції. Колекція, виходячи з концептуальних особливостей, це поняття розкривається як комплексний підхід, метод ведення записів, призначених для зберігання інформації про всі предмети, включно з їхніми властивостями, шаблонами, характеристиками, унікальністю, особливостями та іншими атрибутами.

Основні особливості цієї проблеми включають:

– стиснення та структуризація великої кількості інформації про кожну іграшку (виробник, рік випуску, матеріал, наповнювач, зношеність, обмеженість серії і колекції, вартість, емоційна цінність);

– костомізація інтерфейсу, сучасність та інтуїтивна доступність менеджерів всіх об’єктів колекції для користувачів, включно з особами без знань про Персональний Комп’ютер та цифрові інтерфейси;

– захист персональної інформації користувача, включно з контрольованим публікуванням колекцій, підтримкою приватності, механізми авторизації та підтвердження змін;

– можливість масштабування та подальшого розширення системи, з урахуванням потреб колекціонерів або великої групи користувачів, які спільно працюють над одним об’єктом;

– підтримка мультимедійних елементів (наприклад фотографій іграшок), які зберігають естетику колекцій та надають візуальну складову кожному об’єкту, що може позитивно впливати на оцінку додатку в рамках широкої аудиторії і поширюватись у суспільстві поза межами цільової аудиторії. На рисунку 1.1 зображено типові робочі процеси які ілюструють життєвий цикл програмного забезпечення для управління особистою колекцією, включаючи облік, обслуговування, огляд та аналітику капіталу [1].

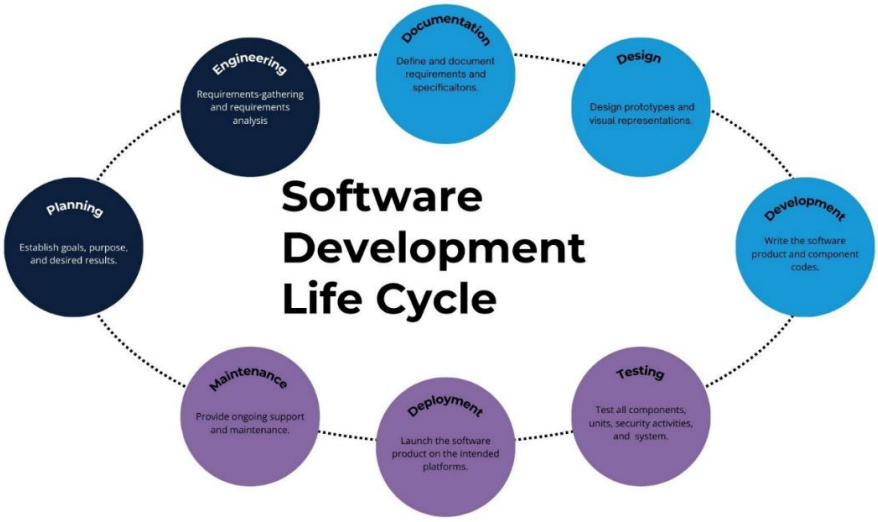


Рис. 1.1 Загальний життєвий цикл програмного забезпечення для управління особистою колекцією

З точки зору інженерної реалізації web-додатка для управління особистою колекцією іграшок важливою фазою є моніторинг і контроль, який забезпечить стабільність, безпеку і відповідність діяльності політик системи своїм користувачам. Найближчим часом після реалізації основних функцій і успішного запуску в експлуатацію додатку подібного роду виникає задача впровадження механізмів, які дозволять контролювати якість даних, автентичність зображень іграшок та запобігання небажаним або помилковим діям в системі.

Ключовою задачею цього етапу є постійний контроль коректності дій звичайного користувача. Контроль повинен бути реалізований з максимальною прозорістю: найкращим рішенням буде вести повний журнал дій (на рівні розробників), який дозволить у будь-який момент здійснити аудит активності та з’ясувати причини порушення. Це також допомагає у формуванні розуміння вразливих та недостатньо зрозумілих політик і правил системи, і подальшій оптимізації логіки й правил.

У системі передбачено два рівні користувачів:

– звичайний користувач – може створювати й редагувати елементи колекції, завантажувати зображення, ділитися своїми збірками з іншими, але діє в межах

чітко визначених шаблонів і правил, що мінімізує ризик допуску помилок і спрощує взаємодію особливо для новачків;

– адміністратор – здійснює загальний контроль над дотриманням вимог, має повноваження затверджувати або блокувати публікації.

Передусім важливо, щоб адміністрування було легким, прозорим,

ефективним і дозволяло швидко адаптувати нових модераторів. Це досягається шляхом створення незмінних шаблонів даних, примітивних інтерфейсів керування та неможливість самоврядування у контексті модерації. Таким чином запровадження системи моніторингу та контролю трансформує web-застосунок із простої утиліти для ведення особистої колекції на повноцінну, структуровану платформу, що забезпечує відповідність встановленим правилам, високий рівень безпеки та ефективне управління цифровими колекціями іграшок. На рисунку 1.2 надана спрощена діаграма потреб.

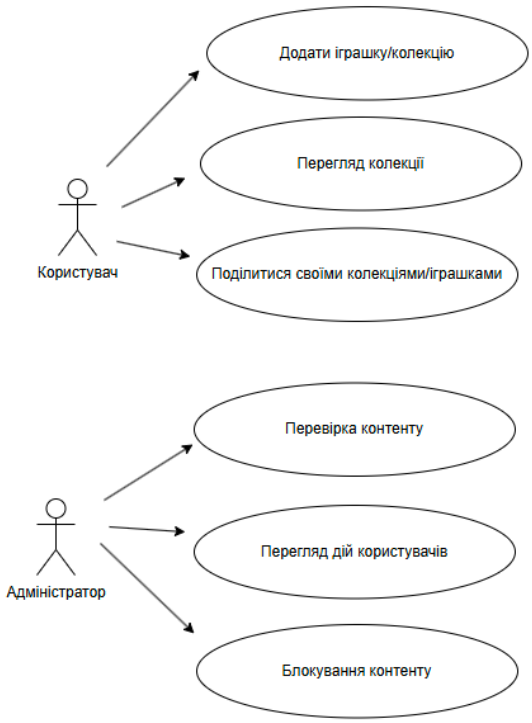


Рис. 1.2 Спрощена візуалізація основних потреб користувачів та адміністраторів.

1.2 Аналіз аналогів

На сучасному ринку етапі розвитку інформаційних технологій на ринку представлено широкий спектр інструментальних засобів, метою яких є ведення, структуризація та управління особистими і колективними цифровими колекціями. Для ефективного проектування власного програмного забезпечення спеціалізованого під потреби певної групи людей, доцільно здійснити глибокий аналіз наявних рішень у цій сфері, розглянувши їх функціональні характеристики, переваги, обмеження та практичні аспекти використання. Такий підхід сприяє виявленню найбільш оптимальних методик реалізації, визначенню актуальних потреб цільової аудиторії та уникненню типових недоліків, поширених серед існуючих систем.

У цьому підрозділі представлено зовнішній огляд поширених інструментальних засобів, що використовують для організації та управління цифровими колекціями. Особливу увагу приділено таким критеріям, як рівень кастомізації, можливості адміністрування, підтримка автоматизованих процесів, наявність шаблонів, складність впровадження, реалізація ролей користувачів та механізмів контролю доступу, а також способи розгортання програмного продукту.

1.2.1 Libib

Libib [2] - це онлайн-сервіс для каталогізації та управління особистими або організаційними медіаколекціями, орієнтований на бібліотеки, навчальні заклади, організації та індивідуальних користувачів. Ця платформа підтримує створення різнопланових колекцій і дозволяє працювати з книгами, фільмами, музикою та відеоіграми. Серед основних переваг, через які вибір користувача может бути саме це програмне забезпечення є тегування, ведення нотаток, можливість ділитися колекціями з іншими користувачами. Libib користується тарифними планами, для адаптації сервісу під різні потреби. Крім того Libib пропонує мобільний застосунок із функцією сканування штрихкодів що значно спрощує процес додавання нових

елементів до колекції. На рисунку 1.3 зображений приклад оформлення візуального рішення за контролем колекцій в застосунку Libib.

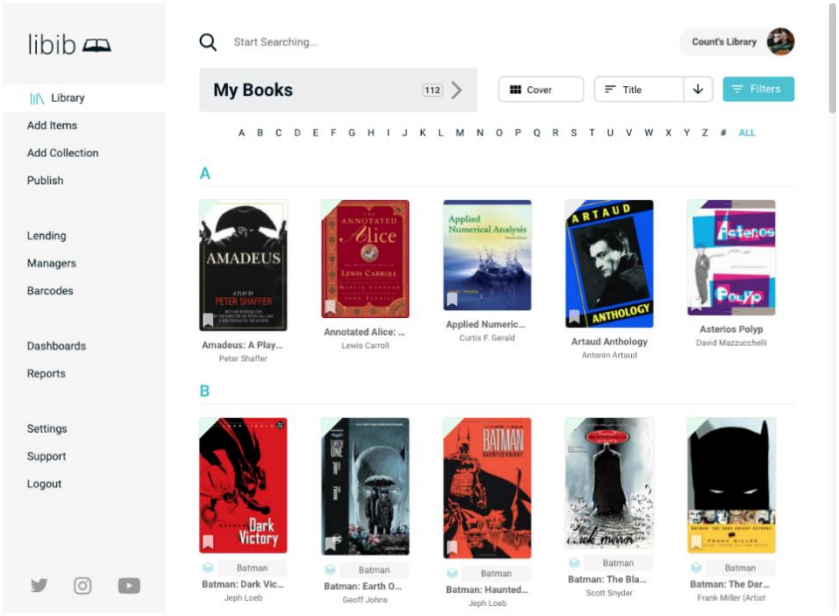


Рис. 1.3 Приклад реалізації сторінки в Libib

На сьогодення можна виділити такі особливості у застосунку Libib:

– імпорт колекцій за штрих-кодом. Надає можливість значного прискорення створення каталогів;

– каталогізація за типами. Дозволяє створити різні типи колекцій в одному акаунті;

– онлайн-синхронізація через web та мобільний застосунок. Зручний інструмент для користувача, дані доступні з будь-якого пристрою;

– можливість поділитися колекцією за допомогою посилання.

Переваги:

– швидкість додавання елементів за допомгою сканера. При роботі з великим обсягом даних, ця функція може економити години праці;

– розподіл за типами колекцій. Структуризація дозволяє мати колекції з різних напрямків;

– наявність мобільного застосунку. Зручно використовувати віддалено від стаціонарного комп’ютеру.

Недоліки:

– немає підтримки адміністративної модерації або схвалення колекцій. Це важливий аспект для контролю якості, а також безпеки користувачів від потенційно небажаних поширюваних даних;

– обмежена кастомізація колекцій. Кожен елемент має фіксований набір полів, через це немає можливості створювати власні типи атрибутів за необхідності;

– відсутність експорту даних. Немає можливості створювати контрольовані для розповсюдження документи для будь-якого користувача мережі інтернет.

На підставі вищевикладеного можна зробити висновок, що Libib – це хороший інструмент для пасивного зберігання колекції. Інструмент подібного роду допомагає організувати та показати, що ти маєш. Але функціонал не сприяє для активного керування спільнотою або динамічної роботи з контентом.

1.2.2 Pinterest

Pinterest [3] – даний інструмент є соціальним web-сервісом з функціональністю фотохостингу, що надає користувачам можливість завантажувати зображення, організовувати їх у тематичні колекції та надсилати іншим користувачам. Основою успішності даного сервісу є орієнтація на візуальний контент і створення особистих або публічних галерей, що сприяє активності соціальної взаємодії споживачів. Такий підхід дозволяє ефективно формувати цифрові добірки за інтересами, підкріплюючи зручними механізмами категоризації та візуального представлення інформації. Також платформа підтримує право користування колективної роботи над дошками, що робить її зручною в використанні спільних проектів чи обговорень. Інтегрованість програми з браузером та мобільним додатком забезпечує швидкий доступ до збереження матеріалів із будь-якого джерела та в будь-якому місці. Це значний плюс до функціональних можливостей сервісу в межах персонального чи професійного користування.

На рисунку 1.4 зображений візуальний інтерфейс застосунку Pinterest.

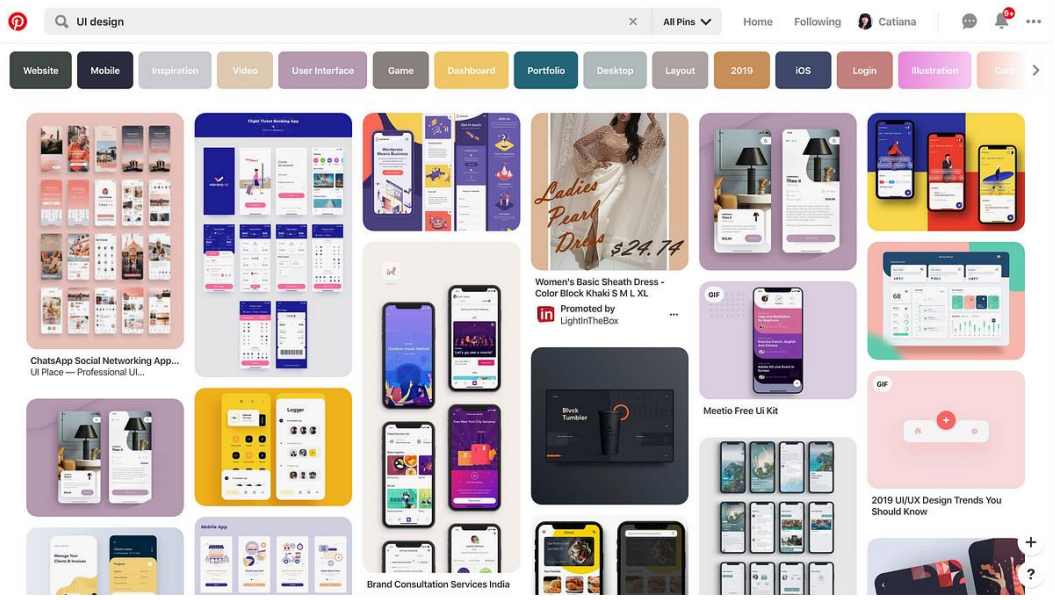


Рис. 1.4 Візуальний інтерфейс застосунку Pinterest

Помітні особливості Pinterest, які можна виділити:

– колекціонування зображень у вигляді пінів. Можливість зберігання будь-яких картинок з власного комп’ютера чи з інтернету;

– соціальна компонента. Можна підписуватися на інших учасників, ставити вподобання, коментувати та ділитися публікаціями;

– рекомендаційна система. Розроблено інноваційне програмне забезпечення, що пропонує схожі зображення на основі вподобань користувача.

Переваги:

– візуально привабливий інтерфейс. Платформа фокусує користувачів на естетиці, що відповідає сучасним міркам стилю;

– наявність великої спільноти. Велика кількість людей активно поширює свої колекції, зображення та надихає інших.

Недоліки:

– відсутність структури або категорій у межах однієї дошки(колекції); Неможливість створення додаткових полів ускладнює деталізовану класифікацію.

– неможливість створення приватних систем із контролем доступу. Реалізації механізмів на кшталт модерації або управління рівнем доступу до поширюваних дошок(колекцій);

Опираючись на вище перелічені факти, можемо дійти висновку що Pinterest чудовий візуальний інструмент зберігання окремих зображень, або одно ідейних фото. Програма має успіх у соціумі завдяки активній спільноті, але не підходить для серйозної каталогізації об’єктів з багатьма характеристиками. Йому бракує гнучкості у структурі даних, модерації контенту, а також експорту фото поза межами застосунку.

1.2.3 Sortly

Sortly [4] – сучасний інструмент управління запасами, орієнтований на простоту використання та доступність з усіх пристроїв. Функціонал включає можливість створення інвентаризаційних списків, систематизацію елементів, а також моніторинг залишків у реальному часі. Корисний застосунок для ведення цифрових каталогів персональних колекцій завдяки гнучкості, мобільності та зручним механізмам візуального представлення одиниць зберігання. На рисунку 1.5 надано візуальне представлення інтерфейсу

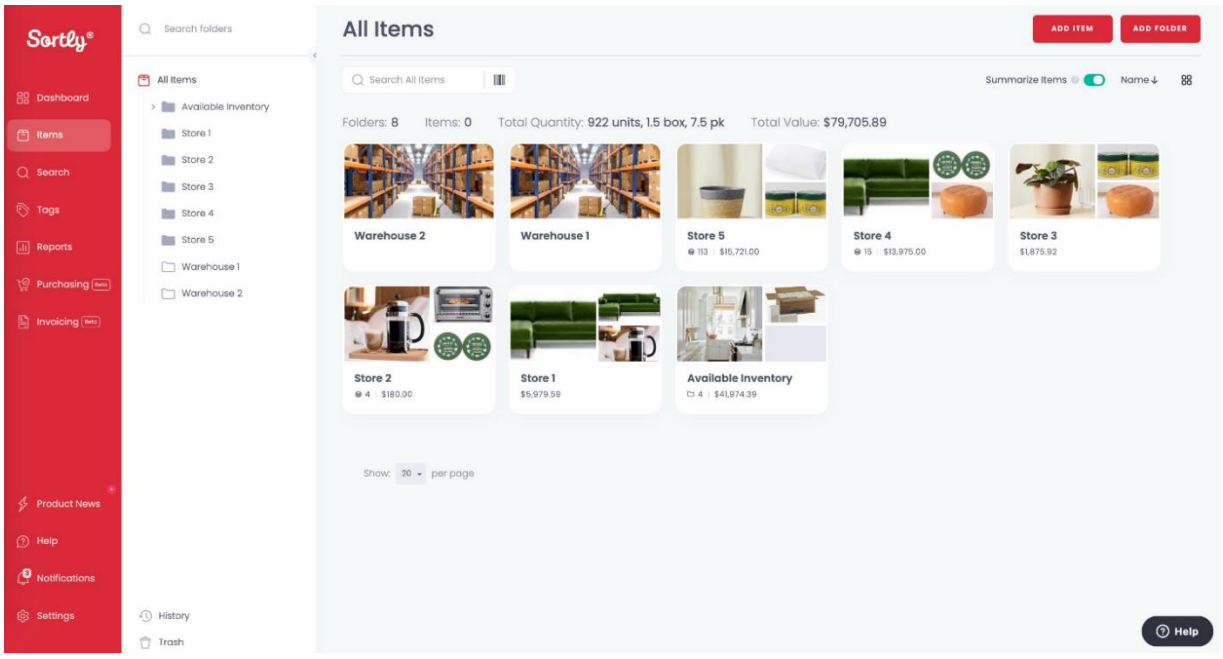


Рис. 1.5 Візуальний інтерфейс застосунку Sortly

На сучасному етапі розвитку застосунку можна відмітити особливості:

– система додавання фото, описів, тегів, серійних номерів, вартості, тощо. Додаток дозволяє дуже ретельно та детально описати кожен об’єкт;

– гнучкий пошук і фільтрація. Наявна система швидкого пошуку за атрибутами предметів;

– підтримка QR-кодів та штрих-кодів. Є можливість згенерувати QR для кожного предмета і зчитувати його за допомогою смартфона;

– хмарна синхронізація. Дані доступні з кількох пристроїв, і можуть бути спільними між користувачами (але за коштовний внесок).

Переваги:

– інвентаризаційний підхід. Прекрасний додаток для ведення ручного обліку речей, і систематизованого підходу до фільтрації предметів;

– вдало спроектована структура. Завдяки вкладеній організації легко підтримувати порядок у великих колекціях.

Недоліки:

– основний функціонал стає доступний лише після купівля платної версії. Безкоштовний функціонал дуже обмежений і не надає змоги повноцінного поглиблення у використання;

– немає соціального аспекту. Неможливо ділитися предметами або колекціями з іншими безпосередньо через додаток;

– немає механізму модерації чи перевірки. Всі дії виконуються самим користувачем без додаткових обмежень і контролю.

Після проведення аналізу цього додатку можна зробити висновок, що безумовно інструмент подібного роду спеціалізований більше для обліку і ділового використання. Йому бракує функціоналу що стосується публічного доступу до окремих колекцій або предметів з перевіркою. Доступний функціонал без внесення коштів є замалим, а спосіб зберігати тут свої колекції є недоречним. Крім того, інтерфейс платформи орієнтований передусім на професіоналів, такий підхід ускладнює його використання для пересічного користувача. Відсутність можливості візуального представлення об’єктів колекції, у форматі інтерактивної галереї з важливими частинами опису, знижує привабливість інструменту.

1.2.4 Collector System

Collector System [5] – потужне хмарне програмне забезпечення, спеціалізоване під управління колекцій мистецтва, антикваріату та інших цінностей, яке орієнтоване як на приватних користувачів так і на професійні установи – музеї, фонди, галереї. Система забезпечує надійний безпечний доступ до цифрових записів колекції з пристрою котрий має доступ до Інтернету, що дозволяє гнучко керувати об’єктами, додавати та корегувати записи, організовувати дані та ділитися інформацією з іншими фахівцями або споживачами. У контексті створення системи для керування колекцій, це програмне забезпечення демонструє ефективні підходи до структуризації великих обсягів даних, захисту інформації та мультиплатформенного доступу. На рисунку 1.6 зображено візуальний інтерфейс застосунку.

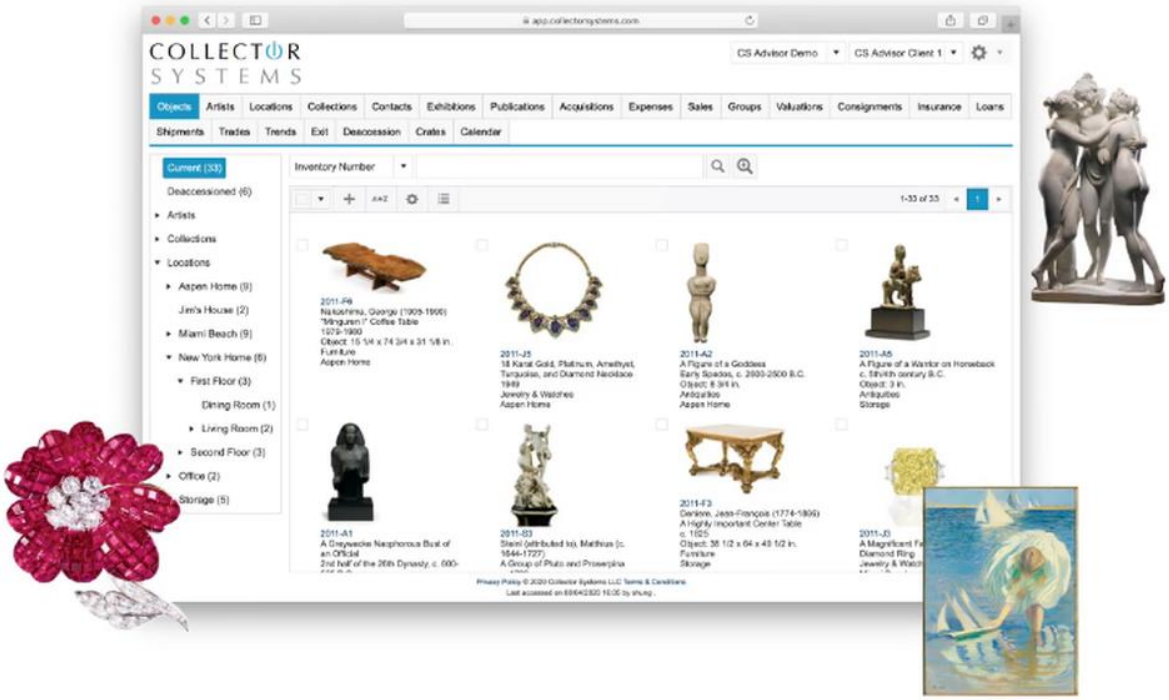


Рис. 1.6 Візуальний інтерфейс застосунку Collector System

На сьогоднішній день, у застосунку Collector System можна виділити особливості у виді:

– хмарне рішення для управління колекціями. Сервіс орієнтований на власників антикваріату, мистецтва та дорогих об’єктів;

– мультимедійна підтримка. До кожного об’єкта можна додати зображення, документи, відео, посилання;

– контроль доступу. Існує реалізація обмеження, кому і яку інформацію можна дивитись;

– інтеграція з іншими інструментами. Має комбінований підхід до можливостей, наприклад реалізація систем оцінки або інвентаризації.

Переваги:

– гнучка система прав доступу. Дуже важливий аспект для колекцій із конфіденційними даними користувачів, котрі мають коштовні об’єкти;

– висока деталізація. Можна зберігати будь-які типи даних, починаючи від фотографій до юридичних документів.

Недоліки:

– преміальний сегмент населення. Обмежена кількість користувачів має потребу на програмне забезпечення такого роду, а також можливість дорогої підписки для використання самого застосунку;

– відсутність механізму публічного показу об’єктів. Хоч і можна налаштовувати обмеження щодо доступу до колекцій, відсутня можливість публікації у загальний доступ контенту.

Опираючись на проведений аналіз застосунку можна зробити висновок, що це потужна система для великих серйозних колекцій, проте застосунок не є дружнім по відношенню до звичайних користувачів. Активний соціальний компонент не передбачається, також відсутня можливість зручно ділитися колекціями.

1.3 Порівняння аналогів

Зведені результати аналізу характеристик розглянутих додатків наведено у таблиці 1.1

Таблиця 1.1

Зведені результати аналізу характеристик додатків для колекціонування

Параметр оцінювання	**Libib**	**Sortly**	**Collector System**	**Pinterest**	**ToyCollectionApp**
Створення окремих колекцій	+, базова функція	+, повна підтримка	+,гнучка структура	+, через дошки	+, з чіткою ієрархією
Публічний перегляд	+, за посиланням	+, має спільний доступ	частковий	+, є основною функцією	+, після проходження модерації
Можливість модерації чужого контенту	-, немає	-, немає	-, немає	-, немає	+, через роль адміністратора
Експорт у PDF	-, відсутній	+,частково для інвентарю	+, є професійний рівень	-, відсутній	+, якщо публікацію схвалено адміном
Генерація QR-коду	-,відсутня	+,часткова для інвентарю	+, є як частина звітності	-,відсутня	+, є унікальний код для кожної публікації
Простота освоєння	-середній рівень складності	-середній рівень складності	-,рівень професійний складний	+ легкий рівень складності	+, легкий рівень складності

На основі проведеного порівняльного аналізу характеристик з п’яти програмних засобів, призначених для ведення колекцій різного типу, можна зробити декілька узагальнюючих висновків щодо їх функціональних можливостей,

реалізації підходів, орієнтації на користувача та відповідності потребам колекціонерів.

По перше, у сфері створення окремих колекцій усі розглянуті інструменти мають базову чи повну підтримку, проте рівень деталізації та структура організації суттєво різниться. Так наприклад Sortly та Collector System забезпечують гнучкість структуризації та орієнтовані насамперед на професійне або інвентаризаційне використання, тоді як Pinterest реалізує цю можливість через використання візуальних «дошок». У власному додатку така функціональність реалізується через чітку ієрархію що дозволяє впорядкувати об’єкти в логічні категорії.

По друге, аналіз демонструє неоднорідність підходів у застосунках до можливостей публічного доступу. Libib та Sortly надають доступ за посилання або у вигляді спільного доступу, однак не мають засобів перевірки вмісту перед публікацією. Collector System обмежує публічний доступ, що знижує його придатність для відкритих цифрових колекцій. Pinterest має повний безконтрольний функціонал до поширення будь-якого роду об’єктів. Власний додаток вирізняється можливістю публічного перегляду та поширенню після проходження модерації, що забезпечує прозорість і якість контролю за вмістом.

Третю важливу відмінність грає підтримка механізмів модерації, Жоден із представлених інструментів, окрім власного, не забезпечує такої можливості. Це свідчить про те, що орієнтація більшості представлених сервісів зосереджена або на індивідуальному зберіганні, або на внутрішньому використанні в межах одної організації. Реалізація модераційного механізму через роль адміністратора у власному застосунку дозволяє створити безпечне середовище для взаємодії користувачів.

Четвертим пунктом вартим уваги слід виділити функціональності експорту, вона повністю відсутня у застосунках Libib та Pinterest, або доступна лише з умови розширеної платної версії у застосунках Sortly та Collector System. Інструмент власного виробництва надає можливість цього функціоналу лише з умовою верифікації об’єктів, що уможливлює створення якісних документованих виглядів,

але зберігає контроль якості за сортуванням та достовірністю представленого контенту до поширення.

Окрема увага приділяється можливості генерації QR-кодів. У більшості систем є не передбаченою функцією, або подається в суто професійних тарифних планах підписки. Програмне забезпечення розроблене власноруч пропонує створення унікальних кодів для кожної публікації, що розширює функціональність для фізичного маркування об’єктів колекцій.

Нарешті, важливою складовою є простота освоєння застосунку користувачем. Виділити можна Collector System, котрий вирізняється найвищим рівнем складності через орієнтацію на фахове середовище. З іншого боку Pinterest забезпечує найпростіший інтерфейс, що підходить ширшому колу можливих користувачів порівняно з іншими. Власний застосунок намагається забезпечити легкий рівень входження та надати зрозумілий підхід до взаємодії.

Узагальнюючи проведений аналіз, можна зробити висновок, що більшість наявних рішень мають вузько спеціалізований характер або орієнтовані на візуально поширюваний контент без належного контролю. Розроблений додаток поєднує переваги обох підходів, впроваджуючи і функції модерації і ієрархічне структурування візуального представлення публікації, це робить його придатним як для особистого користування, так і для соціально орієнтованого. На рисунку 1.7 зображено класифікацію програм за професійністю та відкритістю

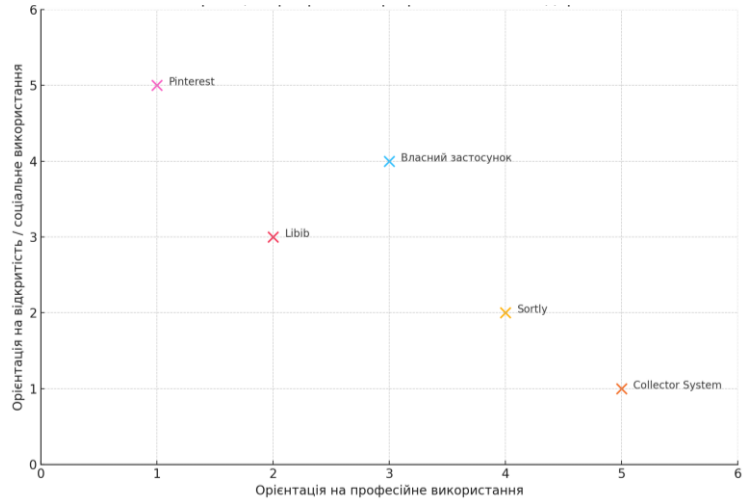


Рис. 1.7 Класифікація програм

2 ПРОЄКТУВАННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Загальні походження

У сучасних умовах поширення тенденції цифровізації у всіх сферах, програмне забезпечення (ПЗ) відіграє ключову роль у забезпеченні зручності, ефективності та автоматизації різних аспектів повсякденного життя та професійної діяльності. Зростаюча кількість користувачів прагне ділитися своєю унікальністю та поширювати інформацію про свою особистість. Представлення інформації щодо предметів власної колекції, має практичну та емоційну цінність, і допомагає людям продемонструвати відображення своїх думок та внутрішнього світу. У зв’язку з цим зростає потреба у створенні адаптивних програмних засобів, здатних задовольнити індивідуальні потреби та специфіку кожного користувача.

Проектування програмного забезпечення передбачає формалізацію вимог до його функціоналу, інтерфейсу, надійності, масштабованості та інших аспектів вимог. Важливо враховувати крім технічних параметрів, ще й контекст використання, особливості цільової аудиторії, можливі сценарії взаємодії між користувачем і застосунком. Таким чином, можна підмітити критично важливу потребу у розробці вимог, котра є невід’ємним етапом у життєвому циклі створення програмного продукту, що безпосередньо впливає на кінцеву якість, відповідність очікуванням від споживачів та конкурентоспроможності застосунку на ринку.

Вимоги до програмного забезпечення класифікуються на дві основні категорії: функціональні та нефункціональні. Функціональні вимоги описують конкретні дії, що повинна виконувати система. Наприклад: створити об’єкт, відредагувати його, переглянути дані, фільтрувати пошук за критерієм тощо. Нефункціональні вимоги встановлюють обмеження та стандарти якості до виконання функціональних вимог. Тобто ці вимоги націлені на такі поняття як:

продуктивність, безпеку, масштабованість, зручність інтерфейсу, мультиплатформенність тощо.

При проектуванні вимог основна мета, є досягнення балансу між технічним аспектом реалізації, економічної ефективності та враженнями користувачів від кінцевого продукту. Формування вимог має бути однозначним, перевірюваним, несуперечливим, повним та пріорітезованим[6]. Саме такий підхід вважається цілеспрямованим для успішного подальшого моделювання, розробки тестування та запуску програмного продукту.

У контексті створення web-застосунку для управління персональними колекціями іграшок, особливого місця набувають вимоги до інтерфейсу користувача, наявності візуальної складової при відображенні елементів та підтримки мультимедійного контенту. Також необхідно забезпечити механізми внутрішньої модерації контенту, захисту даних в цілому та управління правами доступу, що є актуальним у випадку відкритого або спільного доступу.

Одним з вирішальних компонентів залучення аудиторії до використання саме нашого застосунку, є зручність користування [7]. Сучасний користувач очікує інтуїтивно зрозумілий інтерфейс, швидке редагування, адаптивність до різних екранів та систем, можливостей персоналізації інтерфейсу.

Отже, надана вище інформація є ключем до розуміння, як задати чіткий вектор для формалізації функціональних можливостей системи, побудови архітектури, а також забезпеченню якості кінцевого продукту.

2.2 Визначення ролей та потреб

Одним із основних етапів проектування будь-якої інформаційної системи є ідентифікація основних категорій користувачів, які будуть взаємодіяти із розробленим програмним забезпеченням, а також аналіз їхніх потреб і очікувань. Чітке розуміння ролей дозволяє сформулювати більш якісні вимоги до функціональності системи, забезпечити найкращі релевантні інтерфейсні рішення і гарантувати зручність використання кінцевому користувачу. У межах

розробленого застосунку, орієнтованого на створення керування та публікацію колекцій іграшок, виокремлюю такі основні ролі:

1. Звичайний користувач – основна цільова аудиторія системи, особи які мають намір активно вести облік інформації про власні колекції, організовувати її, і за потреби ділитись з іншими. Серед потреб цієї групи:

– мінімальна кількість дій для створення колекцій з можливістю додавання зображень і характеристик;

– редагування та упорядкування вмісту;

– поширення об’єктів із контролем доступу;

– низький поріг входження до використання програми, без потреби у перегляді документації.

Для задоволення цих потреб система має забезпечити гнучкі інструменти керування колекціями, а також базовий функціонал обліку та перегляду даних.

2. Глядач (користувач ззовні) – це особи, які не є творцями контенту, але отримують доступ до публічних колекцій. Їхні потреби включають:

– реалізація швидкого доступу до публікацій за QR-кодом;

– можливість перегляду колекцій поза межами застосунку;

– приваблива візуалізація об’єктів та інформації.

3. Адміністратор – це, технічні або контенті працівники, відповідальний за контроль якості опублікованого вмісту, обробку скарг, схвалення нових колекцій, а також взаємодії з користувачами. Основні потреби представлені таким чином:

– інструменти модерації та схвалення публікацій;

– панель адміністративного керування;

– механізми блокування або видалення контенту, що не відповідає правилам.

Таким чином, чітке окреслення ролей дозволяє сформувати функціональні та нефункціональні вимоги з урахуванням різних сценаріїв.

2.3 Функціональні вимоги

У межах даного проекту функціональні вимоги подано як перелік основних можливостей, усі супроводжуються поясненням їх ролі в загальній системі.

1. Реєстрація та автентифікація користувачів.

Система має надавати користувачам можливість створювати облікові записи, авторизуватись за допомогою електронної пошти та паролю. Після входу кожен користувач отримує індивідуальний простір для втілення бажаної колекції в реальність. Ця опція є базовою передумовою персоналізації застосунку та захисту даних. Завдяки автентифікація розмежуються права доступу до вмісту. Реєстрація також дозволяє здійснювати подальше відстеження дій кожного користувача та зберігати історію взаємодій з платформою. Важливим є забезпечення безпечного механізму відновлення доступу у разі втрати облікових даних.

2. Створення та редагування колекцій.

Користувач повинен мати можливість створювати необмежену кількість колекцій, кожну з яких можна іменувати, додавати опис та категорію, редагувати або видаляти за потребою. Гнучка структура організації колекцій дозволяє користувачам групувати об’єкти за тематикою, призначенням чи іншим критерієм. Цей підхід сприяє зручності навігації та підвищує загальну ефективність роботи з системою. Підтримка редагування є необхідною для врахування можливих змін у складі або характеристиках колекцій. Можливість видалення є базовим правом користувача на контроль над своїми даними.

3. Додавання об’єктів до колекцій.

Застосунок повинен забезпечувати можливість додавання до колекцій окремих об’єктів(іграшок) з потрібним переліком характеристик, які обирає користувач. Ця функція є основою інформаційного наповнення колекцій. Введення детальних характеристик сприяє полегшенню точного обліку предметів, і є зручним не лише для відстеження наявності, а й для інвентаризації або представлення колекції іншим користувачам. Наявність зображень значно впливає на візуальне сприйняття вмісту, особливо у візуально орієнтованих категоріях.

4. Публікація колекцій для перегляду іншими користувачами.

Система має надавати можливість публікації колекцій для огляду іншими користувачами. Реалізація публікацій сприяє соціалізації користувацького досвіду та створенню спільнот за інтересами. Це також інтегрує функціональність зворотнього зв'язку або обговорення в колекціонерських колах.

5. Модерація публікацій перед відкритим переглядом.

Публікації, призначені для відкритого перегляду, мають проходити перевірку адміністратором для запобігання розміщенню неприйнятного контенту. Механізм модерації забезпечує відповідність публікацій етичним нормам та політиці web-застосунку. Модерація поширюваного контенту забезпечує запобіганню поширення шкідливої, хибної чи недостовірної інформації, що може негативно впливати на репутацію системи. Такий підхід особливо важливий для застосунків з відкритим доступом до контенту.

6. Генерація PDF-звітів з поширених колекцій.

Користувач повинен мати змогу експорту своїх колекцій та іграшок у форматі PDF, з автоматично згенерованою стилістикою та структурою файлу. Експортування у вигляді PDF-файлів надає можливість зберігати або роздрукувати об'єкти в універсальному форматі. Це відкриває перспективи для використання матеріалів в повсякденному житті, презентаціях, виставках, каталогах. Формат є стандартним для обміну документами, що підвищує практичну цінність застосунку.

7. Генерація унікального QR-коду для кожного об'єкта.

Після схвалення модератором, кожна публікація отримує унікальний QR-код для швидкого доступу з інших пристроїв. Цей інструмент забезпечує миттєвий перехід до певного ресурсу без потреби вручну шукати адресу. Це зручно для демонстрації, обміну у фізичному середовищі, тож цей підхід покращує доступність та сучасність користувацького досвіду.

8. Панель адміністратора з правами модерації.

Застосунок має містити окрему панель керування для адміністратора, без доступу для звичайних користувачів, щоб переглядати публікації і схвалювати їх

або відхиляти із поясненням причини. Адміністративна панель дозволяє централізовано впливати на контроль системи, виявляти способи зловживання додатком та підтримувати порядок. Це особливо важливо для відкритих систем, де бере участь велика кількість користувачів. Наявність таких механізмів підвищує стабільність функціонування сервісу.

2.4 Нефункціональні вимоги

Цей розділ визначає загальні характеристики системи, які не залежать від технічного або функціонального виконання засобу, але суттєво впливають на досвід користування застосунком. Нижче подано основні вимоги із обґрунтуванням їх важливості:

1. Зручність користування (usability).

Застосунок повинен мати інтуїтивно зрозумілий інтерфейс, який дозволяє користувачам без спеціальної підготовки самостійно розібратись в основних функціях системи. Вимога являє собою спрощення завантаженості користувацького інтерфейсу, мінімізацію ймовірності помилок та є важливо критичною для залучення широкої аудиторії.

2. Продуктивність (performance).

Система повинна забезпечити швидку обробку запитів користувачів, зокрема під час генерації PDF-файлів, завантаженні сторінок і фотографій. Продуктивність має пряме відношення на задоволення користувача. Повільна робота викликає втрату інтересу і знижує ефективність. Швидка відповідь системи є показником її якісної реалізації.

3. Надійність (reliability).

Застосунок має стабільно функціонувати без помилок та збоїв, включно з безпечним збереженням інформації користувача. Оскільки користувачі зберігатимуть у застосунку важливі дані про персональні колекції, втрата або пошкодження даних є неприпустимим. Надійність системи формує довіру та постійне використання програмного продукту.

4. Масштабованість (scalability).

Система повинна мати здібність до масштабування – як по кількості користувачів, так і по обсягу даних які зберігаються. У перспективі застосунок буде набирати популярність. Масштабованість забезпечить здатність системи працювати при збільшенні навантаження без необхідності її повного перероблення.

5. Безпека (security).

Повинні бути реалізовані механізми автентифікації, контроль доступу до публікацій, захист від несанкціонованого перегляду або редагування. У контексті розповсюдження особистих колекцій важливим аспектом є конфіденційність. Безпека забезпечує захист приватного простору користувача.

У результаті проведеного аналізу функціональних та нефункціональних вимог до розроблюваного програмного забезпечення було визначено коло ключових можливостей системи, а також сформульовано загальні вимоги до її якісного функціонування. Таким чином, сукупність вище перелічених вимог формує цілісну архітектуру програмного продукту, орієнтованого на якість, ефективність та безпечність взаємодії користувачів з цифровим середовищем керування колекціями[8].На рисунку 2.1 зображено різницю між функціональними та нефункціональними вимогами.

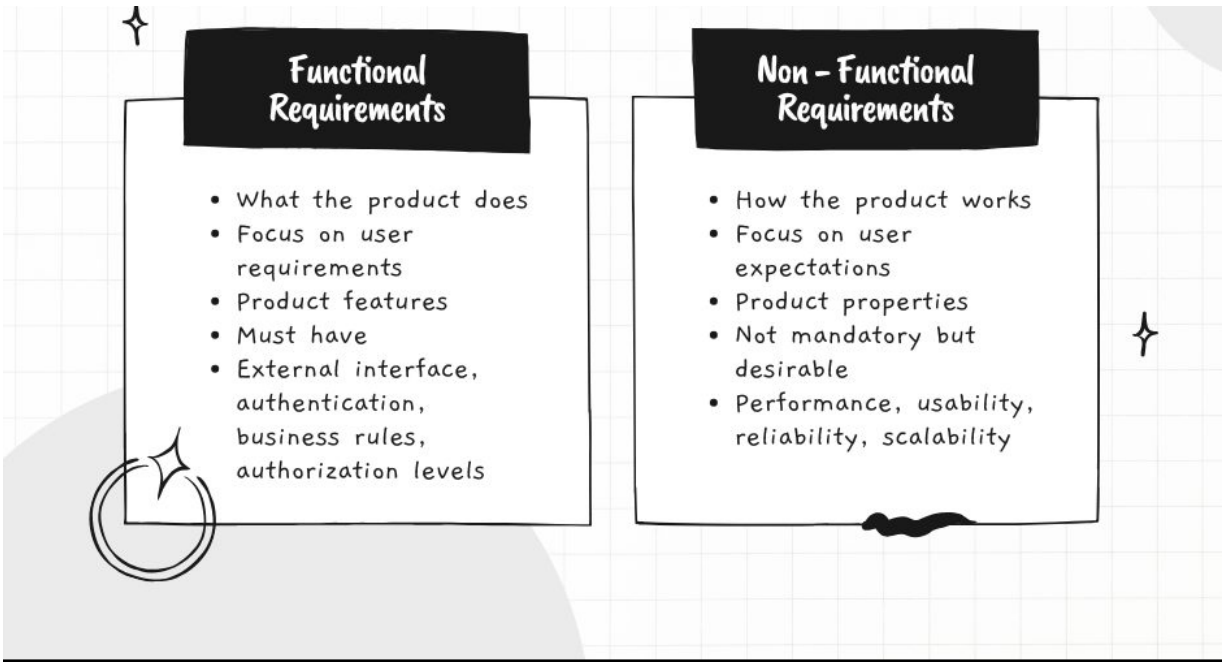


Рис. 2.1 Різниця між функціональними та нефункціональними вимогами

3 ЗАСОБИ РЕАЛІЗАЦІЇ

3.1 Огляд технологій

У процесі розробки web-застосунку важливо обґрунтовано підійти до питання підбору стеку технологій, котрий забезпечить всі вимоги створені на попередніх етапах розробки життєвого циклу програмного забезпечення. У цьому розділі буду представлений аналіз основних інструментів і фреймворків, що були використані під час реалізації програмного забезпечення для керування персональною колекцією іграшок. Кожна технологія розглядається з точки зору функціонального призначення, переваг, доцільності використання а також впливу на загальну архітектуру системи.

3.1.1 ASP.NET Core MVC

Фреймворк створений компанією Microsoft, що поєднує в собі патерн Model-View-Controller (MVC) та новітні підходи до побудови web-застосунків. Використання цього інструменти дозволяє розділяти логіку представлення, обробки запитів і роботи з даними, що значно сприяє підвищенню підтримці та тестуванню коду. На рисунку 3.1 зображено візуальне відображення MVC підходу.

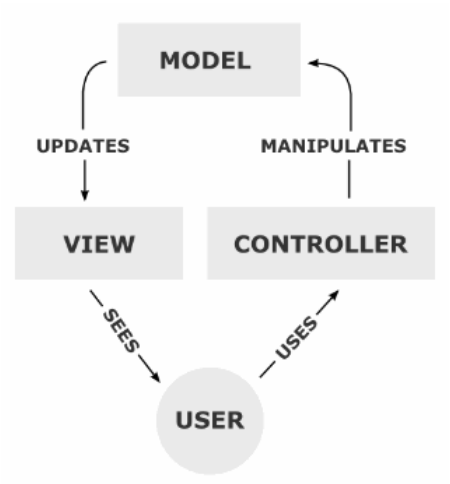


Рис. 3.1 Візуальне відображення MVC моделі.

Переваги використання:

– кросплатформеність (підтримка Windows, Linux, macOS);

– підвищення продуктивності програмного забезпечення завдяки оптимізованому потоку даних (data pipeline – автоматичні етапи, які передають дані з одного або декількох джерел до заданого місця, де з ними можна буде працювати, зберігати, аналізувати і тд.) обробки запитів;

– інверсія керування і Dependency Injection[9] забезпечують слабке зв’язування компонентів і підвищують можливість тестування та спрощують супровід;

– middleware pipeline дає змогу точно контролювати обробку HTTP запитів;

– кешування, логування, локалізація, інтегровані у фреймворк, скорочують витрати часу на реалізацію тих самих завдань;

– підтримка Docker-ready[10] забезпечує гнучкість розгорнення в храмних середовищах.

3.1.2 Entity Framework Core

Фреймворк об’єктно реляційного відображення (ORM) для .NET який дає змогу працювати з базою даних (БД) з підходами об’єктно-орієнтованого програмування (ООП). Замість SQL-запитів напряму взаємодія з БД відбувається через класи язику C# та LINQ-запити[11], що зменшує ймовірність помилок і полегшує створення системи. На рисунку 3.2 відображено спрощений приклад роботи ORM.

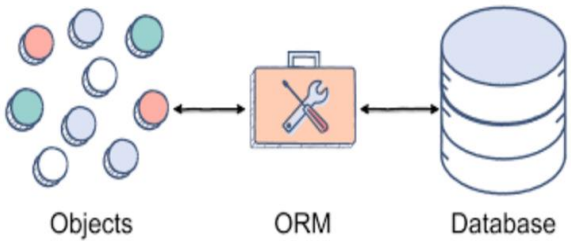


Рис. 3.2 Спрощене візуальне представлення ORM підходу.

Переваги використання:

– міграції схеми БД, дозволяє керувати змінами об’єктів без втрати даних;

– зв’язки між сутностями (One-to-Many, Many-to-Many), важливі для структуризації колекцій об’єктів[11];

– ліниве та явне завантаження (Lazy/Eager Loading), оптимізує доступ до пов’язаних даних[11].

3.1.3 Microsoft SQL Server

Являє собою систему управління базами даних (СУБД), яка зберігає дані користувачів. Вона обрана за її високу надійність, масштабованість та злагоджену інтеграцію з .NET технологіями[12].

Переваги використання:

– підтримка тригерів і процедур дозволяє реалізувати складну бізнес-логіку;

– безпека на рівні ролей і облікових записів;

– оптимізований механізм індексування забезпечує високу продуктивність при об’ємних запитах до колекцій або публікацій.

3.1.4 HTML,CSS, JavaScript

Взаємодія з користувачем та візуальна складова реалізовані за допомогою класичних web-технологій[13]:

– HTML слугує базовим інструментом розмітки сторінок. Він підтримує створювані семантичні елементи, що полегшують навігацію та покращують SEO-оптимізацію;

– CSS є інструментом для стилізації інтерфейсу. Завдяки його можливостям і зберігається адаптивність дизайну та консистентність вигляду на різних приладах;

– JavaScript забезпечує динамічну взаємодію з користувачем таким чином: валідація форм, динамічне оновлення сторінок без перезавантаження і відображення QR-кодів.

3.1.5 Bootstrap

Інструмент для створення адаптивного інтерфейсу. Фреймворк, який базується на CSS і JavaScript компонентах, з попередньо визначеними шаблонами.

Переваги використання:

– прискорює створення форм, таблиць, модальних вікон, що надає різноманітності взаємодії з користувачем..

3.1.6 QRCoder

Для генерації QR-кодів застосовується популярна бібліотека в .NET з відкритим кодом. Вона підтримує кодування довільних текстових даних у QR-код, генерацію зображень у форматах PNG, SVG та інші.

Переваги використання:

– це дозволяє швидко створювати унікальні ідентифікатори до публікацій і підвищує доступність поширення застосунку серед користувачів.

3.1.7 iText7

Ця бібліотека використовується для експорту колекцій у PDF-форматі. Вона забезпечує генерацію документів за шаблонами з можливістю наповнення динамічним контентом.

Переваги використання:

– можливість створювати професійні цифрові каталоги;

– експорт файлів PDF-формату є нефункціональною вимогою, яка дозволяє інтегрувати дані з онлайн середовища у фізичне.

3.2 Порівняння аналогів технологій

У процесі розробки web-застосунку для керування персональною колекцією іграшок, було розглянуто декілька альтернативних технологічних стеків, що використовують в індустрії для вирішення схожих web-рішень. До розгляду взято популярні зв’язки фреймворків і мов програмування.

3.2.1 MERN

Це відомий технологічний стек для комплексної розробки web-застосунків, що базується виключно на мові JavaScript(або її надмножині - TypeScript). Він охоплює всі шари побудови застосунку: від бази даних до клієнтського інтерфейсу. MongoDB виступає як нереляційна база даних, Node.js – серверне середовище використання, Express.js забезпечує серверну логіку та API, а React використовується для створення користувацького інтерфейсу [14].

Завдяки повністю орієнтованому на JavaScript середовищу розробки, підхід стеку MERN дозволяє розробникам розробляти як фронтенд, так і бекенд, використовуючи єдину мову програмування. Це пришвидшує розробку і дозволяє ефективно використовувати частини коду повторно.

Переваги:

– універсальність завдяки одній мові (JavaScript/TypeScript), команда може працювати з однаковими підходами, як на сервері так і на клієнті, це знижує складність взаємодії між розробниками та скорочує час розробки;

– активна спільнота з великою кількістю готових рішень, велика кількість open-source готових частин коду, бібліотек, статей значно полегшує пошук рішень до типових проблем;

– гнучкість у побудові API-first архітектури, Express.js забезпечує свободу у проектуванні RESTful або GraphQL API, дозволяє реалізувати складну серверну логіку;

– висока продуктивність клієнтської частини через бібліотеку React, що дозволяє створювати SPA.

Недоліки:

– MongoDB обмежується підходом NoSQL, через це база менш придатна для взаємодії з структурованими реляційними даними;

– необхідність налаштовувати велику кількість окремих компонентів для стабільної роботи системи;

– обмежена типізація без додаткових бібліотек, ускладнює розробку масштабованих систем.

Для створення web-застосунку з управління персональними колекціями іграшок, критично важливо підтримка характеристик таких як: транзакційність, надійна типізація даних, гарантована обробка запитів, контрольована структура об’єктів і взаємозв’язків. Реалізація за допомогою стеку MERN можлива, проте є недоцільно ресурсозатратно та невиправдано тяжко.

ASP.NET Core фреймворк, спочатку орієнтований на чітко структуровані, типізовані рішення з глибокою інтеграцією ORM та підтримкою транзакцій на рівні баз даних.

Таким чином, опираючись на вище перелічені переваги та тонкощі доцільніше буде використовувати обраний мною стек ASP.NET Core.

3.2.2 Laravel + Vue.js

Одним з сучасних рішень web-застосунків є стек, побудований на базі потужного PHP-фреймворку Laravel у поєднанні з фреймворком Vue.js відповідальний за візуальну частину. Широкий вибір інструментів ORM Eloquent для взаємодії з базами даних, гнучкий механізм маршрутизації, вбудовані засоби автентифікації, підтримку RESTful API, а також систему шаблонів Blade [15].

Переваги:

– низький поріг входу і простота старту проекту: Laravel добре задокументований і має безліч готових пакетів, що дозволяє швидко запускати MVP-рішення;

– шаблонізація та розділення відповідальностей: Blade-шаблонізатор, поєднує HTML логіку в мережах представлення, водночас побудова поноцінної API-архітектури, що дозволяє винести всю логіку на бекенд, а з допомогою Vue.js окремо створити інтерфейс;

– гнучке використання компонентів Vue.js, легко інтегрується в існуючі Blade-шаблони або може бути повноцінною SPA-платформою, що дозволяє адаптувати технологічний стек під конкретні потреби проекту.

Недоліки:

– менша продуктивність у порівнянні з компільованими фреймворками: PHP інтерпретована мова поступається за швидкістю виконання застосункам на базі .NET, особливо при великому навантаженні на сервіс;

– відсутність строгої типізації на рівні ядра мови: на відміну від мов з статичною типізацією, PHP не гарантує безпеку на етапі компіляції.

Laravel у поєднанні з Vue.js є ефективним вибором для розробки web-застосунків, де пріоритетом є швидкий запуск проекту, масштабування інтерфейсу та використання готових рішень. Цей стек дозволяє досягти високого рівня продуктивності на початкових етапах, забезпечити зручну роботи з базою даних, реалізувати авторизацію, моделі даних, маршрутизацію та інші ключові функції з мінімальними зусиллями.

Однак довготривала підтримка, чітка типізація, суровий контроль за бізнулогікою та високою безпекою, фреймворк на базі C# та ASP.NET Core пропонують більше гарантій на рівні компілятора та передбачуваності поведінки системи. Статична типізація, висока продуктивність та гнучкість контролю доступу, краще підходять для створення масштабованих, комерційно орієнтованих програмних рішень, тож вибір ASP.NET Core є доцільнішим.

3.2.3 Ruby on Rails

Повноцінний web-фреймворк що реалізує архітектурний шаблон MVC (Model-View-Controller) і базується на мові програмування Ruby. Однією з головних особливостей фреймворку є дотримання принципу «Convention over Configuration», що значно пришвидшує розробку, мінімізує потребу в явному налаштуванні та конфігураційних файлах. Завдяки великій кількості генераторів та автоматизованих шаблонів, фреймворк дозволяє швидко створювати CRUD-інтерфейси для роботи з базами даних та реалізовувати базову бізнес-логіку [16].

Переваги:

– структурована архітектура MVC, дозволяє чітко розділити логіку та обробку запитів, що сприяє підтримуваності та масштабованості проекту;

– висока швидкість розробки MVP, завдяки генераторам коду, інтеграції з ActiveRecord, зручній маршрутизації та простому синтаксису самого язику Ruby, фреймворк гарантує швидкий запуск перших версій застосунку;

– scaffolding, генератори коду автоматично створюють базову логіку для операцій створення, перегляду, оновлення та видалення (CRUD), що є значним прискоренням часу початкової розробки.

Недоліки:

– маленька розповсюдженість стеку, проблеми з підтримкою та кадрами;

– низька продуктивність при великих навантаженнях, через те що фреймворк не є оптимальним для високонавантажених систем або мікросервісної архітектури, через особливості інтерпретованої мови Ruby та відсутність контролю на рівні компіляції;

– відсутність статичної типізації, Ruby має динамічну типізацію, що знижує рівень перевірки типів на етапі розробки й ускладнює підтримку великих кодових баз, де важлива типова безпека.

Незважаючи на всі переваги, пов’язані з швидкістю розробки та зручністю старту, Ruby on Rails не є найоптимальнішим вибором для проекту, де потрібне розгалуження бізнес-логіки, багаторівневої автентифікації, гнучкого управління правами доступу, інтеграції з PDF-файлами та забезпеченням високого рівня безпеки. У подібному вимогливому середовищі доцільнішим є використання ASP.NET Core на базі C#.

4 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Архітектура програмного забезпечення

Архітектура програмного забезпечення є концептуальною основою, що окреслює загальну структуру системи, її основні компоненти, методи їх взаємодії та принципи організації. Вибір відповідної архітектури напряму впливає на масштабованість, підтримуваність, ефективність та безпеку розроблюваного програмного забезпечення. У контексті створення web-застосунку для управління персональною колекцією плюшевих іграшок, архітектура повинна забезпечити чіткий поділ відповідальностей, зручну інтеграцію для взаємодії з базою даних, простоту додавання нових функцій та можливість забезпечення контролю доступу [17].

У рамках цього проекту було обрано багатошарову архітектуру, зокрема реалізовану за принципом MVC (Model-View-Controller). Цей підхід забезпечує чіткий та зручний поділ логіки на три незалежні компоненти:

– Model — відповідає за бізнес-логіку програми та доступ до даних. Цей шар містить класи, що моделюють предметну галузь(іграшки, колекції, користувачі тощо), а також сервіси, які виконують операції з базою даних через ORM-фреймворк Entity Framework Core. Усі перетворення даних та перевірки здійснюються саме тут;

– View – відповідає за створення користувацького інтерфейсу. У проекті використовуються Razor Pages, що забезпечують генерацію HTML-розмітки на основі переданих даних моделі. Цей шар фокусується винятково візуалізації інформації та взаємодії з користувачем;

– Controller – посередник між View та Model. Він отримує HTTP-запити, опрацьовує введені користувачем дані, викликає відповідні методи моделі та повертає наслідок у вигляді представлення. Контролери також реалізують логіку авторизації, маршрутизації та керування потоками обробки запитів.

Додаткові архітектурні аспекти:

– інверсія керування (Inversion of Control, IoC) та втілення залежностей (Dependency Injection, DI) реалізовано через вбудовані здібності ASP.NET Core. Завдяки цьому сервіси, репозиторії та інші компоненти легко замінюються мок-об'єктами для модульного тестування, що помітно поліпшує якість та стабільність коду;

– рівень сервісів (Service Layer) — в архітектурі також передбачено проміжний рівень сервісів, який інкапсулює складну бізнес-логіку, що не повинна розміщуватись у контролерах. Це збільшує повторне використання коду та сприяє підтримці принципів SOLID;

– рівень доступу до даних (Data Access Layer) — реалізований у вигляді репозиторіїв, які абстрагують безпосередню взаємодію з базою даних. Це дозволяє легко змінювати спосіб зберігання даних без впливу на іншу частину застосунку;

– масштабованість — завдяки розподіленню обов’язків між рівнями, застосунок легко масштабувати як горизонтально (шляхом розгортання додаткових екземплярів), так і вертикально (розширенням функціональності без переписування наявних модулів);

– безпека — архітектура дозволяє централізовано впроваджувати механізми автентифікації та авторизації на основі ASP.NET Core Identity, а також реалізувати контроль доступу до ресурсів на рівні контролерів та методів;

– реакція на зміни — гнучка модульна архітектура забезпечує легке внесення змін без потреби переробляти всю систему. Наприклад, додавання нових функцій не потребує модифікації основної логіки інших шарів.

Застосування архітектури MVC дозволяє збільшити масштабованість системи, спростити супровід коду та гарантувати можливість паралельної розробки коду в різних частинах застосунку. Крім того, така структура добре узгоджується з концепцією інверсії керування (Inversion of Control) та впровадження залежностей (Dependency Injection), що у свою чергу полегшує тестування та сприяє використанню коду повторно.

4.2 Структури даних та бази даних

У процесі розробки web-застосунку було спроектовано та втілено реляційну модель бази даних, яка забезпечує ефективне збереження, організацію та обробку відомостей про користувачів, іграшки, колекції, доступи та допоміжні об’єкти. Як СУБД було використано Microsoft SQL Server, а для доступу до неї застосовано Entity Framework Core – ORM-фреймворк, що забезпечує об’єктно-реляційне відображення моделей та автоматичне створення міграцій [18]. На рисунку 4.1 зображено ER-діаграму мого застосунку.

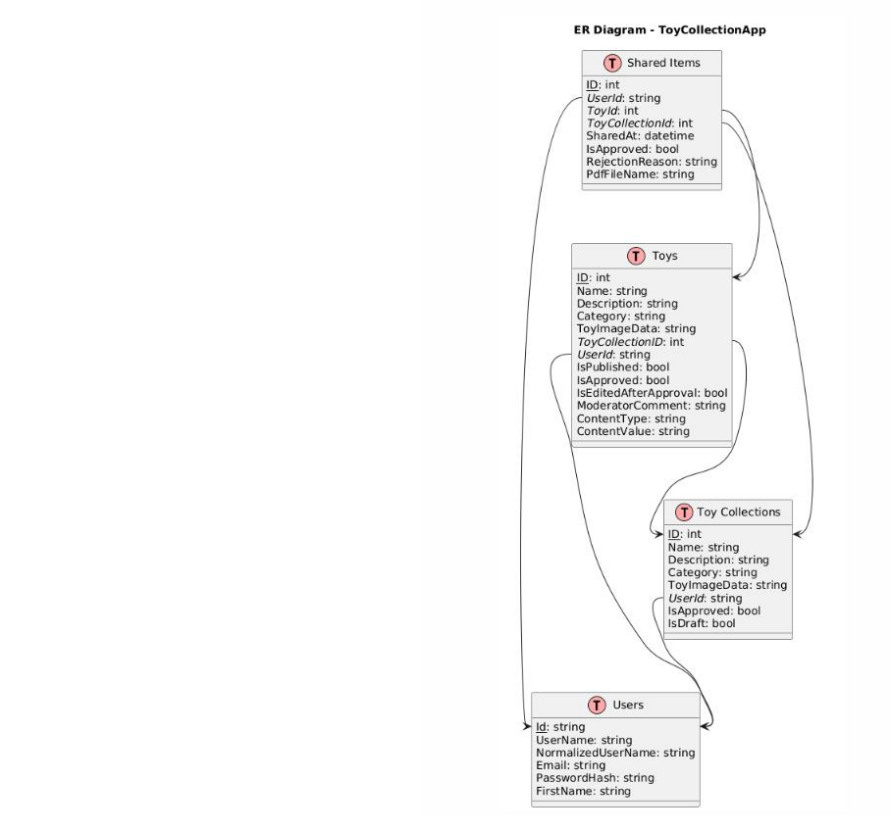


Рис. 4.1 ER-діаграма бази даних розробленого застосунку.

Кожна сутність у системі відповідає окремій таблиці в базі даних, а зв’язки між ними реалізовано за допомогою зовнішніх ключів, що дозволяє підтримувати цілісність даних на рівні Системи Управління Базами Даних (СУБД). Основними сутностями є користувачі (Users), іграшки (Toys), колекції (ToyCollections) та спільний доступ (SharedItem). Для впровадження функцій автентифікації та

авторизації використано стандартні таблиці, що входять до складу механізму ASP.NET Identity, включно з таблицями ролей, токенів і зв’язків між ними.

Зберігання мультимедійних даних, зокрема зображень іграшок і колекцій, реалізовано за допомогою полів типу nvarchar(max) у вигляді Base64-рядків, що дозволяє уникнути зовнішніх файлових сховищ та забезпечує цілісність контенту в межах бази. Крім того, передбачено окремі [NotMapped] поля типу IFormFile, які використовуються виключно на рівні представлення для обробки завантажених файлів у веб-інтерфейсі.

Важливою особливістю моделі є зв’язок один-до-багатьох між колекціями та іграшками, де кожна іграшка належить лише до однієї колекції (через поле ToyCollectionID), а колекція містить довільну кількість іграшок. Таким чином, на відміну від класичної схеми багато-до-багатьох, додаткова проміжна таблиця не застосовується — структура спрощена й оптимізована під потреби предметної області.

Сутність SharedItem реалізує механізм керування спільним доступом до іграшок або колекцій, дозволяючи створювати записи, що вказують, який саме об’єкт (іграшка чи колекція) було поділено, коли це було зроблено (SharedAt), чи схвалено модерацією (IsApproved), а також зберігає причину відмови та ім’я згенерованого PDF-файлу, якщо такий був сформований. Передбачена також підтримка nullable-зв’язків (ToyId, ToyCollectionId), що дозволяє гнучко застосовувати механізм спільного доступу для різних типів об’єктів.

Розробка структури бази даних виконувалася за підходом Code First [19] з використанням Entity Framework Core. Визначення моделей на рівні C#-коду та створення міграцій надали змогу забезпечити гнучке розширення структури, зручне відстеження змін і ефективне розгортання застосунку. Також застосовувалося Fluent API для точного налаштування типів зв’язків, каскадної поведінки при видаленні (OnDelete), а також визначення обмежень, наприклад Required, MaxLength або HasOne().WithMany().

Загальна структура бази даних проекту відповідає вимогам третьої нормальної форми (3НФ), що дозволяє уникати надмірності даних, зменшити ризик

аномалій при оновленні записів і забезпечити логічну роздільність предметних сутностей. Така модель підтримує цілісність, консистентність і масштабованість на рівні як додатку, так і бази даних.

4.3 Реалізація ключових функцій

Тут висвітлюється технічна реалізація найважливіших функціональних підсистем створеного web-застосунку, побудованого на платформі ASP.NET Core MVC із використанням Entity Framework Core та бібліотек сторонніх розробників для генерації PDF-документів і QR-кодів. Архітектурною передумовою стало дотримання принципу інверсії залежностей: усі контролери отримують контекст бази даних AppDbContext і необхідні служби через механізм вбудованого контейнера служб, що забезпечує слабке зв’язування компонентів і спрощує модульне тестування.

У модулі управління іграшками реалізовано повноцінний життєвий цикл об’єкта Toy. Методи Create, Details, InlineEdit та Delete використовують асинхронні виклики до контексту даних, що уможливлює неблокувальне обслуговування запитів та підвищує масштабованість сервера. Зміст моделей зв’язується з параметрами action-методів засобами прив’язки MVC; додаткова валідація здійснюється через атрибути моделей і перевірку ModelState. Для оптимізації взаємодії з інтерфейсом редагування застосовано техніку часткового оновлення: InlineEdit приймає об’єкт Toy, змінений клієнтським скриптом, і повертає результат у форматі JSON, усуваючи необхідність повного перезавантаження сторінки.[20] На рисунку 4.2 Наочно показано типову послідовність асинхронної взаємодії між контролером, сервісом і базою даних.

Рис. 4.2 Асинхронна взаємодія компонентів у ToyController.

Функції керування колекціями реалізовані аналогічно, проте у ToyCollectionsController особливу увагу приділено запобіганню проблемі «ледачого завантаження». Метод Details виконує eager-loading через Include, що гарантує наявність пов’язаних іграшок у пам’яті під час серіалізації представлення. При редагуванні здійснюється перевірка конкурентних змін за допомогою токена RowVersion, що зменшує ризик конфліктів при паралельній роботі кількох користувачів.

Підсистема спільного доступу концентрується у ShareController. Метод Index виконує двофазну обробку: спочатку видаляє прострочені записи, термін дії яких минув, а далі формує модель подання з розділенням на схвалені та відхилені елементи. Асинхронні запити GetPendingUserItems та GetItems повертають узагальнені анонімні об’єкти у форматі JSON, що забезпечує реактивну поведінку клієнтської частини. Верифікація прав доступу здійснюється порівнянням UserId, а рішення про публікацію приймається адміністраторами через методи Approve та Reject, захищені атрибутом [Authorize(Roles = "Admin")].

Генерація QR-кодів винесена у QrController. Метод GenerateQrCodeBase64 формує матричний код бібліотекою QRCoder в пам’яті та кодує результат у Base64, що дозволяє виводити зображення засобами HTML <img> без створення

тимчасових файлів. Для локальної розробки передбачено динамічну підстановку IP-адреси замість localhost, аби коди коректно працювали на мобільних пристроях у спільній мережі.

Найскладнішим з погляду візуалізації став PdfController. Метод Generate виконує маршрутизацію запиту залежно від типу об’єкта й повертає скомпонований масив byte[] змісту PDF разом із правильним Content-Type. У процесі рендерингу використано бібліотеку PdfSharpCore, що дає низькорівневий доступ до графіки сторінки. При створенні документів іграшок або колекцій зображення декодуються з Base64, після чого масштабуються з урахуванням співвідношення сторін, а текстові дані малюються різними шрифтами для заголовків, підписів і основного тексту. Алгоритм автоматично додає нові сторінки, якщо висота чергового блоку перевищує залишок вільного простору, що унеможливлює обрізання вмісту. У випадку колекції в колонтитул записується електронна адреса автора, що підвищує трасованість документів. На рисунку 4.3 зображено послідовність викликів під час генерації PDF-документів.

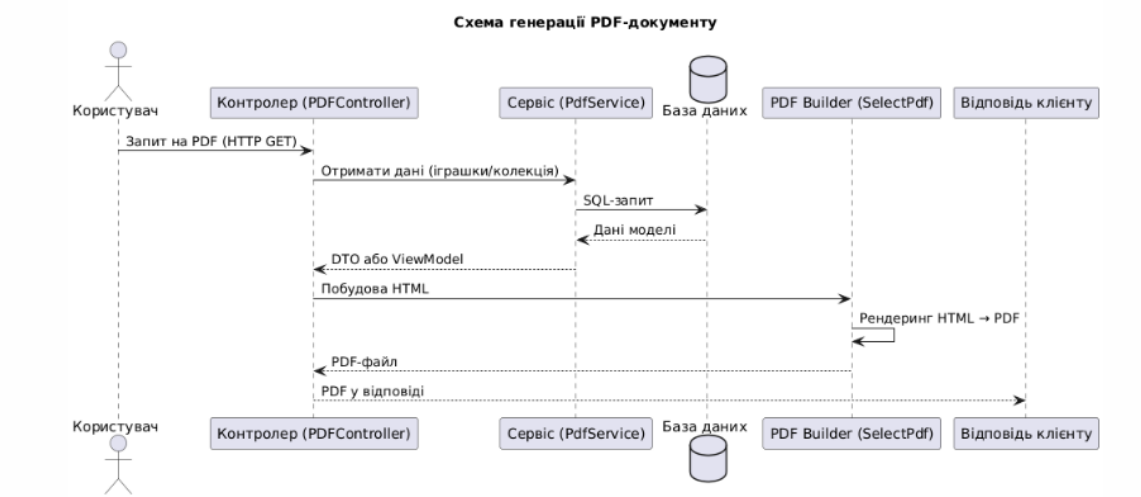


Рис 4.3 Зображення послідовності викликів при генеруванні PDF.

Обліковий модуль AccountController інтегрує стандартні засоби ASP.NET Core Identity. Аутентифікація ґрунтується на cookies-файлах, авторизація — на ролях і клеймах. Для скидання пароля реалізовано крок перевірки електронної пошти без відправлення листів: користувач вводить адресу, система перевіряє її

наявність у базі й пропонує встановити новий пароль після видалення старого хешу.

Розробка приділяє особливу увагу безпеці: у всіх змінних операціях використано атрибут [ValidateAntiForgeryToken][21], а посилання, що приймають ідентифікатори, перевіряють коректність параметрів та належність даних поточному користувачеві. Таким чином, ключові функції застосунку поєднують асинхронну обробку, строгий контроль доступу, динамічне формування мультимедійного контенту й адаптивну взаємодію з клієнтом, що разом забезпечує стабільність, масштабованість і високу якість користувацького досвіду.

У цьому підрозділі було детально розглянуто реалізацію критичних функціональних підсистем web-застосунку, включаючи управління іграшками, колекціями, генерацію QR-кодів, PDF-документів, механізми спільного доступу та автентифікації користувачів. Завдяки використанню ASP.NET Core MVC та бібліотеці Entity Framework Core гарантовано високу масштабованість і надійність. Впровадження асинхронних викликів, підтримка інверсії залежностей і захист від атак CSRF стали основою стабільної та безпечної архітектури. Інтеграція сторонніх бібліотек для мультимедійних функцій (QRCoder, PdfSharpCore) дозволила розширити функціональність без ускладнення структури проекту. Описані рішення формують технічний каркас застосунку, який дозволяє забезпечити повноцінну підтримку користувацьких сценаріїв у динамічному середовищі.

4.4 Розгортання

Процес розгортання створеного web-застосунку передбачає поетапну підготовку, конфігурацію та перенесення серверної та клієнтської складових у цільове середовище виконання. Застосунок, реалізований на платформі ASP.NET Core MVC, підтримує кросплатформенне розгортання, однак для цілей тестування та демонстрації функціональності було обрано середовище Windows Server із попередньо налаштованим IIS (Internet Information Services) у режимі зворотного проксі. Це забезпечує сумісність із наявною інфраструктурою та спрощує

адміністрування за рахунок інтеграції з Active Directory та політиками контролю доступу.

Оптимізація процесу публікації здійснюється за допомогою вбудованих засобів середовища розробки Visual Studio, яке надає можливість створення самодостатнього пакета публікації з усіма необхідними бінарними файлами, бібліотеками, конфігураційними файлами та статичними ресурсами. У конфігураційному файлі appsettings.Production.json задано параметри з'єднання з виробничою базою даних, розміщеною на окремому екземплярі SQL Server, а також увімкнено ведення журналу подій для діагностики поведінки застосунку в умовах справжнього навантаження.

На етапі розгортання особливу увагу приділено захисту конфіденційної інформації. Із цією метою секрети, такі як рядки підключення або SMTP-кредeнціали, не зберігаються безпосередньо в коді, а зчитуються з середовищ виконання через систему змінних оточення або засоби Azure Key Vault у разі хостингу в хмарному середовищі. Також впроваджено HTTPS-шифрування за допомогою сертифікатів TLS, що забезпечує цілісність та конфіденційність даних, які передаються між клієнтом і сервером.

Застосунок налаштовано на автоматичне мігрування бази даних при старті в режимі production, що дозволяє зберегти цілісність структури даних та уникнути ручного втручання адміністратора під час оновлень. Для контролю працездатності впроваджено базові health-check endpoints, що дають змогу перевіряти доступність бази даних та коректність конфігурації служби без необхідності втручання у внутрішню логіку.

У разі виявлення змін або оновлень, система CI/CD (Continuous Integration / Continuous Deployment), побудована з використанням GitHub Actions, автоматично ініціює процес перескладання проекту, тестування, створення артефактів і розгортання у тестовому середовищі. Це дозволяє скоротити час між внесенням змін до коду та їх відображенням на цільовому сервері, забезпечуючи високу гнучкість і надійність під час розробки та супроводу системи.

Таким чином, розгортання застосунку не є одноразовою процедурою, а інтегрується як органічна частина процесу розробки, що включає автоматизацію, безпечну конфігурацію, масштабовану інфраструктуру та безперервний моніторинг, що в сукупності гарантує стабільну роботу системи в умовах змінного навантаження та постійного розвитку. На рисунку 4.4 зображено процес розгортання застосунку.

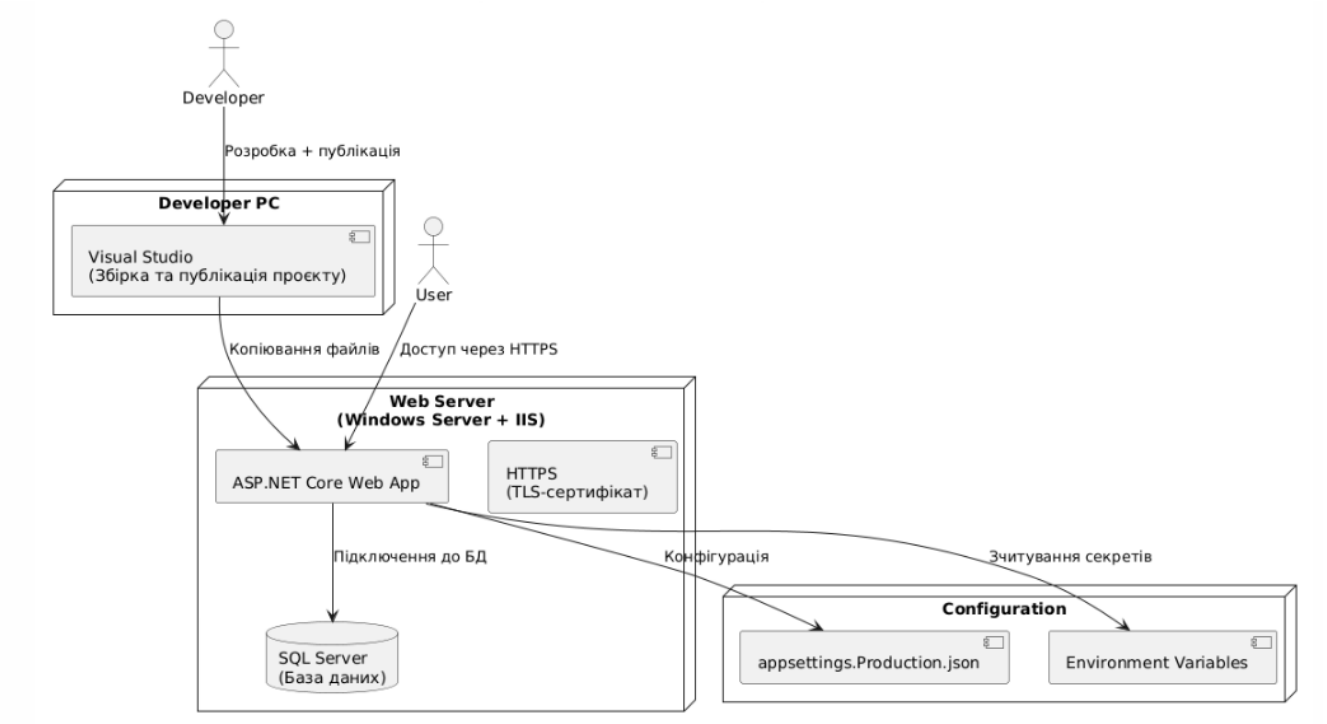


Рис. 4.4 Процес розгортання застосунку.

4.5 Інтерфейс користувача та досвід взаємодії(UI/UX)

Інтерфейс користувача відіграє визначальну роль у сприйнятті якості програмного продукту, оскільки саме він виступає єдиною точкою взаємодії користувача із системою. У створеному web-застосунку реалізація UI/UX-компонентів ґрунтується на принципах простоти, передбачуваності, інтуїтивності та естетичної відповідності сучасним стандартам дизайну. Ці принципи дозволили сформувати стабільне, зручне та технологічно ефективне середовище роботи з цифровими колекціями, яке не потребує навчання з боку користувача.

Фронтенд-сегмент реалізовано на базі HTML5, CSS3 та фреймворку Bootstrap 5, що дало змогу реалізувати адаптивний дизайн без додаткового стилістичного фреймворку. Bootstrap забезпечує уніфіковану візуальну стилістику елементів інтерфейсу — кнопок, форм, меню, таблиць, карток тощо — що підвищує загальну зручність навігації. За рахунок використання сіткової системи й медіа-запитів інтерфейс автоматично адаптується до ширини вікна браузера, коректно відображаючись як на десктопних, так і на мобільних пристроях. Під час тестування було підтверджено повну працездатність системи в браузерах Chrome, Firefox, Edge і Safari, включно з мобільними версіями.

З погляду структурної організації, інтерфейс розподілений на кілька логічних зон: верхня панель навігації (navbar), контентна область, бічні блоки (за потреби) та нижній колонтитул. Панель навігації автоматично змінює свій вміст залежно від автентифікаційного статусу користувача, надаючи доступ до персонального кабінету, загального списку колекцій, сторінки входу/реєстрації або панелі адміністратора. Усі переходи реалізовано за принципом RESTful-орієнтації: кожен запит формує унікальне посилання, що дозволяє підтримувати глибоке посилання та зберігання сесійного стану в браузері.

Інтерактивна частина інтерфейсу реалізована із застосуванням JavaScript та бібліотеки jQuery. Впроваджено механізми часткового оновлення контенту без перезавантаження сторінки (Partial View з AJAX-підтримкою), що значно зменшує навантаження на сервер та покращує сприйняття швидкодії з боку користувача.

Найбільш виражене застосування таких рішень спостерігається в реалізації функції InlineEdit, яка дозволяє редагувати атрибути іграшки без залишення поточної сторінки. Зміни передаються на сервер у форматі JSON, а оновлення відбувається асинхронно через $.ajax() з обробкою відповіді у форматі HTML-фрагмента.

Повідомлення про результат дії користувача (успіх, помилка, попередження) реалізовано через тимчасове збереження значення у TempData, що згодом виводиться як Bootstrap-alert. Такі повідомлення не лише інформують про результат, але й зберігають контекст виконаної дії, що є суттєвим для підтримки послідовної логіки взаємодії. Крім того, всі форми мають вбудовану перевірку полів через механізми валідації моделі (DataAnnotations) та відповідні JavaScript-засоби (jQuery Validation). Це дозволяє попереджати помилки ще до надсилання запиту на сервер. На рисунку 4.5 зображено життєвий цикл повідомлення у системі.

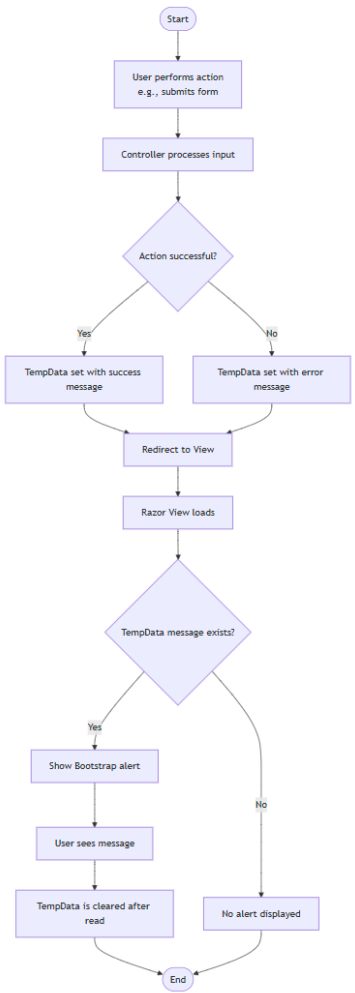


Рис. 4.5 Схема життєвого циклу повідомлень (TempData Alerts)

З метою покращення досвіду взаємодії впроваджено логіку автоматичного оновлення списків за допомогою асинхронних викликів. Наприклад, функція GetPendingUserItems повертає у форматі JSON перелік об’єктів, що очікують на затвердження, а GetItems — перелік доступних елементів для спільного доступу. Такі сценарії дозволяють будувати інтерфейси типу «живого пошуку», автоматичних підказок, динамічного фільтрування або контекстного рендерингу даних без потреби перезавантаження сторінки.

На особливу увагу заслуговує реалізація адміністративного інтерфейсу модерації. Усі запити на публікацію зібрані в одному представленні, доступному лише адміністраторам, із можливістю прийняття або відхилення кожного елемента з коментарем. Завдяки використанню Bootstrap-компонентів, таких як modal, адміністратор може переглянути деталі елемента, не покидаючи головного списку. Такий підхід сприяє оперативному опрацюванню запитів без перевантаження інтерфейсу.

Інтерфейс також адаптований для роботи з мультимедійними компонентами — зображеннями та QR-кодами. Виведення зображень реалізовано шляхом розкодування Base64-рядків у тег <img>, що дозволяє уникнути створення фізичних файлів на сервері й прискорює завантаження сторінок. QR-коди виводяться динамічно на основі отриманого JSON-об’єкта з бази64-зображенням, що генерується на льоту без збереження в файловій системі.

Щодо навігації між розділами — користувач має змогу швидко переходити між сторінками, що логічно пов’язані між собою: від іграшки — до її колекції, від колекції — до пов’язаного PDF-документа чи QR-коду. Така взаємозв’язана структура полегшує орієнтацію в системі та скорочує кількість необхідних кліків. Усі посилання інтуїтивно зрозумілі, мають читабельні адреси (/Toys/Details/5), що позитивно впливає як на користувацький досвід, так і на індексацію в разі публічного доступу.

Додатково передбачено покращення доступності (accessibility). Застосовано контрастні кольори для тексту й фону, відповідність елементів управління

розмірам пальця для сенсорних екранів, а також атрибути aria-label для ключових елементів (де це доцільно). Такі заходи відповідають базовим рекомендаціям стандарту WCAG 2.1 і створюють умови доступу для користувачів із тимчасовими або постійними функціональними обмеженнями [22].

Оцінювання користувацького досвіду здійснювалося під час тестування системи різними користувачами, які підтвердили зрозумілість навігації, швидкість роботи інтерфейсу та загальну привабливість дизайну. Усі звернення до підтримки інтерфейсної частини вирішувалися швидко завдяки структурованій організації HTML-коду та повторному використанню компонентів. Це також зменшує загальні витрати на супровід системи.

Таким чином, інтерфейс користувача створено відповідно до кращих практик UI/UX-дизайну: він є адаптивним, функціонально повним, інтерактивним і доступним. Інтеграція таких елементів, як AJAX, повідомлення зворотного зв’язку, гнучка навігація та підтримка мультимедійного контенту, дозволила досягти високої якості взаємодії між користувачем і системою, що безпосередньо впливає на загальну ефективність і прийнятність програмного забезпечення. На рисунку 4.6 зображена ілюстрація описуючу логіку взаємодії користувача з інтерфейсом, AJAX-запитами, повідомленнями, QR-кодами та динамічним оновленням.

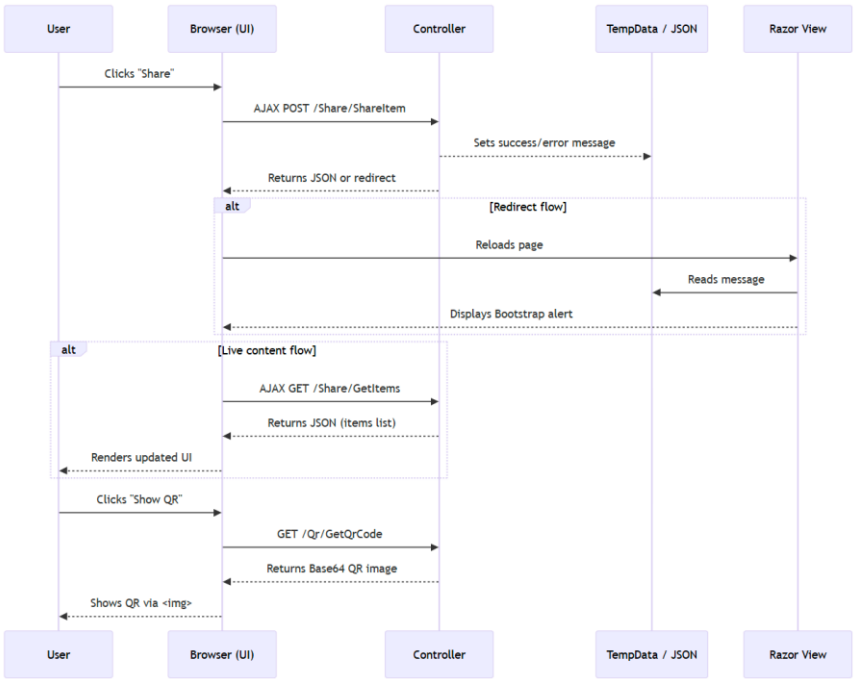


Рис. 4.6 Логіка взаємодії користувача з контентом для публікації.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи досягнуто поставленої мети — спростити взаємодію між користувачами створивши web-застосунок для управління персональною колекцією плюшевих іграшок з урахуванням вимог цільової аудиторії, технічних характеристик сучасних web-технологій і необхідності забезпечення функціональності, зручності використання та безпеки в рамках багатокористувацької взаємодії. Робота охоплює повний цикл створення програмного забезпечення — від аналізу предметної області до реалізації ключових модулів і тестування кінцевого продукту.

Одним із важливих досягнень є ґрунтовний аналіз предметної галузі, який дав змогу сформулювати чіткі вимоги до системи з урахуванням особливостей роботи з особистими колекціями та сучасних практик цифрової каталогізації. Здійснене дослідження аналогів продемонструвало відсутність комплексного інструменту, який би одночасно задовольняв потреби колекціонерів, підтримував багаторівневу модерацію, гнучке управління доступом і надавав можливості мультимедійного представлення. На підставі цього було запропоновано концепцію web-застосунку, що переосмислює підходи до колекціонування, роблячи акцент на візуалізації, безпеці, інтегрованому обміні й функціональності.

У процесі проектування програмного забезпечення визначено основні ролі користувачів — звичайний користувач, глядач і адміністратор. Для кожної з ролей сформульовано перелік потреб і поведінкових сценаріїв, що стало основою для технічного моделювання системи. Вимоги сформульовано відповідно до кращих практик інженерії програмного забезпечення, із чітким поділом на функціональні та нефункціональні, що дозволило впровадити структуровану архітектуру з високою адаптивністю до змін.

На етапі реалізації використано стек сучасних технологій: ASP.NET Core MVC, Entity Framework Core, SQL Server, HTML/CSS, JavaScript, Bootstrap, QRCoder та PdfSharpCore. Це забезпечило ефективну взаємодію між клієнтською та серверною частинами, зручний інтерфейс, можливість інтеграції

мультимедійного контенту, а також масштабованість рішення. Застосування патерна Model-View-Controller забезпечило розділення відповідальності між модулями системи, а принцип інверсії залежностей підвищив гнучкість і тестованість компонентів.

Особливу увагу приділено реалізації модераційної системи, яка забезпечує контроль за поширенням публічного контенту та підтримує відповідність вмісту етичним і функціональним нормам. Реалізація ролей з відповідними правами доступу — критичний фактор безпеки й стабільності багатокористувацького середовища. Це дозволяє підтримувати баланс між відкритістю системи й захистом інтересів користувачів.

Впровадження генерації PDF-документів та QR-кодів є вагомою інновацією з точки зору інтеграції цифрових колекцій у фізичний простір. Це дозволяє використовувати систему як інструмент не лише для персонального зберігання, але й для виставкової, демонстраційної або комерційної діяльності. Алгоритм візуалізації PDF-документів реалізовано з урахуванням автоматичного розміщення мультимедійних елементів на сторінках та дотримання стилістики, що забезпечує презентабельність і професійний вигляд результатів.

Процес розгортання системи охоплює налаштування бази даних, підключення служб автентифікації, конфігурацію параметрів публічного доступу та підготовку середовища для хостингу. Важливо, що архітектура системи дозволяє гнучко змінювати конфігурацію без потреби в повному рефакторингу, що свідчить про її життєздатність у довгостроковій перспективі. Інструментальні можливості інтеграції з хмарними сервісами створюють передумови для масштабування проекту до SaaS-рішення.

Підсумовуючи, можна стверджувати, що розроблений web-застосунок є повноцінною реалізацією інженерного підходу до вирішення проблеми цифрової організації персональних колекцій з урахуванням сучасних вимог до інтерфейсної доступності, інформаційної безпеки, адаптивності дизайну й соціальної інтеграції. Отримані результати можуть бути використані як основа для подальшого розвитку системи, впровадження мобільного застосунку, інтеграції з соціальними

платформами або розширення на інші типи цифрових колекцій (книги, моделі, артоб'єкти тощо).

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Башич В.А., Садовенко В.С. Визначення вимог до web-застосунку для управління персональною колекцією ігрових предметів (на прикладі плюшових іграшок). Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ», 24.04.25, ДУІКТ, Київ, Україна, с.144-146.

2. Башич В.А., Садовенко В.С. Розробка програмних засобів захисту інформації у веб-застосунку в процесіуправління персональною колекцією. Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с.260-261.

5 ПЕРЕЛІК ПОСИЛАНЬ

1. Arthur M. Langer Analysis and Design of Next-Generation Software Architectures. Springer International Publishing, 2020. 308 с.

2. Libib URL: https://www.libib.com

3. Pinterest URL: https://www.pinterest.com

4. Sortly URL: https: https://www.sortly.com

5. Collector System URL: https://www.collectorsystems.com

6. Waterfall Model - Software Engineering URL:https://www.geeksforgeeks.org/waterfall-model/

7. Ben S. Human-Centered AI. Oxford University Press, 2022. 416 с.

8. Ian S. Software Engineering. Pearson, 2015. 812 с.

9. Adam F. Pro ASP.NET Core 6 MVC . Apress, 2022. 1268 с.

10. Elton S. Learn Docker in a Month of Lunches. Manning, 2020. 464 с.

11. John P. Entity Framework Core in Action, Second Edition. Manning, 2021. 624 с.

12. Korotkevitch D. Pro SQL Server Internals. Apress, 2014. 804 с.

13. John D. Web Programming with HTML5, CSS, and JavaScript. Jones & Bartlett Learning, 2018. 699 с.

14. Eddy W. MERN Quick Start Guide. Packt Publishing, 2018. 302 с.

15. ASHLEY M., Anthony G. Full-Stack Vue.js 2 and Laravel 5. Packt Publishing, 2017. 376 с.

16. David B. Agile Web Development with Rails 6. Pragmatic Bookshelf, 2020. 496 с.

17. Len B., Paul C., Rick K. Software Architecture in Practice. Pearson Education, 2021. 464 с.

18. Adam F. Pro Entity Framework Core 2 for ASP.NET Core MVC. Apress, 2018. 652 с.

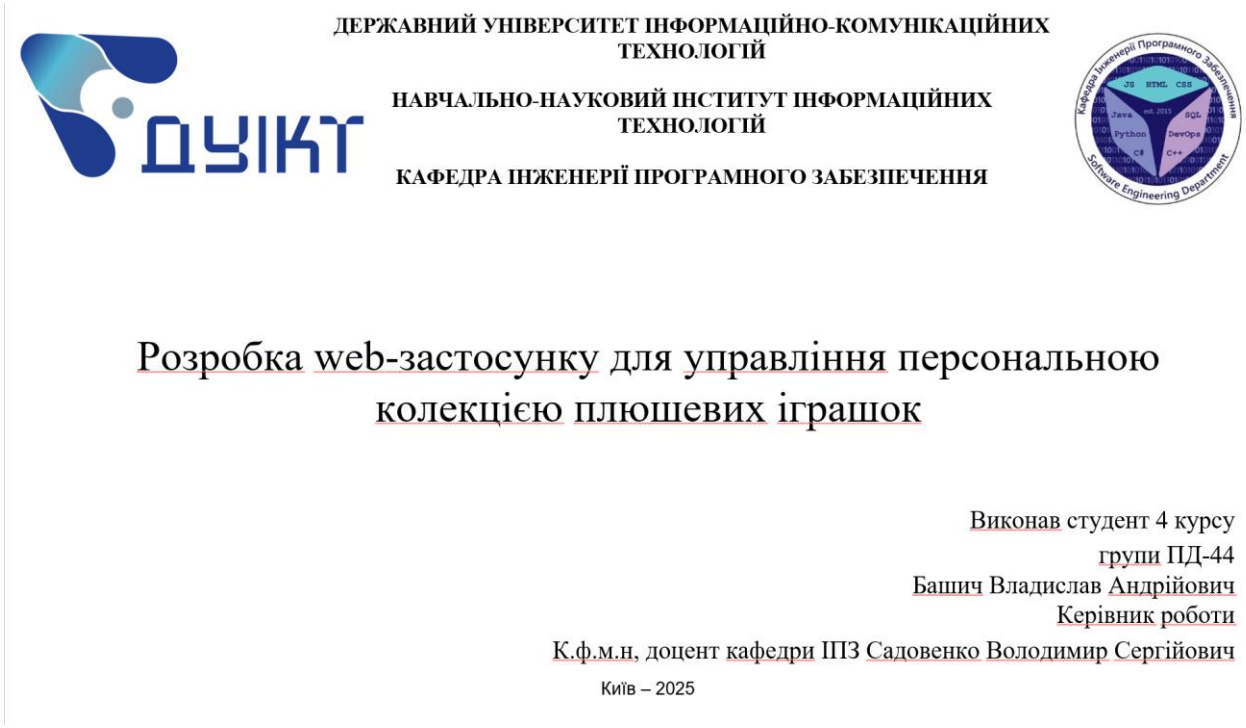
19. Romiller Reducing Code First Database Chatter URL: https://romiller.com/2014/06/10/reducing-code-first-database-chatter/

20. Andrew L. ASP.NET Core in Action, Second Edition. Manning, 2021. 832 c.

21. Christian W. ASP.NET Core Security. Manning, 2022. 368 c.

22. Regine M. Gilbert Inclusive Design for a Digital World. Apress, 2019. 272 c.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку для управління персональною колекцією плюшевих іграшок

Виконав студент 4 курсу
групи ПД-44
Башич Владислав Андрійович
Керівник роботи
К.ф.м.н, доцент кафедри ІПЗ Садовенко Володимир Сергійович

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи**: Спростити взаємодію користувачів для обміну елементами персональних колекцій плюшевих іграшок.
- **Об’єкт дослідження**: Процес обміну інформацією про персональні колекції користувачів.
- **Предмет дослідження**:web-застосунок з взаємодії користувачів з об’єктами персональних колекцій.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Огляд застосунку для управління персональною колекцією плюшевих іграшок та аналіз існуючих аналогів, визначення вимог до програмного забезпечення.

2. Огляд та аналіз засобів реалізації web-додатку для управління персональною колекцією плюшевих іграшок.

3. Проектування архітектури застосунку для управління персональною колекцією плюшевих іграшок.

4. Програмне втілення та тестування застосунку для управління персональною колекцією плюшевих іграшок.

3

АНАЛІЗ АНАЛОГІВ

Параметр оцінювання	Libib	Sortly	Collector System	Pinterest	Мій застосунок
Створення окремих колекцій	+, базова функція	+, повна підтримка	+,гнучка структура	+, через дошки	+, з чіткою ієрархією
Публічний перегляд	+, за посиланням	+, має спільний доступ	частковий	+, є основною функцією	+, після проходження модерації
Можливість модерації чужого контенту	-, немає	-, немає	-, немає	-, немає	+, через роль адміністратора
Експорт у PDF	-, відсутній	+,частково для інвентарю	+, є професійний рівень	-, відсутній	+, якщо публікацію схвалено адміном
Генерація QR-коду	-,відсутня	+,часткова для інвентарю	+, є як частина звітності	-,відсутня	+, є унікальний код для кожної публікації
Простота освоєння	-середній рівень складності	-середній рівень складності	-,рівень професійний складний	+ легкий рівень складності	+, легкий рівень складності

4

ВИМОГИ ДО ДОДАТКУ

Функціональні вимоги:

1.Реєстрація та автентифікація користувачів

2.Створення та редагування колекцій

3.Додавання об’єктів до колекцій

4.Публікація колекцій для перегляду іншими користувачами.

5.Модерація публікацій перед відкритим переглядом

6.Генерація PDF-звітів з поширених колекцій

7.Генерація унікального QR-коду для кожного об’єкта.

8.Панель адміністратора з правами модерації

Нефункціональні вимоги:

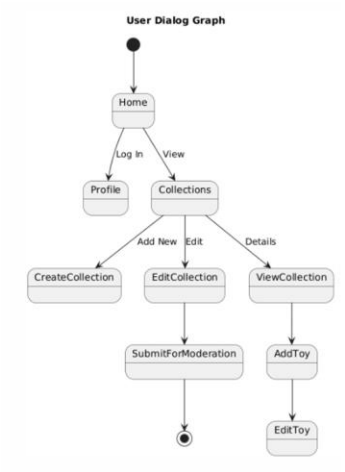
1.Зручність користування (usability)
2.Продуктивність (performance)
3.Надійність (reliability)
4.Масштабованість (scalability)
5.Безпека (security)

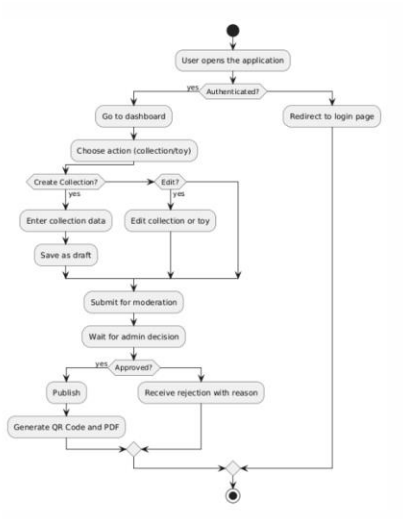
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

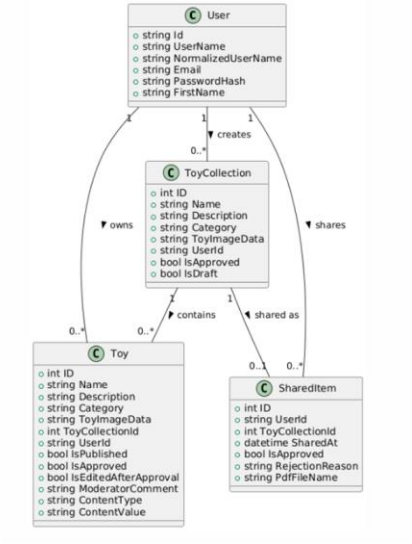
Граф діалогу та діаграма діяльності

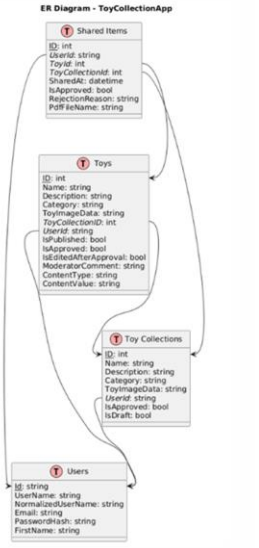




7

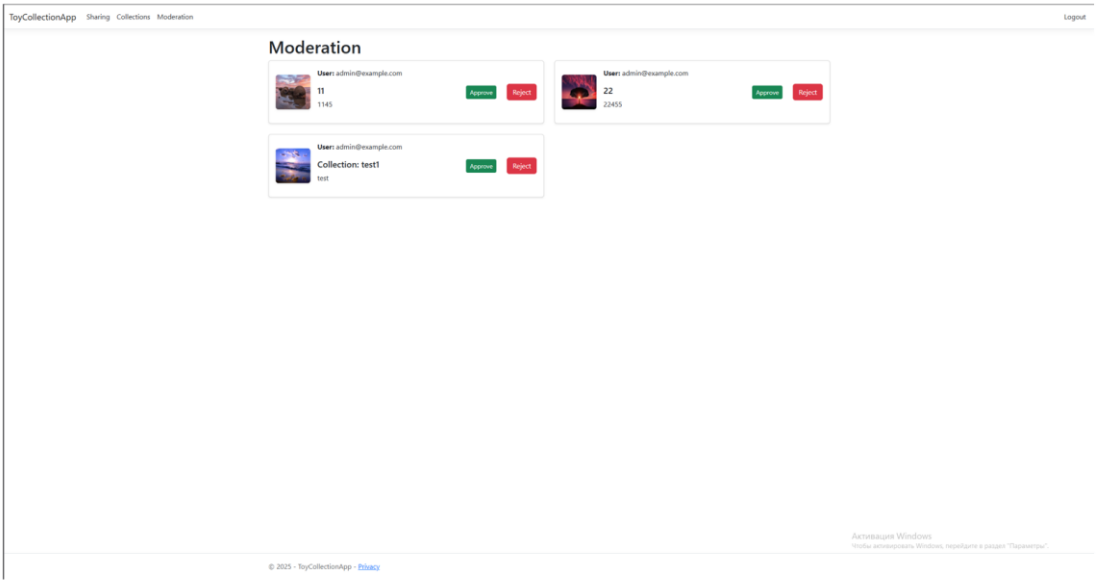
Діаграма класів та ER-діаграма





8

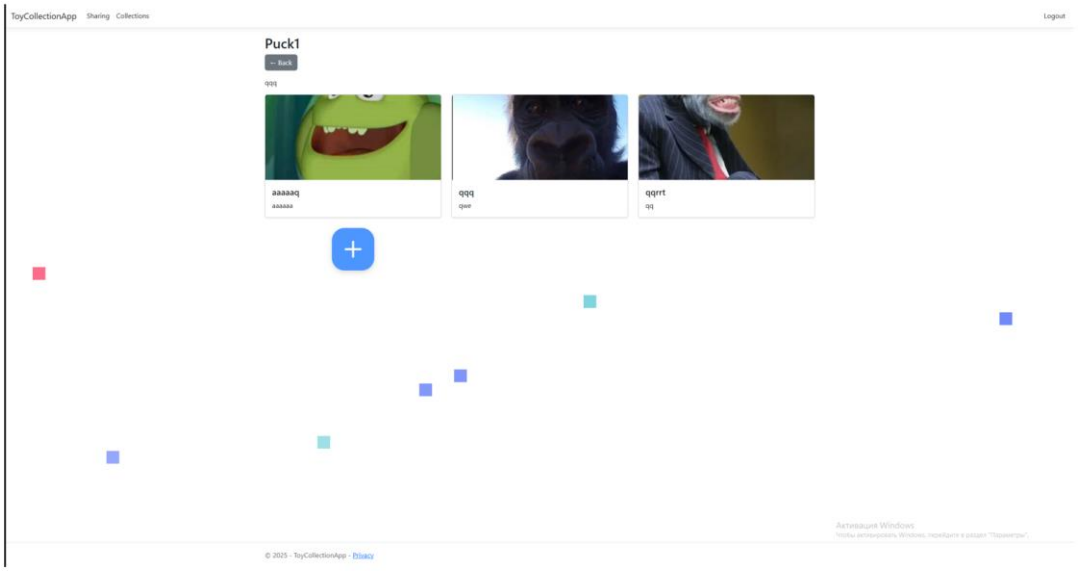
ЕКРАННІ ФОРМИ



Вікно модерації

9

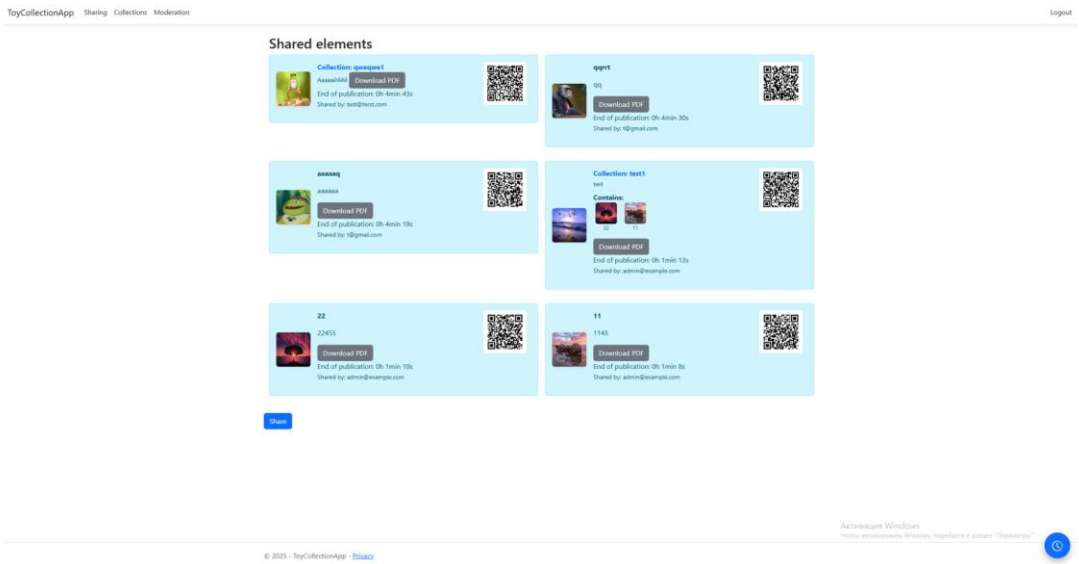
ЕКРАННІ ФОРМИ



Сторінка колекції

10

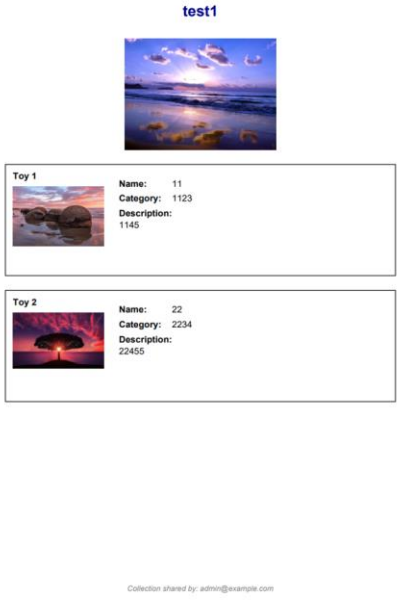
ЕКРАННІ ФОРМИ



Сторінка поділених публікацій

11

ЕКРАННІ ФОРМИ



Приклад генерованого PDF-файлу

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези:

1. Башич В.А., Садовенко В.С. Визначення вимог до web-застосунку для управління персональною колекцією ігрових предметів (на прикладі плюшових іграшок). Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ», 24.04.25, ДУІКТ, Київ, Україна, с.144-146.
2. Башич В.А., Садовенко В.С. Розробка програмних засобів захисту інформації у веб-застосунку в процесіуправління персональною колекцією. Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с.260-261.

13

ВИСНОВКИ

1. Проаналізував існуючі аналоги, визначив вимоги до програмного забезпечення.
2. Проаналізував засоби реалізації web-додатку для управління персональною колекцією плюшевих іграшок.
3. Спроектував архітектуру застосунку для управління персональною колекцією плюшевих іграшок.
4. Виконав програмне втілення та тестування застосунку для управління персональною колекцією плюшевих іграшок.

14

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ToyCollectionApp.Data;
using ToyCollectionApp.Models;

namespace ToyCollectionApp.Controllers
{
   public class ToyController : Controller
   {
      private readonly AppDbContext _context;

      public ToyController(AppDbContext
context)
      {
         _context = context;
      }

      public IActionResult Create(int
collectionId) // Метод для відображення форми
створення іграшки
      {
         ViewBag.CollectionId = collectionId; //
Передача ID колекції у view

         return View(); // Повернення view
форми створення
      }

      [HttpPost]
      [ValidateAntiForgeryToken] // Захист від
CSRF атак
      public async Task<IActionResult>
Create(Toy toy) // Метод для обробки
створення іграшки
      {
         if (toy.ToyImage == null)
         {

ModelState.AddModelError("ToyImage",
"Please select an image.");
         }

         var userId =
User.FindFirst(System.Security.Claims.ClaimTy
pes.NameIdentifier)?.Value; // Отримання ID
користувача

         //Перевірка: користувач має бути
власником колекції
         var collection = await
_context.ToyCollections.FirstOrDefaultAsync(c
=> c.ID == toy.ToyCollectionID); // Пошук
колекції
         if (collection == null || collection.UserId !=
userId)
            return Forbid();

         var isApproved = await
_context.SharedItems
      .AnyAsync(s => s.ToyCollectionId ==
collection.ID && s.IsApproved); // Перевірка,
чи колекція опублікована

         if (isApproved)
            return BadRequest("You cannot
modify a published collection. Please create a
new one.");
```

```
        if (toy.ToyImage != null)
        {

            using var ms = new MemoryStream();
            await
toy.ToyImage.CopyToAsync(ms);
            toy.ToyImageData =
Convert.ToBase64String(ms.ToArray());
        }

        toy.UserId = userId; // Прив'язка
користувача до іграшки
        _context.Toys.Add(toy); // Додавання
іграшки до БД
        await _context.SaveChangesAsync();

        return RedirectToAction("Details",
"ToyCollections", new { id =
toy.ToyCollectionID }); // Перенаправлення до
деталей колекції
    }


    public async Task<IActionResult>
Details(int id) // Метод для перегляду деталей
іграшки
    {
        var toy = await
_context.Toys.FindAsync(id); // Пошук іграшки
за ID
        if (toy == null)
            return NotFound();
        var userId =
User.FindFirst(System.Security.Claims.ClaimTy
pes.NameIdentifier)?.Value;

        // Получаем коллекцию, чтобы
проверить владельца
        var collection = await
_context.ToyCollections.FindAsync(toy.ToyColl
ectionID);
        if (collection == null)
            return NotFound();

        ViewBag.IsOwner = (collection.UserId ==
userId);
        return View(toy);
    }

    [HttpPost]
    [ValidateAntiForgeryToken] // Захист від
CSRF атак
    public async Task<IActionResult>
InlineEdit(Toy toy) // Метод для редагування
іграшки без перезавантаження сторінки
    {

ModelState.Remove(nameof(Toy.ToyImage)); //
Видалення валідації зображення (не
змінюється тут)
        if (!ModelState.IsValid)
        {
            return BadRequest(new { message =
"Validation failed." });
        }

        var existing = await
_context.Toys.FindAsync(toy.ID); // Пошук
існуючої іграшки
        if (existing == null)
            return NotFound();

        var userId =
User.FindFirst(System.Security.Claims.ClaimTy
pes.NameIdentifier)?.Value; // Отримання ID
користувача
```

```
        //Перевірка: тільки власник колекції може редагувати
        var collection = await _context.ToyCollections.FindAsync(existing.ToyCollectionID); // Пошук колекції
        if (collection == null || collection.UserId != userId)
            return Forbid();

        if (collection.IsApproved)
            return BadRequest("Cannot edit toys in a published collection.");

        existing.Name = toy.Name;
        existing.Description = toy.Description;
        existing.Category = toy.Category;

        await _context.SaveChangesAsync(); // Збереження змін у БД

        return Json(new // Повернення оновлених даних у JSON
        {
            name = existing.Name,
            category = existing.Category,
            description = existing.Description
        });
    }


    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Delete(int id) // Метод для видалення іграшки
    {
        var toy = await _context.Toys.FindAsync(id); // Пошук іграшки за ID
        if (toy == null)
            return NotFound();

        var userId = User.FindFirst(System.Security.Claims.ClaimTypes.NameIdentifier)?.Value; // Отримання ID користувача

        // Перевірка: тільки власник колекції може видаляти
        var collection = await _context.ToyCollections.FindAsync(toy.ToyCollectionID);  // Пошук колекції
        if (collection == null || collection.UserId != userId)
            return Forbid();

        if (collection.IsApproved)
            return BadRequest("Cannot delete toys from a published collection.");

        int collectionId = toy.ToyCollectionID; // Зберігання ID колекції для перенаправлення
        _context.Toys.Remove(toy); // Видалення іграшки у БД
        await _context.SaveChangesAsync(); // Збереження змін у БД

        return RedirectToAction("Details", "ToyCollections", new { id = collectionId }); // Перенаправлення на деталі колекції
    }
```

```
    }
}
using System.Security.Claims;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ToyCollectionApp.Data;
using ToyCollectionApp.Models;

namespace ToyCollectionApp.Controllers
{
    public class ToyCollectionsController :
Controller
    {
        private readonly AppDbContext _context;

        public
ToyCollectionsController(AppDbContext
context)
        {
            _context = context;
        }

        public IActionResult Create()// Повертає
сторінку створення нової колекції іграшок
        {
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create(ToyCollection collection)// Обробляє
створення нової колекції
        {
            if (ModelState.IsValid)
            {
                collection.UserId =
User.FindFirst(System.Security.Claims.ClaimTy
pes.NameIdentifier).Value;// Прив'язуємо ID
користувача до колекції

                if (collection.ToyImage != null)  //
Якщо завантажено зображення — зберігаємо
його як base64
                {
                    using var ms = new
MemoryStream();
                    await
collection.ToyImage.CopyToAsync(ms);
                    collection.ToyImageData =
Convert.ToBase64String(ms.ToArray());
                }

                // Встановлюємо, що це чернетка
                collection.IsDraft = true;


_context.ToyCollections.Add(collection);
                await _context.SaveChangesAsync();
                return
RedirectToAction(nameof(Index));
            }

            return View(collection);// Якщо дані
некоректні — повертаємо форму зі
збереженими значеннями
        }



        [HttpPost]
        public async Task<IActionResult>
JavaEdit() // Обробка AJAX-запиту
редагування (наприклад, через jQuery)
```

```
    {
        if (!int.TryParse(Request.Form["ID"],
out int id))
            return BadRequest("Invalid ID");

        var collection = await
_context.ToyCollections.FindAsync(id);
        if (collection == null)
            return NotFound();

        var userId =
User.FindFirst(ClaimTypes.NameIdentifier)?.V
alue;
        if (collection.UserId != userId)
            return Forbid(); // Заборонено
редагувати чужу колекцію

        collection.Name =
Request.Form["Name"];
        collection.Description =
Request.Form["Description"];
        collection.Category =
Request.Form["Category"];

        if (Request.Form.Files.Count > 0)
        {
            var file = Request.Form.Files[0];
            if (file.Length > 0)
            {
                using var ms = new
MemoryStream();
                await file.CopyToAsync(ms);
                collection.ToyImageData =
Convert.ToBase64String(ms.ToArray());
            }
        }

        try
        {
            _context.Update(collection);
            await _context.SaveChangesAsync();
            return Ok();
        }
        catch (Exception ex)
        {
            return StatusCode(500, "Ошибка при
сохранении данных: " + ex.Message);
        }
    }

    public async Task<IActionResult>
Details(int? id)
    {
        if (id == null)
            return NotFound();

        var userId =
User.FindFirst(ClaimTypes.NameIdentifier)?.V
alue;

        var collection = await
_context.ToyCollections // Отримуємо
колекцію разом з пов’язаними іграшками
            .Include(c => c.Toys)
            .FirstOrDefaultAsync(c => c.ID == id);

        if (collection == null)
            return NotFound();

        var isOwner = collection.UserId ==
userId; // Перевірка: або власник, або
колекція поділена з користувачемм

        var isShared = await
_context.SharedItems
```

```
            .AnyAsync(s => s.ToyCollectionId ==
id && s.IsApproved);

        if (!isOwner && !isShared)
            return Forbid();

        ViewBag.IsOwner = isOwner;

        return View(collection);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(int
id, ToyCollection toyCollection)
    {
        if (id != toyCollection.ID)
            return NotFound();

        var userId =
User.FindFirst(ClaimTypes.NameIdentifier)?.V
alue;
        var original = await
_context.ToyCollections.AsNoTracking().FirstO
rDefaultAsync(c => c.ID == id);
        if (original == null)
            return NotFound();

        if (original.UserId != userId)
            return Forbid();

        if (original.IsApproved)
            return BadRequest("Cannot edit a
published collection.");

        if (ModelState.IsValid)
        {
            try
            {
                toyCollection.UserId =
original.UserId;

                // Завжди зберігаємо як чорнетку
                toyCollection.IsDraft = true;
                toyCollection.IsApproved = false;

                _context.Update(toyCollection);
                await _context.SaveChangesAsync();
            }
            catch
(DbUpdateConcurrencyException)
            {
                if
(!ToyCollectionExists(toyCollection.ID))
                    return NotFound();
                else
                    throw;
            }
            return
RedirectToAction(nameof(Index));
        }
        return View(toyCollection);
    }

    public async Task<IActionResult>
Delete(int? id) // Підтвердження видалення
    {
        if (id == null)
            return NotFound();

        var collection = await
_context.ToyCollections.FirstOrDefaultAsync(m
=> m.ID == id);
        if (collection == null)
            return NotFound();
```

```
        var userId =
User.FindFirst(ClaimTypes.NameIdentifier)?.V
alue;
        if (collection.UserId != userId)
            return Forbid();

        return View(collection);
    }

    [HttpPost, ActionName("Delete")] //
Видаляє колекцію остаточно
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
DeleteConfirmed(int id)
    {
        var collection = await
_context.ToyCollections.FindAsync(id);
        if (collection == null)
            return NotFound();

        var userId =
User.FindFirst(ClaimTypes.NameIdentifier)?.V
alue;
        if (collection.UserId != userId)
            return Forbid(); // Заборонено
видаляти чужу колекцію


_context.ToyCollections.Remove(collection);
        await _context.SaveChangesAsync();
        return
RedirectToAction(nameof(Index));
    }

    private bool ToyCollectionExists(int id) //
Перевірка існування колекції за ID
    {
        return _context.ToyCollections.Any(e =>
e.ID == id);
    }

    public async Task<IActionResult> Index()
// Повертає всі колекції поточного
користувача
    {
        var userId =
User.FindFirst(System.Security.Claims.ClaimTy
pes.NameIdentifier).Value;
        var toys = await _context.ToyCollections
            .Where(tc => tc.UserId == userId)
            .ToListAsync();

        return View(toys);
    }



  }
}
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Authorization;
using ToyCollectionApp.Data;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;
using ToyCollectionApp.Models;
using Microsoft.AspNetCore.Identity;

namespace ToyCollectionApp.Controllers
{
  [Authorize]
  public class ShareController : Controller
  {
    private readonly AppDbContext _context;
    private readonly UserManager<Users>
_userManager;
```

```
    public ShareController(AppDbContext context, UserManager<Users> userManager)
    {
        _context = context;
        _userManager = userManager;
    }

    public async Task<IActionResult> Index()
    {
        const int expirationMinutes = 5;
        var expiration = DateTime.UtcNow.AddMinutes(-expirationMinutes); // Обчислюємо час до якого елементи вважаються застарілими (минулі 5 хвилин)

        // Отримуємо список спільних елементів, які затверджені і термін їх дії минув
        var expired = await _context.SharedItems
            .Where(s => s.IsApproved && s.SharedAt < expiration)
            .ToListAsync();

        if (expired.Any())
        {

_context.SharedItems.RemoveRange(expired); // Видаляємо застарілі елементи зі сховища
            await _context.SaveChangesAsync(); // Зберігаємо зміни в базі даних
        }

        var userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value; // Отримуємо ідентифікатор поточного користувача

        // Отримуємо всі затверджені спільні елементи, сортуємо за датою спільного доступу за спаданням
        var approvedItems = await _context.SharedItems
            .Include(s => s.User)
            .Include(s => s.Toy)
            .Include(s => s.ToyCollection)
            .Where(s => s.IsApproved)
            .OrderByDescending(s => s.SharedAt)
            .ToListAsync();

        // Отримуємо всі спільні елементи поточного користувача, які були відхилені (мають причину відмови)
        var rejectedItems = await _context.SharedItems
            .Include(s => s.User)
            .Include(s => s.Toy)
            .Include(s => s.ToyCollection)
            .Where(s => s.UserId == userId && s.RejectionReason != null)
            .OrderByDescending(s => s.SharedAt)
            .ToListAsync();

        var viewModel = new SharePageViewModel
        {
            ApprovedSharedItems = approvedItems, // Затверджені спільні елементи
            RejectedSharedItems = rejectedItems // Відхилені спільні елементи
        };

        return View(viewModel); // Повертаємо модель для відображення
```

```
    }

    [HttpGet]
    public async Task<IActionResult>
GetPendingUserItems()
    {
        var userId =
User.FindFirst(ClaimTypes.NameIdentifier)?.V
alue; // Ідентифікатор поточного користувача

        // Отримуємо всі елементи
користувача, що очікують на затвердження і
не мають причини відмови
        var pendingItems = await
_context.SharedItems
            .Include(s => s.Toy)
            .Include(s => s.ToyCollection)
            .Where(s => s.UserId == userId &&
!s.IsApproved && s.RejectionReason == null)
            .ToListAsync();

        // Формуємо результат із типом
(іграшка чи колекція) та назвою
        var result = pendingItems.Select(s => new
{
            type = s.Toy != null ? "Toy" :
"Collection",
            name = s.Toy?.Name ??
s.ToyCollection?.Name
        });

        return Json(result); // Повертаємо
результат у форматі JSON
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
ShareItem(string type, int id)
    {
        var userId =
User.FindFirst(ClaimTypes.NameIdentifier)?.V
alue; // Ідентифікатор користувача

        var shared = new SharedItem
        {
            UserId = userId, // Встановлюємо
користувача, що ділиться елементом
            SharedAt = DateTime.UtcNow //
Встановлюємо час спільного доступу (UTC)
        };

        // Перевіряємо, чи не було вже раніше
надіслано запиту на спільний доступ цього
елемента
        var alreadyShared = await
_context.SharedItems.AnyAsync(s =>
            s.UserId == userId &&
           ((type == "toy" && s.ToyId == id) ||
(type == "collection" && s.ToyCollectionId ==
id)) &&
        s.RejectionReason == null &&
!s.IsApproved
        );
        if (alreadyShared)
        {
            return BadRequest("You have already
requested to share this item."); // Повертаємо
помилку, якщо запит вже існує
        }

        if (type == "toy")
        {
```

```
            var toy = await
_context.Toys.FirstOrDefaultAsync(t => t.ID ==
id && t.UserId == userId);
            shared.ToyId = toy.ID; // Прив'язуємо
іграшку до запиту на спільний доступ
        }
        else if (type == "collection")
        {
            var collection = await
_context.ToyCollections.FirstOrDefaultAsync(c
=> c.ID == id && c.UserId == userId);
            shared.ToyCollectionId = collection.ID;
// Прив'язуємо колекцію до запиту на
спільний доступ
        }
        else
        {
            return BadRequest("Unknown type");
// Повертаємо помилку, якщо тип невідомий
        }

        _context.SharedItems.Add(shared); //
Додаємо новий запит на спільний доступ до
контексту
        await _context.SaveChangesAsync(); //
Зберігаємо зміни у базі даних

        return Ok(); // Повертаємо успішний
результат
    }

    [AllowAnonymous] // Дозволяємо доступ
без автентифікації
    public async Task<IActionResult>
PublicShares()
    {
        // Отримуємо всі затверджені спільні
елементи, сортуємо за датою спільного
доступу за спаданням
        var sharedNow = await
_context.SharedItems
            .Where(s => s.IsApproved)
            .Include(s => s.Toy)
            .Include(s => s.ToyCollection)
            .OrderByDescending(s => s.SharedAt)
            .ToListAsync();

        return View(sharedNow); // Повертаємо
список затверджених елементів для
публічного перегляду
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
ClearRejected()
    {
        var userId =
User.FindFirst(ClaimTypes.NameIdentifier)?.V
alue;

        // Знаходимо всі відхилені елементи
користувача
        var rejectedItems = await
_context.SharedItems
            .Where(s => s.UserId == userId &&
s.RejectionReason != null)
            .ToListAsync();

        if (rejectedItems.Any())
        {

_context.SharedItems.RemoveRange(rejectedIte
ms);
```

```
            await _context.SaveChangesAsync();
        }

        // Після видалення редірект на сторінку, або можна повертати JSON для AJAX
        return RedirectToAction(nameof(Index));
    }

    [HttpGet]
    public async Task<IActionResult> GetItems(string type)
    {
        var userId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value; // Ідентифікатор користувача

        if (type == "toy")
        {
            // Отримуємо всі іграшки поточного користувача (id та назва)
            var toys = await _context.Toys
                .Where(t => t.UserId == userId)
                .Select(t => new { id = t.ID, name = t.Name })
                .ToListAsync();
            return Json(toys); // Повертаємо список іграшок у форматі JSON
        }

        if (type == "collection")
        {
            // Отримуємо всі колекції іграшок поточного користувача (id та назва)
            var collections = await _context.ToyCollections
                .Where(c => c.UserId == userId)
                .Select(c => new { id = c.ID, name = c.Name })
                .ToListAsync();
            return Json(collections); // Повертаємо список колекцій у форматі JSON
        }

        return BadRequest("Invalid type"); // Повертаємо помилку, якщо тип некоректний
    }

    [Authorize(Roles = "Admin")] // Доступ дозволений лише для користувачів з роллю Адміністратора
    public async Task<IActionResult> Moderation()
    {
        var query = _context.SharedItems
            .Include(si => si.Toy)
            .Include(si => si.ToyCollection)
            .Include(si => si.User)
            .Where(si => !si.IsApproved)
            .Where(si => !si.IsApproved && si.RejectionReason == null);
        var items = await query.ToListAsync();

        return View(items);
    }

    [HttpPost]
    [Authorize(Roles = "Admin")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Approve(int id)
    {
        var item = await _context.SharedItems.FindAsync(id); // Знаходимо елемент за ідентифікатором
```

```
        if (item != null)
        {
            item.IsApproved = true; // Позначаємо елемент як затверджений
            await _context.SaveChangesAsync(); // Зберігаємо зміни у базі даних
        }

        return RedirectToAction(nameof(Moderation)); // Переадресовуємо назад на сторінку модерації
    }

    [HttpGet]
    [Authorize(Roles = "Admin")]
    public async Task<IActionResult> Reject(int id)
    {
        // Знаходимо елемент разом з іграшкою та колекцією за ідентифікатором
        var item = await _context.SharedItems
            .Include(s => s.Toy)
            .Include(s => s.ToyCollection)
            .FirstOrDefaultAsync(s => s.ID == id);

        if (item == null) return NotFound(); // Якщо елемент не знайдено, повертаємо 404

        return View(item); // Показуємо сторінку для відхилення із деталями елемента
    }

    [HttpPost]
    [Authorize(Roles = "Admin")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Reject(int id, string reason)
    {
        var item = await _context.SharedItems.FindAsync(id); // Знаходимо елемент за ідентифікатором

        if (item != null)
        {
            item.RejectionReason = reason; // Записуємо причину відмови
            item.IsApproved = false; // Позначаємо елемент як відхилений
            await _context.SaveChangesAsync(); // Зберігаємо зміни
        }

        return RedirectToAction(nameof(Moderation)); // Переадресовуємо назад на сторінку модерації
    }
```