МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ЛЕСІ УКРАЇНКИ

Кафедра комп'ютерних наук та кібербезпеки

ФЕДОРЕНКО КОСТЯНТИН ОЛЕКСАНДРОВИЧ

РОЗРОБКА ТА ВПРОВАДЖЕННЯ ОНЛАЙН-ПЛАТФОРМИ ДЛЯ КРАУДФАНДИНГУ КНИЖКОВИХ ПРОЄКТІВ

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки та інформаційні технології
Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:
БУЛАТЕЦЬКА ЛЕСЯ ВІТАЛІЇВНА,
кандидат фізико-математичних наук, доцент кафедри
комп'ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол №_____
засідання кафедри комп'ютерних наук
та кібербезпеки
від _____ 2025 р.
Завідувач кафедри
(_____) Гришанович Т. О.

ЛУЦЬК - 2025

ЗМІСТ

ВСТУП

Актуальність. Сучасний книжковий ринок переживає значну трансформацію під впливом цифрових технологій. Одним із ключових трендів є зростання популярності самвидавництва: дедалі більше авторів прагнуть незалежності від традиційних видавництв, які, як правило, надають перевагу комерційно вигідним проєктам. Натомість креативні, нішеві або експериментальні твори часто залишаються без належної уваги та підтримки.

У таких умовах краудфандинг [1] виявляється ефективним інструментом реалізації творчих ініціатив. Він дозволяє авторам напряму взаємодіяти з потенційними читачами й отримувати фінансування на ранніх етапах створення книжки. Проте більшість існуючих краудфандингових платформ орієнтовані на широкий спектр тем (технології, стартапи, благодійність) і не враховують особливості книжкових проєктів – від специфіки роботи з аудиторією до потреб у редакційній, маркетинговій чи логістичній підтримці.

Крім того, ринок друкованих і електронних книг, незважаючи на конкуренцію з боку цифрових медіа, стабільно зростає. Це свідчить про сталий інтерес читачів до якісного літературного контенту, особливо створеного за участі спільноти. Попит на платформи, що забезпечують прозорий механізм фінансування, зручну взаємодію авторів і аудиторії, а також прості інструменти для управління книжковими проєктами, постійно зростає.

З точки зору інформаційних технологій, постає завдання розробити масштабований, зручний та ефективний вебзастосунок з мінімальними ресурсними витратами. Така платформа повинна мати інтеграцію платіжних систем і сторонніх API (аналітика, підтримка, нотифікації), гнучку архітектуру з можливістю масштабування, інтерфейс, орієнтований на користувача (UI/UX) та інструменти для управління кампаніями, читачами та прогресом видання.

Отже, створення онлайн-платформи для краудфандингу саме книжкових проєктів є актуальним як з культурної, так і з технологічної точки зору. Такий продукт сприятиме подоланню фінансових бар’єрів для авторів-початківців,

розширенню можливостей для самовираження та творчого зростання, розвитку читацьких спільнот та залученню меценатів до культурного процесу і демократизації книговидання і зростанню прозорості в цій сфері.

Таким чином, реалізація спеціалізованої онлайн-платформи відповідає актуальним потребам ринку і суспільства, об’єднуючи інтереси авторів, читачів та інвесторів у єдиному функціональному середовищі.

Метою роботи є створення функціональної та зручної онлайн-платформи для краудфандингу книжкових проєктів, яка одночасно відповідатиме потребам як авторів, так і читачів. Такий сервіс має стати інструментом, що дозволяє авторам презентувати свої творчі ідеї, знаходити фінансування та отримувати підтримку на всіх етапах реалізації проєкту – від задуму до видання. Водночас платформа має стати місцем для активної участі читачів, які зможуть підтримувати обрані ініціативи, впливати на їхній розвиток і відчувати свою причетність до народження нових книг.

Особлива увага приділяється створенню умов для прозорої взаємодії між усіма учасниками процесу. Це передбачає наявність інструментів для планування, координації та моніторингу проєктів, що сприятиме ефективнішій організації праці авторів. Окрім цього, важливо забезпечити механізми, які дозволять користувачам відслідковувати використання зібраних коштів, тим самим формуючи атмосферу довіри, відповідальності та відкритості. Платформа має не лише задовольняти функціональні вимоги, а й стимулювати розвиток літературного середовища, підтримувати молоді таланти й заохочувати до створення інноваційних книжкових проєктів.

Для досягнення поставленої мети необхідно створити технічну основу, яка дозволить у стислі терміни та з мінімальними витратами ресурсів реалізувати Мінімально життєздатний продукт (MVP). Насамперед передбачається вибір оптимальних технологій і інструментів, що відповідають вимогам функціональності, стабільності й економічної ефективності. Важливим **завданням** є розробка архітектури, здатної підтримувати подальшу

масштабованість платформи, не створюючи надмірного навантаження на команду розробників.

Крім того, проєкт потребує створення сучасного, інтуїтивно зрозумілого інтерфейсу, орієнтованого на кінцевого користувача, що забезпечить зручність взаємодії для авторів, читачів і меценатів. Особливу увагу слід приділити інтеграції зовнішніх сервісів, зокрема платіжних систем, які забезпечать прозоре й ефективне функціонування фінансової частини платформи. У цьому контексті важливо закласти передумови для подальшого використання рішень, подібних до сервісу Fondy[18], який дозволяє організовувати прийом платежів у кілька кліків і має готові інтеграційні рішення для українського ринку.

У сукупності ці завдання спрямовані на створення стабільної, доступної та орієнтованої на користувача системи, яка може бути розширена або вдосконалена в міру зростання спільноти та функціональних потреб.

Об’єкт дослідження – процес створення та функціонування інформаційних систем у сфері краудфандингу книжкових проєктів.

Предмет дослідження – архітектурні рішення, технології веброзробки та інтеграційні механізми, що використовуються при розробці онлайн-платформи для краудфандингу книжкових проєктів.

Апробація результатів та публікації.

Костянтин Федоренко. Онлайн книжкова краудфандингова платформа. Проблеми комп’ютерних наук, програмного моделювання та безпеки цифрових систем.: матеріали II міжнар. науково-практ. конф. Луцьк, 9–11 червн. 2025 р. Луцьк, 2025.

РОЗДІЛ 1.
ПРИНЦИПИ МАСШТАБУВАННЯ СУЧАСНИХ ВЕБЗАСТОСУНКІВ

1.1 Принципи масштабованої архітектури вебзастосунків

Масштабований вебзастосунок – це програмне забезпечення, що працює в вебсередовищі та здатне ефективно функціонувати за умов зростання навантаження (користувачів, запитів, даних) завдяки розширенню інфраструктури або оптимізації архітектури. Масштабованість – одна з ключових характеристик сучасних вебзастосунків, яка забезпечує їхню продуктивність та стабільність навіть за різких змін у кількості користувачів.

Щоб досягнути високого рівня масштабованості, архітектура вебзастосунку має передбачати розділення обчислювальних навантажень, гнучке розгортання нових компонентів і можливість адаптації до зростання системи. Основою такої архітектури часто є розподілені системи, мікросервісний підхід, кешування, контейнеризація, використання хмарної інфраструктури, реплікація баз даних і балансування навантаження. Всі ці елементи не лише покращують технічні характеристики системи, а й дозволяють масштабувати окремі її частини незалежно одна від одної.

Під час розробки масштабованої архітектури критично важливим є також правильне проєктування схеми даних і механізмів їхньої міграції. Коли система зростає, структура бази даних неминуче змінюється: додаються нові таблиці, поля, зв’язки. Щоб ці зміни були контрольованими й не викликали збоїв, розробники використовують систему міграцій. Міграції – це керовані сценарії зміни структури бази даних, які дозволяють точно відтворити зміни як на етапі розробки, так і під час оновлення production-версії. Важливо, щоб ці міграції були ідпотентними та могли виконуватися автоматизовано, наприклад, під час CI/CD-процесів.

Ще одним важливим аспектом масштабованої архітектури є розгортання production-версії застосунку. На цьому етапі стає особливо помітною роль

контейнеризації – кожен компонент (сервер, база даних, проксі, сервіс обробки зображень чи платежів) ізольовано в окремому контейнері, що гарантує стабільність, контроль версій та простоту масштабування. За допомогою систем на кшталт Docker Compose[26] або Kubernetes можна визначити залежності між сервісами, контролювати кількість реплік кожного з них та автоматизувати оновлення. Оскільки в production-середовищі важлива безперервна доступність, оновлення застосунку зазвичай відбувається з використанням blue-green deployment або rolling-update стратегій, що мінімізують або повністю усувають час простою.

1.1.1. Передбачення масштабування під майбутні бізнес-потреби

Ефективне масштабування вебзастосунку неможливе без врахування майбутніх бізнес-потреб ще на етапі планування архітектури. Часто системи, що створюються лише для задоволення поточних вимог, виявляються обмеженими, коли компанія виходить на нові ринки або починає залучати більшу кількість користувачів. Тому архітектурні рішення мають враховувати потенційний ріст навантаження, розширення функціональності, підтримку багатомовності, інтеграції з додатковими сервісами та зміну бізнес-моделі.

Підготовка до масштабування – це не стільки про «програмувати наперед», скільки про створення гнучкої основи, яка не потребуватиме повного переписування системи при зростанні. Наприклад, проєктування бази даних із можливістю легкого введення нових сутностей або залежностей, використання сервіс-орієнтованих підходів замість монолітної логіки, визначення чітких контрактів API для забезпечення сумісності в майбутньому.

Крім того, слід враховувати і зміну характеру навантаження. Наприклад, у початковій фазі платформа може обслуговувати сотню користувачів на день, але бізнес-стратегія передбачає зростання до десятків тисяч за кілька місяців. У такому разі необхідно одразу закладати можливість горизонтального масштабування, кешування даних, розділення задач на фонові процеси та впровадження черг повідомлень.

Передбачення масштабування означає також обґрунтований вибір зовнішніх сервісів, які можуть ефективно працювати з великим навантаженням. Наприклад, якщо планується підключення платіжних сервісів, доцільно одразу обирати таких провайдерів, як Fondy, які мають стабільні API, документацію, інструменти звітності та підтримку великого обсягу транзакцій. Вибір таких технологій з урахуванням майбутніх вимог дозволяє уникнути дорогих міграцій у майбутньому.

1.1.2. Технологічна база для підтримки масштабування

Сучасна технологічна база є основою здатності застосунку витримувати високі навантаження та адаптуватися до зростання. Йдеться не лише про вибір мов програмування або фреймворків, а про комплексне середовище розробки, деплойменту та підтримки системи. Кожен рівень – від клієнтського інтерфейсу до зберігання даних і моніторингу – має бути готовим до масштабування.

На рівні бекенду важливо використовувати фреймворки, які підтримують модульність, чітку організацію коду та інтеграцію з різними базами даних і зовнішніми сервісами. Наприклад, Ruby on Rails[5] забезпечує потужну систему міграцій, Active Job для фонових задач, Action Cable для роботи в реальному часі та Active Storage для зберігання файлів, що дозволяє масштабувати не лише логіку, а й побічні функціональності. У поєднанні з Redis[6], Sidekiq[14] або PostgreSQL[16] із підтримкою реплікації, система стає здатною обробляти тисячі одночасних запитів із мінімальними затримками.

З боку фронтенду застосування React або Vue дозволяє реалізовувати SPA-архітектуру, яка зменшує навантаження на сервери завдяки клієнтському рендерингу. Адаптивні інтерфейси забезпечують однаково зручну роботу з різних пристроїв, що особливо важливо для платформи з широким охопленням аудиторії.

Інфраструктурний рівень має бути побудований з використанням контейнеризації та CI/CD-інструментів. Docker забезпечує ізоляцію середовища для кожного сервісу, що дозволяє легко запускати нові екземпляри без повторної

конфігурації. Kubernetes – платформа оркестрації – автоматизує балансування навантаження, масштабування, самовідновлення та оновлення компонентів. Це знижує навантаження на команду підтримки та підвищує стабільність системи.

Хмарні сервіси, такі як AWS, Google Cloud або Azure, надають готові рішення для баз даних, зберігання файлів, безсерверних обчислень та глобального кешування. Їх використання значно пришвидшує розгортання системи у production-середовищі й зменшує час простою при масштабуванні.

Підключення зовнішніх сервісів, зокрема платіжних систем (наприклад, Fondy), вимагає врахування таких аспектів, як обробка помилок, повторна доставка подій (webhooks), відповідність стандартам безпеки та ліміти транзакцій. Тому технологічна база має включати механізми черг, журналювання, аудитів і резервного збереження, що дозволяє не втратити критичну інформацію у випадку збоїв.

Підсумовуючи, можна сказати, що технологічна база масштабованого вебзастосунку – це не набір інструментів, а скоординована система, здатна підтримувати гнучкість, продуктивність і надійність при зростанні навантаження. Її ефективне формування закладає основу для довготривалого успіху продукту.

1.2. Використання API

У масштабованих вебзастосунках API є невід’ємним інструментом, що забезпечує взаємодію між різними компонентами системи, інтеграцію з зовнішніми сервісами та можливість швидко розширювати функціональність без потреби змінювати основну архітектуру. API (Application Programming Interface) дозволяє будувати гнучкі рішення, де кожна частина – незалежна, проте взаємопов’язана. В умовах зростання навантаження або змін бізнес-процесів це дає змогу адаптуватися швидко та безболісно.

Особливе місце в архітектурі займають API зовнішніх постачальників послуг. Вони дозволяють делегувати критично важливі, але спеціалізовані

функції (такі як оплата, автентифікація, відеострімінг) стороннім рішенням, які вже масштабовані, протестовані та відповідають галузевим стандартам.

Одним із найяскравіших прикладів такого підходу є інтеграція платіжної системи Fondy. Цей сервіс надає добре задокументований REST API, що дозволяє зручно обробляти платежі з банківських карток, Apple Pay, Google Pay та інших джерел. Для масштабованого застосунку важливо, що Fondy підтримує webhook-повідомлення – це події, які Fondy надсилає до вашого серверу при зміні статусу транзакції. Це дозволяє реалізувати надійний облік платежів, навіть у випадку тимчасових збоїв на фронтенді або втрати інтернет-з'єднання у користувача. Крім того, API Fondy дозволяє створювати кастомізовані платіжні форми, реалізовувати регулярні платежі, переглядати статус транзакцій і формувати фінансову звітність – усе це необхідно для стабільного функціонування та масштабування комерційних вебплатформ.

Ще одним прикладом зовнішнього API, що часто використовується в масштабованих системах, є вбудоване API YouTube[21]. Через нього можна інтегрувати відеоконтент прямо в інтерфейс вебзастосунку. Повна взаємодія з контентом можлива завдяки YouTube Player API, яке дає змогу програмно керувати відеопрогравачем. Це включає відтворення, паузу, відстеження часу перегляду, обробку подій та побудову аналітики взаємодії користувача з контентом. Такий підхід особливо важливий у навчальних платформах або медіапроєктах, де взаємодія з відео має аналітичне або поведінкове значення.

Важливо також враховувати, що робота з API потребує дотримання певних принципів безпеки – зокрема, правильного зберігання ключів доступу, захисту від CSRF-атак, перевірки достовірності відповідей (наприклад, за допомогою підпису повідомлень, як у Fondy), а також обмеження частоти запитів (rate limiting), щоб уникнути перевантаження або блокування сервісу.

Таким чином, використання API не лише спрощує розробку, а й дозволяє зосередитися на унікальній логіці власного продукту, покладаючи рутинні або складні задачі на спеціалізовані сервіси. Це і є одна з ключових умов побудови масштабованої, гнучкої та конкурентоспроможної архітектури вебзастосунку.

1.2.1 Принципи використання API

Правильне використання API у вебзастосунках передбачає дотримання низки архітектурних, безпекових та функціональних принципів, які відрізняються залежно від того, на якій стороні – клієнтській чи серверній – здійснюється інтеграція. На фронтенді API зазвичай використовуються безпосередньо у браузері, тому мають відкриті ключі доступу або не потребують автентифікації зовсім. Наприклад, при вбудовуванні відео через YouTube достатньо iframe-інтеграції з передачею ідентифікатора відео. Такий підхід не дає змоги керувати контентом з боку клієнта, однак дозволяє швидко й безпечно показати сторонній ресурс без потреби доступу до приватної інформації чи серверної логіки.

У випадках, коли потрібно реалізувати більш складну взаємодію, фронтенд може звертатися до API з використанням відкритих ключів – наприклад, для підключення інтерактивних карт. Такі сервіси, як Google Maps або Leaflet із плагінами, дозволяють виводити карту з маркерами, маршрутизацією чи фільтрацією за геолокацією. У більшості випадків API-ключ розміщується в JavaScript-коді і передається разом із запитом, але має обмеження за походженням запиту (через перевірку домену або IP), що дещо захищає ключ від несанкціонованого використання. Водночас потрібно пам’ятати, що повну безпеку на клієнтському рівні гарантувати неможливо, тому всі критичні дані та логіка завжди мають оброблятись на сервері.

На бекенді принципи роботи з API значно суворіші. Тут часто використовуються приватні ключі, токени доступу або сертифікати, які повинні зберігатись поза межами репозиторію – зазвичай у змінних середовища або в захищених секрет-сховищах. У середовищі розробки такі ключі зберігаються у спеціальних файлах, які виключаються з системи контролю версій, і підставляються у конфігурацію під час запуску застосунку. У production-середовищі значення передаються через механізми CI/CD або систему розгортання, з обов’язковим обмеженням доступу до них для сторонніх користувачів і розробників, які не мають відповідного рівня прав.

Крім збереження ключів, важливо також чітко розмежовувати тестове та production середовище API. Наприклад, платіжні сервіси, такі як Fondy, надають два набори ключів – для тестового режиму та для повноцінного використання. У тестовому режимі всі операції симулюються, тому можна без ризику перевіряти логіку відображення статусів транзакцій, обробку помилок, сценарії з невдалими платежами або навмисно затриманими відповідями. Перехід у production передбачає увімкнення реального обліку коштів, тому будь-яке неправильне поводження з API або відкриті ключі можуть призвести до фінансових втрат або витоку персональних даних. Саме тому під час розгортання важливо автоматизувати перевірку правильності середовища, а також мати журнал подій та логування викликів API – зокрема, для підтвердження вебгачків, які є основою зворотного зв’язку у таких сервісах.

Загальним принципом, якого слід дотримуватись при роботі з API, є чітке розділення ролей між фронтендом і бекендом: інтерфейс взаємодіє лише з безпечними відкритими даними або посередництвом власного серверного API, а весь контроль і перевірки мають залишатися на боці сервера. Це дозволяє не тільки убезпечити систему, а й забезпечити стабільність, логування та зручне масштабування. Крім того, в умовах масштабованого застосунку, де одночасно працює багато користувачів, критично важливо забезпечити обмеження кількості запитів (rate limiting) та кешування відповідей API, щоб уникнути перевантаження зовнішніх сервісів і власного бекенду.

1.2.2. Огляд API банківських сервісів

API банківських сервісів є одним із найважливіших компонентів фінансової інтеграції у вебзастосунках. Завдяки ним можна реалізувати прийом і обробку платежів, перевірку балансу, автоматичні виплати, перевірку рахунків, а також моніторинг транзакцій у реальному часі. Різні банки й платіжні системи пропонують різні підходи до API – як за структурою, так і за ступенем відкритості та вимогами до безпеки.

Крім неодноразово згаданого раніше Fondy поширеним прикладом є API банків за стандартом Open Banking[17], який дозволяє отримувати доступ до рахунків користувачів, ініціювати перекази та переглядати історію транзакцій. Такі API доступні, наприклад, у Монобанк, Приватбанк, Райфайзен банк або Revolut. Вони зазвичай вимагають складної процедури автентифікації – використання OAuth 2.0, багатоетапного погодження доступів і дій з боку користувача. У більшості випадків застосунку заборонено прямо зберігати облікові дані користувача – натомість система працює з токенами доступу, які мають обмежений строк дії і повинні бути регулярно оновлювані.

Важливим розмежуванням між різними API є підхід до зберігання даних. У платіжних API, як-от LiqPay чи WayForPay, система зазвичай дозволяє зберігати лише технічні дані про транзакцію: суми, статуси, унікальні ідентифікатори. Жодні банківські реквізити користувачів (номери карток, CVV, термін дії) не можуть зберігатися на стороні вебзастосунку – це суворо заборонено як політиками самих сервісів, так і міжнародними стандартами, зокрема PCI DSS. Дані, які все ж таки можна зберігати локально, мають проходити через системи логування, які не містять конфіденційної інформації, та архівуватися з дотриманням політик захисту даних.

Крім того, деякі API надають доступ до аналітики й дозволяють інтегрувати перевірку фінансової репутації – наприклад, API від платіжного шлюзу Portmone дозволяє зберігати результати скорингу або історію замовлень у системі CRM. Такі дані часто зберігаються у власній базі даних застосунку, але потребують анонімізації, якщо використовуються для аналітики або тестування.

Зовнішнє зберігання даних також передбачене у випадках, коли застосовується посередник для обробки платежів. У таких випадках бази транзакцій розташовані на серверах самого банку або платіжної системи, а вебзастосунок лише зчитує результат. Це зменшує відповідальність розробника за збереження критичних даних, але водночас потребує постійного зв’язку з API для оновлення статусів, що впливає на архітектуру і стійкість системи.

1.3. Збереження дружнього до користувача інтерфейсу у масштабованих вебзастосунках

Інтерфейс користувача є обличчям будь-якого вебзастосунку. Його зручність, зрозумілість та адаптивність визначають не лише перше враження, а й загальний досвід взаємодії з системою. Масштабування вебзастосунку не повинно порушувати логіки або ергономіки інтерфейсу – навпаки, зі збільшенням кількості користувачів, ролей та сценаріїв взаємодії, інтерфейс має ставати ще гнучкішим і чіткіше реагувати на контекст використання.

У масштабованих вебзастосунках, де одна і та сама сторінка може набувати різного вигляду залежно від ролі користувача або контексту її використання, збереження дружнього до користувача інтерфейсу є особливо важливим завданням. Для цього застосовуються низка технічних рішень, які забезпечують адаптивність і послідовність взаємодії. Одним із ключових підходів є компонентна архітектура, що реалізується за допомогою фреймворків на кшталт React або Vue.js. Компоненти дозволяють створювати багаторазові елементи інтерфейсу з параметричним керуванням, що забезпечує їхнє динамічне відображення залежно від ролі користувача чи стану застосунку. Розробка й підтримка єдиної дизайн-системи, яка включає стилістично уніфіковані елементи інтерфейсу, також сприяє підтриманню візуальної послідовності під час масштабування.

Важливу роль відіграє контроль доступу на рівні інтерфейсу, який дозволяє відображати або приховувати певні елементи згідно з правами користувача. Цей підхід супроводжується умовним рендерингом, що дозволяє враховувати не лише ролі, а й поточний стан об’єкта, контекст взаємодії або етап бізнес-процесу. Для підтримки адаптивності в середовищах з різними розмірами екранів застосовується mobile-first підхід, який забезпечує логічну і зручну структуру інтерфейсу як на десктопах, так і на мобільних пристроях. Рольові конфігурації дозволяють задавати окремі правила і логіку виводу інтерфейсних

компонентів для кожної категорії користувачів, що значно спрощує підтримку й розвиток системи.

Крім цього, використовуються механізми динамічного підключення модулів, зокрема lazy loading, що дозволяє підвантажувати лише ті компоненти, які необхідні в конкретному контексті, зменшуючи тим самим навантаження на клієнт і пришвидшуючи час завантаження сторінки. Для кращого розуміння взаємодії користувачів з інтерфейсом впроваджуються інструктивні повідомлення та підказки, що адаптуються до ролі та дій користувача. Це допомагає уникнути дезорієнтації в інтерфейсі, особливо за умов частих змін або появи нових функцій. З метою перевірки стабільності інтерфейсу в умовах масштабування застосовуються юніт-тести, які дають змогу виявити і виправити помилки, пов’язані з логікою відображення контенту. Додатково інтегруються засоби аналітики поведінки користувача, що дозволяє на основі реальних даних коригувати структуру інтерфейсу й оптимізувати його під різні сценарії використання. Усі ці рішення разом формують гнучку, адаптивну і масштабовану модель побудови користувацького інтерфейсу, що відповідає вимогам якості й зручності сучасних вебзастосунків.

1.4. Огляд та аналіз існуючих онлайн-платформ для краудфандингу книжкових проєктів

У межах дослідження ринку було розглянуто низку проєктів[3], які частково або опосередковано реалізують функціональність, подібну до книжкової краудфандингової платформи. З огляду на український контекст, важливо аналізувати не лише світові приклади, але й локальні ініціативи, які мають культурну або структурну близькість до запропонованої моделі.

Одним із найпомітніших проєктів в українському середовищі є Booknet[7] (раніше Litnet) – платформа, що дозволяє авторам публікувати книги частинами, як безкоштовно, так і за передплатою. Вона має активну читацьку спільноту, систему оцінювання й коментування, а також механізми попереднього доступу

до контенту. Однак ця платформа орієнтована передусім на цифровий формат і не пропонує централізованої системи краудфандингу саме для друкованих книжок. Водночас вона демонструє ефективність побудови спільноти навколо автора і твору, що є релевантним аспектом.

Британський Unbound [15] є прикладом спеціалізованої книжкової краудфандингової платформи, де видання книжки відбувається лише після збору необхідної суми. Це ближче до концепції, що розробляється в даній роботі, проте сервіс не є доступним в українському правовому чи платіжному полі і фокусується на англомовних авторів. Його перевага – підтримка маркетингу та редактури, але водночас він має високу конкуренцію і жорсткі вимоги до прийому проєктів.

Щодо українських рішень, можна згадати загальні краудфандингові платформи, як-от Спільнокошт[2] (спільна ініціатива "Великої Ідеї"), де час від часу з’являються проєкти з видання книжок. Проте сам механізм платформи не є спеціалізованим під потреби авторів: немає персоналізованої бібліотеки, постійної взаємодії з читачами або окремого авторського кабінету для керування процесом роботи над книгою. Проєкти тут часто мають разовий характер, без можливості довготривалої підтримки з боку аудиторії.

Певну функціональну спорідненість має й український сервіс Yakaboo Publishing[12], який з одного боку є видавництвом, а з іншого – експериментує з прямим продажем книжок на стадії передзамовлення. Водночас, це не є краудфандинг у повному сенсі: покупці отримують готову продукцію, а не фінансують сам процес її створення.

Також варто згадати практику авторських передзамовлень через соцмережі, яку активно застосовують сучасні українські автори та ілюстратори. Вони збирають кошти через Patreon [4], PayPal або банківські перекази, рекламуючи майбутню книгу в Instagram, Facebook чи Telegram. Цей спосіб лишається фрагментованим, неформалізованим і не гарантує прозорості процесу, однак демонструє потребу в централізованому рішенні.

Таким чином, серед існуючих рішень не виявлено повноцінної української платформи, що поєднувала б краудфандинг саме для книжкових проєктів із внутрішньою логікою підтримки авторів на етапі створення, фінансування, друку й комунікації з читачами. Запропонована в цій роботі платформа може заповнити цю нішу, адаптувавши успішні елементи від згаданих проєктів – зокрема прозорий збір коштів, передзамовлення, прямий контакт автора з аудиторією, можливість публікації фрагментів та отримання зворотного зв’язку, а також інтеграцію з сервісами друку на вимогу.

РОЗДІЛ 2.
ПРОЄКТУВАННЯ ТА РОЗРОБКА ОНЛАЙН-ПЛАТФОРМИ «ВИДАЛА ГРОМАДА» ДЛЯ КРАУДФАНДИНГУ КНИЖКОВИХ ПРОЕКТІВ

2.1. Постановка задачі, призначення та вимоги до онлайн платформи «Видала Громада»

Завданням кваліфікаційної роботи було створення технічної основи вебплатформи «Видала Громада», призначеної для підтримки книжкових проєктів у форматі краудфандингу. Метою розробки стало забезпечення ефективної взаємодії між авторами та потенційними читачами, надаючи обом сторонам зручні інструменти для комунікації, фінансування та представлення творчих ініціатив.

У межах роботи реалізовано початкову версію програмного засобу, яка включає розробку адаптивних вебсторінок відповідно до заданого дизайну, реалізацію ключового функціоналу, побудову архітектури серверної частини та забезпечення її інтеграції з інтерфейсом користувача. Серед окремих завдань варто відзначити налаштування структури вебсайту, реалізацію механізмів масштабування, а також чітке розмежування середовищ розробки (development) і використання (production), що стало передумовою для безпечного та передбачуваного розгортання проєкту.

Особливу увагу було приділено налаштуванню платформи для публічного доступу шляхом розгортання на Apache-сервері університету, що дало змогу забезпечити її постійну доступність та протестувати роботу в реальних умовах. Програмний засіб відповідає сучасним вимогам до безпеки, функціональності та стабільності, а його архітектура передбачає гнучке розширення функцій у майбутньому.

2.2 Загальний опис проєкту

2.2.1. Вибір моделі розробки

Під час розробки краудфандингової платформи «Видала Громада» було обрано ітеративний підхід, що дав змогу оперативно реагувати на зміни у вимогах і забезпечити гнучкість у процесі реалізації. Враховуючи те, що я, як розробник, мав прямий контакт із замовником, це значно пришвидшило узгодження технічних рішень і дозволило оперативно вносити необхідні корективи до функціоналу.

Проєкт стартував із розробки візуальної частини – дизайну, створеного у Figma. Дизайнер сформував загальну стилістику майбутнього застосунку, включаючи макети основних сторінок, враховуючи як естетичні, так і ергономічні вимоги. Співпраця між розробником і дизайнером здійснювалася безпосередньо, що дозволило адаптувати графічні рішення до особливостей обраної технологічної бази, а також вчасно виявляти потенційні обмеження.

Весь процес реалізації відбувався поетапно: кожна функціональна частина проєкту – від автентифікації користувачів до механізмів платежів – розроблялася окремо, із подальшим тестуванням та інтеграцією в загальну систему. Кожна ітерація завершувалася демонстрацією результатів замовнику, після чого зібраний зворотний зв’язок спрямовував подальший розвиток. Така методика, що відповідає практикам Agile[29], забезпечила постійний прогрес проєкту без втрати стабільності системи.

Постійний діалог із замовником сприяв точному формулюванню вимог до функціоналу та інтерфейсу, що у підсумку дало змогу досягти високого рівня відповідності очікуванням кінцевих користувачів. Особлива увага приділялася зручності навігації, зрозумілості взаємодії з елементами системи та швидкодії інтерфейсу.

Вимоги до основних функціональних модулів були формалізовані та частково візуалізовані на рисунках 2.1 та 2.2.

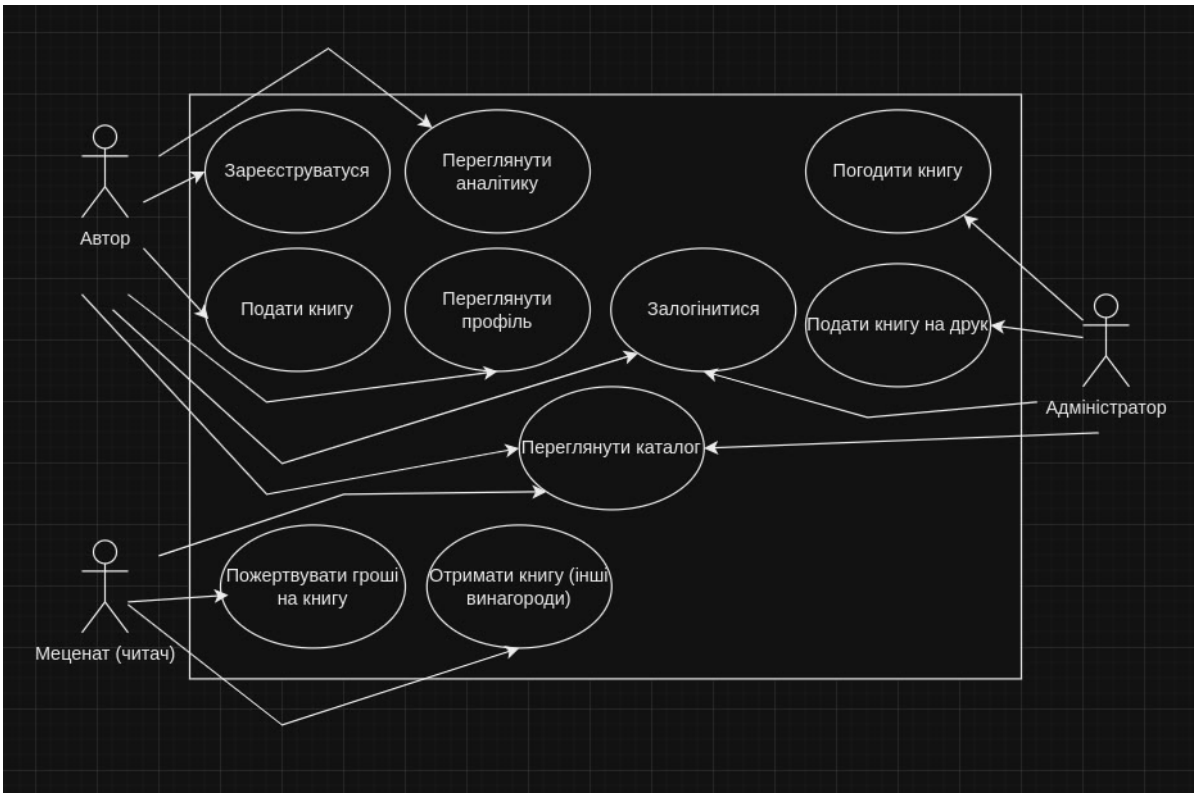

Рисунок 2.1 - Діаграма використання

Рисунок 2.2 - Діаграма послідовності

2.2.2. Структура бази даних

У межах реалізації онлайн-платформи для краудфандингу книжкових проєктів було спроєктовано реляційну базу даних[39], яка структурно охоплює всі ключові аспекти функціональності системи. Архітектура бази даних відображена на діаграмі сутностей (ER-діаграмі[40]), яка побудована відповідно до принципів нормалізації[41] Рис. 2.3. Це дозволяє ефективно зберігати та обробляти інформацію про користувачів, проєкти, платежі, винагороди, жанри та взаємодію між ними.

Центральною таблицею є Book, яка репрезентує окремий книжковий проєкт. Вона містить інформацію про назву книги, адресу її файлу, мову, дати початку та завершення збору коштів, короткий та розгорнутий опис, а також відеопрезентацію та зображення обкладинки. Кожен проєкт пов’язаний з певним користувачем через зовнішній ключ user_id, що вказує на автора або ініціатора збору коштів. Окрім цього, таблиця Book має зв’язки з іншими таблицями, які доповнюють функціональність системи.

Таблиця User зберігає облікову та особисту інформацію про користувачів, включаючи електронну пошту, зашифрований пароль, ім’я, прізвище, а також службові поля, пов’язані з підтвердженням та скиданням пароля. Через зовнішні ключі користувачі можуть залишати коментарі (Comment) до проєктів, а також здійснювати фінансову підтримку через платежі (Payment).

Коментарі зберігаються у таблиці Comment, яка містить текст повідомлення, а також зовнішні ключі до користувача й книги. Це дозволяє реалізувати функціональність спілкування та зворотного зв’язку між авторами та підтримувачами проєктів.

Таблиця Payment відповідає за зберігання інформації про пожертви. Кожен запис містить суму, валюту, статус транзакції, дату створення та оновлення, а також прив’язаний до конкретного проєкту за допомогою book_id. Це дозволяє системі прозоро відображати хід фінансування книжкових ініціатив.

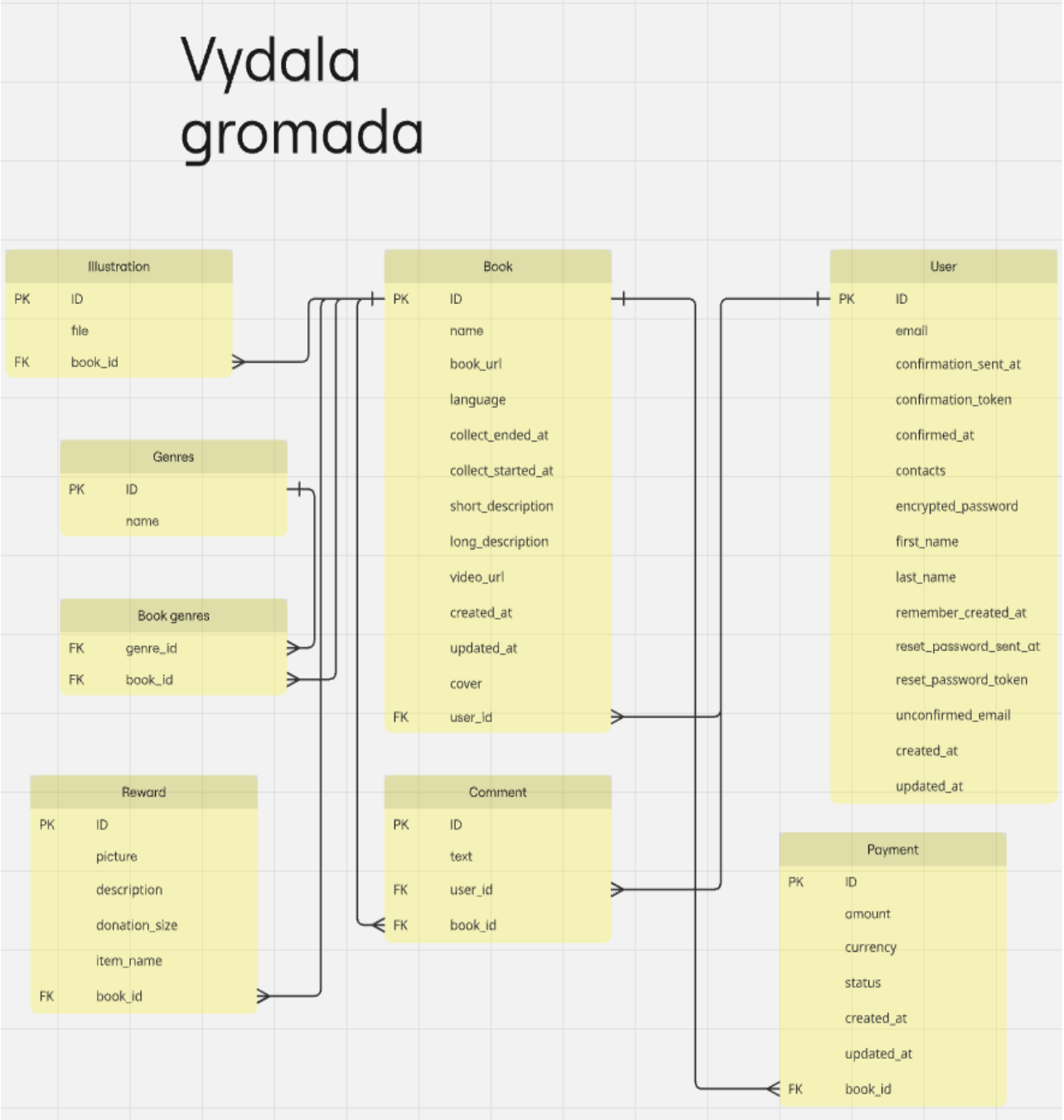

Рисунок 2.3 - Діаграма сутностей для структури бази даних

Для реалізації функції мотивації користувачів до донатів впроваджена таблиця Reward, яка зберігає опис винагород, пов’язаних із певними сумами пожертв. До кожної винагороди додається зображення та назва предмета, що пропонується користувачеві. Усі винагороди також прив’язані до відповідної книги.

Щоб підтримати мультимедійну складову платформи, у базі даних передбачена таблиця Illustration, яка дозволяє прикріплювати додаткові зображення до проєктів. Це надає змогу авторам краще візуалізувати свої ідеї та зацікавити потенційних донаторів.

Класифікація проєктів за жанрами реалізована через окрему таблицю Genres, що містить перелік можливих жанрових категорій. Зв’язок між книгами та жанрами здійснюється через проміжну таблицю BookGenres, яка забезпечує відношення типу «багато до багатьох». Це дає змогу одній книзі належати до кількох жанрів одночасно.

2.3. Обґрунтування вибору інструментальних засобів розробки «Видала Громада»

У процесі розробки краудфандингової платформи «Видала Громада» було обрано стек технологій, який поєднує простоту у використанні, швидкість розгортання і гнучкість при подальшій модифікації. Основою проєкту став фреймворк Ruby on Rails, який реалізує архітектуру MVC та дозволяє ефективно будувати вебзастосунки з повним циклом – від бекенду до фронтенду. Для роботи з базою даних обрано PostgreSQL. Вибір на користь реляційної моделі бази даних з використанням PostgreSQL зумовлений вимогами до надійності, масштабованості та інтеграції з фреймворком Ruby on Rails. Усі сутності проєкту спроєктовані відповідно до конвенцій Active Record, що дозволяє забезпечити ефективну взаємодію з базою даних на рівні прикладного коду. Передбачена структура є розширюваною й придатною до масштабування, що відкриває можливості для подальшого розвитку системи – наприклад, додавання системи рейтингів, тегів, інтеграції з платіжними сервісами чи модулів для статистичного аналізу ефективності кампаній.

Сама мова Ruby[36], на якій написано Rails, суттєво сприяє швидкій розробці завдяки своїй виразності та зручному синтаксису, що наближається до природної мови[37]. Це робить код легким для читання, підтримки й подальшого

розвитку навіть у великих командах. Ruby дотримується принципу «розробка з насолодою» (developer happiness)[38], завдяки чому багато типових завдань можна реалізувати лаконічно, без надмірної шаблонності. Крім того, динамічна типізація й розвинена об’єктна модель дозволяють гнучко підходити до побудови бізнес-логіки й легко адаптувати структуру застосунку під змінні вимоги.

Однією з переваг Rails є можливість розширення функціональності за допомогою так званих «гемів» – спеціалізованих бібліотек. У цьому проєкті використано низку таких бібліотек для реалізації основних модулів. Зокрема, Devise[13] забезпечує надійний функціонал для автентифікації користувачів, включаючи реєстрацію, авторизацію та керування сесіями. Pundit відповідає за контроль доступу до ресурсів відповідно до ролей користувачів.

Для роботи із зображеннями задіяні бібліотеки ImageProcessing та MiniMagick, які дозволяють масштабувати, стискати й обробляти графічні файли. Їхнє використання у поєднанні з ActiveStorage та ActiveStorageValidations дозволяє ефективно організувати завантаження файлів із клієнтської частини, включаючи перевірку типу, розміру і формату файлу.

На стороні клієнта використано бібліотеки Stimulus[19] і Turbo[20], що є складовими modern Rails підходу до фронтенд-розробки без потреби у важких фреймворках типу React чи Vue. Вони дозволяють реалізовувати асинхронні оновлення сторінки, взаємодію з формами та іншими елементами через звичайний HTML та JavaScript. Разом з Importmap-Rails [30], це спрощує підключення сторонніх JS-бібліотек без потреби у Webpack.

Для стилізації інтерфейсу використано Tailwind CSS[34] – сучасний утилітарний фреймворк, який дозволяє гнучко й швидко задавати стилі без надмірного CSS-коду. Завдяки системі класів розробка адаптивних та охайних інтерфейсів значно пришвидшується.

Pagy[33] реалізує швидку і гнучку пагінацію даних, що є важливою частиною для відображення довгих списків проєктів або новин. Redis виконує

роль кешу і використовується для прискорення взаємодії з базою даних та обробки фонових процесів.

Для генерації фіктивних даних під час тестування застосовано Faker[22] у зв'язці з FactoryBot[23], що дозволяє моделювати типові сценарії роботи користувачів. Інструменти Pry[24], Debug та Annotate[35] допомагають у налагодженні коду та аналізі структури бази даних, особливо на етапі активної розробки.

На клієнтській стороні додатково інтегровані бібліотеки Dropzone[25] – для зручного drag-and-drop завантаження файлів, та Trix[31] з ActionText – для створення текстового редактора з підтримкою форматування, зображень і посилань. Це дозволяє користувачам створювати привабливі та інформативні описи своїх проєктів.

Усі ці інструменти було обрано з урахуванням потреб проєкту, а також їх сумісності між собою, що забезпечило ефективну розробку й подальшу масштабованість платформи.

Серед багатьох доступних середовищ розробки було обрано Visual Studio Code (VS Code)[11] як основний інструмент для роботи над проєктом «Видала Громада». Цей вибір обумовлений кількома ключовими перевагами, які VS Code пропонує саме для розробки вебзастосунків із використанням таких технологій, як Ruby on Rails, JavaScript, HTML та CSS.

По-перше, VS Code є легким і водночас потужним редактором коду, який швидко запускається та не перевантажує систему, що важливо під час розробки в умовах обмежених ресурсів. Його підтримка великої кількості розширень, зокрема Ruby, Rails, ERB Syntax Highlighting [32], GitLens, Prettier та Live Server, значно спрощує робочий процес. Це дозволяє не тільки зручно працювати з шаблонами, стилями та логікою застосунку, а й швидко виявляти помилки та конфлікти у версіях коду.

По-друге, інтеграція з Git [27] через вбудований інтерфейс дозволяє легко здійснювати контроль версій, переглядати історію змін, створювати гілки та працювати з репозиторіями без необхідності перемикання на інші програми. Це

підвищує ефективність командної роботи і дозволяє краще відслідковувати етапи розробки.

Також важливим фактором стала висока гнучкість налаштувань VS Code, що дозволяє адаптувати середовище під власні потреби: від шрифтів і кольорових тем до макросів, фрагментів коду та автоматизації форматування.

Крім того, завдяки великій спільноті VS Code має вичерпну документацію та постійно оновлювану базу рішень, що пришвидшує вирішення технічних проблем і знижує поріг входу для розробників.

З урахуванням усіх перелічених чинників, Visual Studio Code є вдалим вибором для реалізації цього проєкту, поєднуючи простоту у використанні з багатофункціональністю, що повністю відповідає вимогам до сучасної розробки вебзастосунків.

2.4 Особливості програмної реалізації

Розглянемо головну сторінку застосунку (рис. 2.4) та відповідний метод контролера, що обслуговує її (2.5). При зверненні до цього методу здійснюється запит до бази даних для отримання обмеженої кількості книжок, що відображаються з використанням механізму пагінації. У даному випадку використовується кнопка «Показати більше» (рис. 2.6), яка ініціює довантаження нових книжок. Метод також обробляє HTTP-запити двох типів: при GET-запиті повертається стандартна HTML-сторінка, а при POST-запиті – відповідь формується у вигляді Turbo Stream, що дозволяє динамічно підвантажувати новий контент без повного перезавантаження сторінки.

На відповідному фрагменті коду (рис. 2.5) також видно перевірку на авторизацію користувача для методів new та create, що відповідають за створення нових книжок. Це забезпечує обмеження доступу: лише авторизовані користувачі можуть додавати нові проєкти. Беручи до уваги структуру проєкту, контролери для коментарів та оплат вважаються дочірніми для контролера

книжок та по-замовчуванню отримують дані книги до якої додається коментар або донат (рис. 2.7, рис. 2.8).

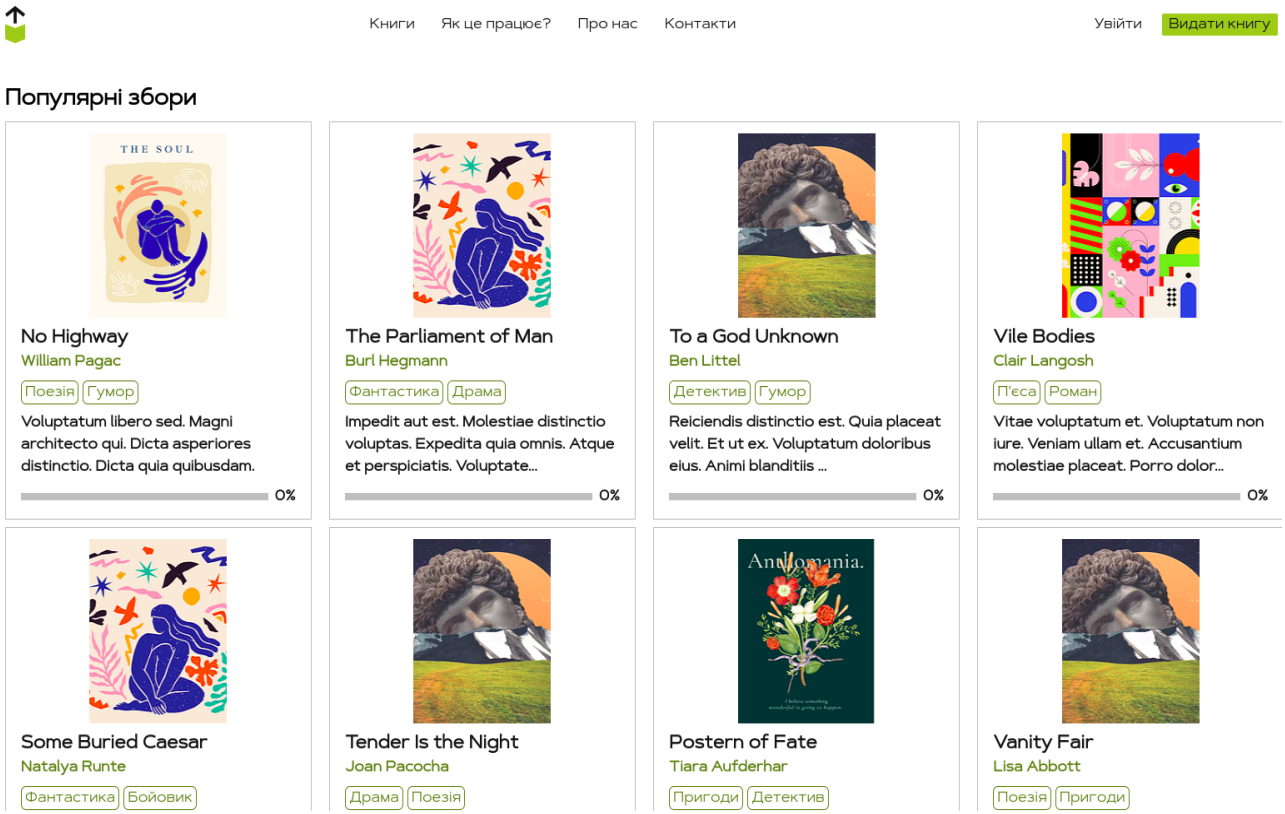

Рисунок 2.4 – Корінна і головна сторінка застосунку. Каталог доступних книг

```ruby
# frozen_string_literal: true

# Controller for books
class BooksController < ApplicationController
  before_action :authenticate_user!, only: %i[new create]

  def index
    @pagy, @books = pagy_countless(Book.all, items: 8)

    respond_to do |format|
      format.html
      format.turbo_stream
    end
  end
```

Рисунок 2.5 – Контролер шляху “/books”, метод index

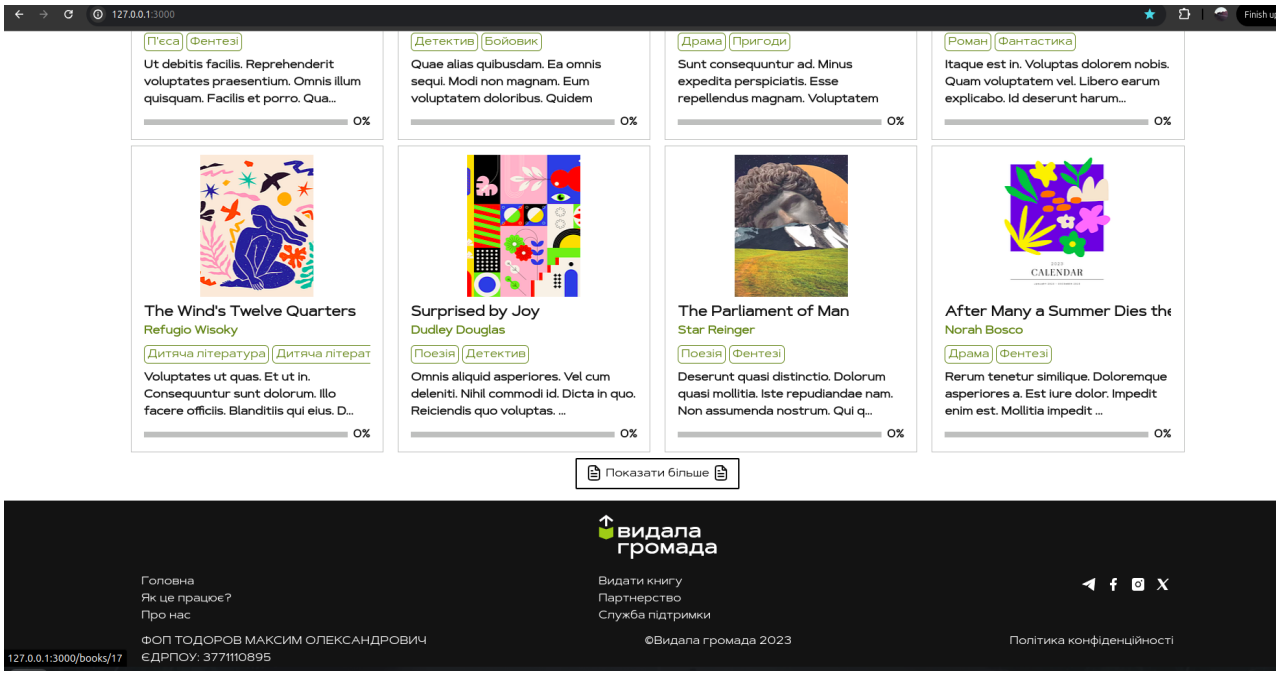

Рисунок 2.6 – Кнопка “Показати більше” та футер застосунку

```ruby
# frozen_string_literal: true

# Controller for comments
class CommentsController < ApplicationController
  before_action :set_book

  def create
    @comment = @book.comments.build(comment_params)
    @comment.user = current_user

    if @comment.save
      redirect_to @book, notice: 'Коментар додано!'
    else
      redirect_to @book, notice: 'Не вдалося зберегти коментар!'
    end
  end

  private

  def set_book
    @book = Book.find_by(id: params[:book_id])
  end

  def comment_params
    params.require(:comment).permit(:text)
  end
end
```

Рисунок 2.7 – Контролер коментарів

```ruby
# frozen_string_literal: true

# Controller for payments
class PaymentsController < ApplicationController
  before_action :set_book, only: %i[create]
  skip_before_action :verify_authenticity_token, only: %i[callback callback_page]

  def create
    @payment = @book.payments.new(payment_params)

    if @payment.valid?
      send_payment_data
    else
      redirect_to @book
    end
  end

```

Рисунок 2.8 - Контролер оплат

У процесі створення нової книжки (рис. 2.9) використовується стандартна пара методів – get для отримання форми (new) та post для її обробки (create). Додатково задаються дозволені параметри, які можуть бути зчитані або змінені під час збереження. У цьому блоці реалізована логіка вкладених атрибутів: книжка створює пов’язані з нею об’єкти нагород (рис. 2.10, рис. 2.11). Приватні методи реалізовують попередній пошук книжки у базі даних та створення хешу жанрів книги.

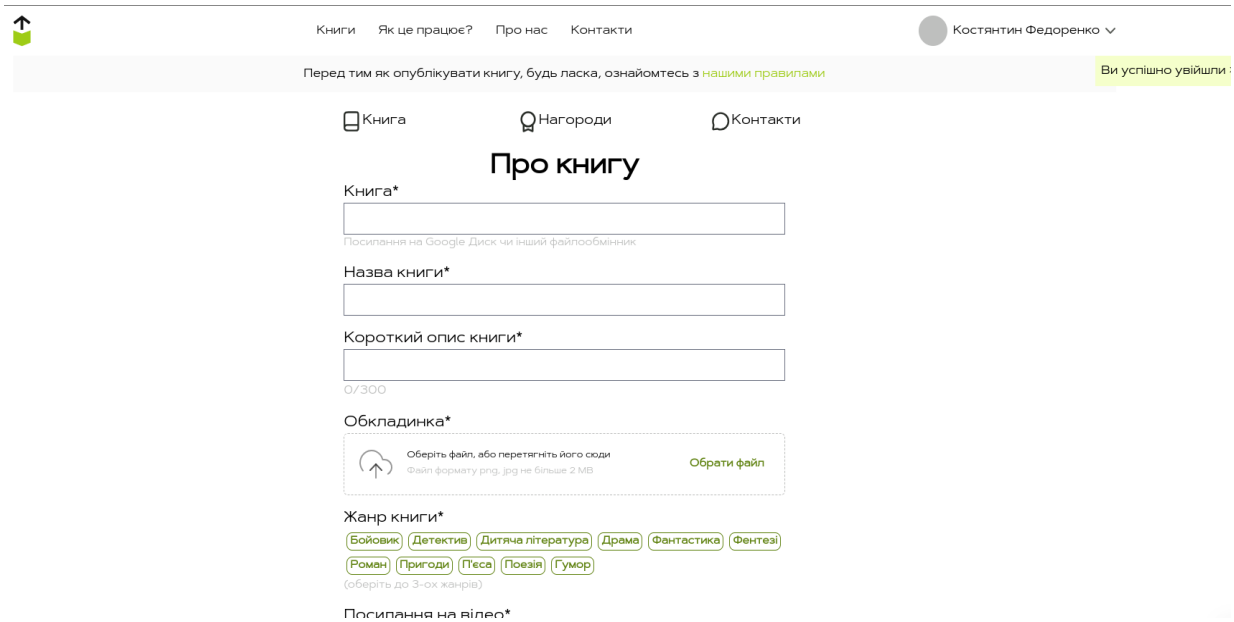

Рисунок 2.9 – сторінка створення нової книжки

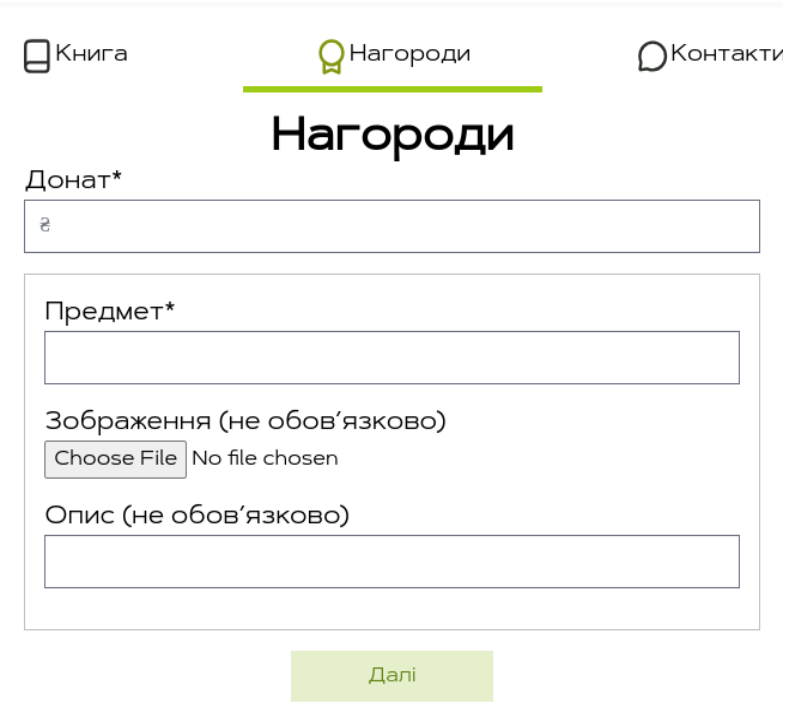

Рисунок 2.10. – Вкладка нагород

```ruby
private

def resource
  Book.find(params[:id])
end

def genres
  @genres = '['
  Genre.all.each do |genre|
    @genres += "{\"value\": \"#{genre.id}\", \"text\": \"#{genre.name}\"},"
  end
  @genres = @genres.chop
  @genres += ']'
end

def book_params
  params.require(:book).permit(
    :book_url, :name, :short_description,
    :video_url, :cover, :illustrations,
    :long_description, :language,
    rewards_attributes: %i[description donation_size item_name picture]
  ).merge(user_id: current_user.id)
end
end
```

Рисунок 2.11. – private методи books контролера

У фрагменті представлення (рис. 2.12) демонструється використання HTML-розмітки в поєднанні з TailwindCSS для створення візуальних плейсхолдерів, які показуються до того, як дані про книжки будуть довантажені. На наступному етапі (рис. 2.13) наведено складніший фрагмент, де реалізована інтеграція бібліотек Active Storage, Dropzone і Stimulus для реалізації direct upload – прямого завантаження файлів із клієнта на сервер, минаючи додаткові проміжні обробки.

```
<div class="placeholder-glow flex place-content-center border-gromada-gray border book-block mb-2">
  <div class="m-3 content-block">
    <div class="image-block flex place-content-center">
      <div class="w-[50%] mb-2 placeholder"></div>
    </div>
    <h5 class="text-xl font-bold col-6 placeholder"></h5>
    <div>
      <span class="col-5 placeholder"></span>
      <span class="col-5 placeholder"></span>
    </div>
    <div>
      <span class="col-5 placeholder"></span>
      <span class="col-5 placeholder"></span>
    </div>
    <div>
      <span class="col-5 placeholder"></span>
      <span class="col-5 placeholder"></span>
    </div>
  </div>
</div>
```

Рисунок 2.12. – view елемент з placeholders

```
<%= render "shared/error_messages", resource: @book %>
<%= form_with model: @book do |f| %>
  <div class="flex place-content-center">
    <div id="book_section" class="w-[40%]">
      <h1 class="text-4xl text-center font-extrabold">Про книгу</h1>
      <%= f.label :book_url, "Книга*", class: "text-xl"%><br />
      <%= f.text_field :book_url, class: "w-full" %>
      <p class="text-sm text-gromada-gray mt-[1px] mb-4">Посилання на Google Диск чи інший файлообмінник</p>

      <%= f.label :name, "Назва книги*", class: "text-xl"%><br />
      <%= f.text_field :name, class: "mb-4 w-full" %><br />

      <div data-controller="show-limit">
      <%= f.label :short_description, "Короткий опис книги*", class: "text-xl" %><br />
      <%= f.text_field :short_description, "maxLength": "300", class: "w-full", data: { show_limit_target: "field", action: "input->show-limit#update" } %>
      <p data-show-limit-target="text" class="text-gromada-gray mt-[1px] mb-4">0/300</p>
      </div>

      <%= f.label :cover, "Обкладинка*", class: "text-xl"%><br />
      <div class="mb-4 cursor-pointer border-dashed border-[1.5px] p-5 flex rounded-md border-gromada-gray dropzone dropzone-default dz-clickable" data-controller
        <%= f.file_field :cover, direct_upload: true, data: { dropzone_target: 'input' } %>
        <div class="flex place-content-between w-full dropzone-msg text-gray-600 needsclick dz-message">
          <div class="flex">
            <%= image_tag "upload.svg", class: "pr-5" %>
            <div>
              <h3 class="text-xs dropzone-msg-title">Оберіть файл, або перетягніть його сюди</h3>
              <span class="text-xs text-gromada-gray dropzone-msg-desc">Файл формату png, jpg не більше 2 MB</span>
            </div>
          </div>
          <div class="font-bold p-2 content-center text-sm text-gromada-dark-green">Обрати файл</div>
        </div>
      </div>
```

Рисунок 2. 13. – частина view для створення нової книги

Модель книжки (рис. 2.14) містить визначення зв’язків між таблицями бази даних, що дозволяють зберігати як текст з редактора Trix, так і прикріплені файли.

```ruby
# Table name: books
#
#  id                 :bigint           not null, primary key
#  book_url           :string
#  collect_ended_at   :datetime
#  collect_started_at :datetime
#  language           :string
#  name               :string
#  short_description  :string
#  video_url          :string
#  created_at         :datetime         not null
#  updated_at         :datetime         not null
#  user_id            :bigint
#
# Indexes
#
#  index_books_on_user_id  (user_id)
#
class Book < ApplicationRecord
  has_one_attached :cover

  has_many_attached :illustrations

  has_rich_text :long_description

  belongs_to :user

  has_many :books_genres, dependent: :destroy
  has_many :genres, through: :books_genres
  has_many :rewards, dependent: :destroy
  has_many :payments, dependent: :nullify
  has_many :comments, dependent: :nullify

  accepts_nested_attributes_for :rewards, allow_destroy: true

  validates :cover, attached: true, content_type: ['image/png', 'image/jpeg']

  def collected_amount
    payments.where(status: 'approved').sum(:amount)
  end

  def collected_amount_uah
    collected_amount / 100.0
  end
end
```

Рисунок 2.14 – модель книги

Тут реалізовано складний зв’язок типу has_many :through між книжками й жанрами (одна книжка може мати багато жанрів, і жанр – багато книжок). Має багато коментарів та оплат (payments). Реалізовує методи обчислення кількості задоначених коштів у копійках і гривнях. Також передбачено валідацію завантаженої обкладинки на формат PNG або JPG, а пов’язані об’єкти налаштовано так, щоб знищувалися разом із основним записом.

Це лише частина коду, але вона демонструє ключові технічні рішення, закладені в архітектуру проєкту.

2.5. Організація тестування та налагодження програмного засобу «Видала Громада»

У процесі розробки програмного засобу «Видала Громада» тестування проводилося паралельно з написанням коду, що дозволяло швидко виявляти та усувати помилки на ранніх етапах. Основний акцент було зроблено на ручне (мануальне) тестування [28], оскільки саме воно найкраще відображає досвід реального користувача й дозволяє оцінити функціональність інтерфейсу та загальну зручність користування.

На початковому етапі було проведено аналіз вимог до застосунку, що дало змогу сформувати чітке розуміння як функціональних, так і нефункціональних характеристик системи. Виходячи з цього, формувався план тестування, який включав детальні сценарії перевірки основних функцій: реєстрації користувача, створення й перегляду книжок, фільтрації, додавання зображень та інше.

Розгортання продукту було здійснено на тестовому сервері хостингу університету, що працює під керуванням операційної системи Ubuntu 22.04.2 LTS (GNU/Linux 5.15.0-73-generic x86_64). Серверне середовище забезпечувалося контейнеризованою інфраструктурою за допомогою Docker, що дозволяло спростити конфігурацію та ізолювати програмні компоненти. Для коректної роботи клієнтської частини вебзастосунку необхідна підтримка браузером технології Drag and Drop API.

Ручне тестування дозволяло не лише перевіряти окремі функції, а й оцінити взаємодію між компонентами системи в умовах, наближених до реальних. Після кожного виявленого дефекту проводилося усунення помилки з подальшим ретестингом – повторною перевіркою для підтвердження результату. Окрему увагу було приділено кросбраузерному тестуванню: функціональність вебзастосунку перевірялася в різних браузерах (зокрема Chrome, Firefox та Safari), що дозволило виявити потенційні відмінності у відображенні та забезпечити стабільну роботу системи в різних середовищах.

Попри трудомісткість, ручне тестування дало змогу гнучко реагувати на зміни й швидко адаптуватися до виявлених проблем. Для підвищення ефективності та зменшення навантаження в проєкті також передбачалося використання засобів автоматизованого тестування, таких як Rspec [9], Capybara [10] та SimpleCov [8]. RSpec дозволяє створювати модульні тести з читабельною структурою, Capybara – моделювати поведінку користувача у браузері, а SimpleCov – відстежувати покриття коду тестами. Втім, через обмеженість у часі та відносну складність налаштування, основне тестування залишалося ручним, а автоматизовані тести застосовувалися лише для окремих компонентів.

У підсумку, обраний підхід до тестування – з акцентом на мануальну перевірку та часткову автоматизацію – дозволив упевнитися у стабільності роботи вебзастосунку, відповідності його функціоналу поставленим вимогам і забезпечити якісний користувацький досвід.

2.6. Рекомендації по використанню та впровадженню програмного засобу «Видала Громада»

Оскільки користувачі платформи виконуватимуть різні ролі, кожна з них потребує окремого підходу до взаємодії із системою.

Для авторів книг важливо максимально наповнювати свої проєкти змістовним і привабливим контентом, який зможе зацікавити потенційних меценатів і читачів. Рекомендується використовувати фотографії,

відеоматеріали, уривки з майбутніх книжок або особисті історії, що розкривають значущість і унікальність проєкту. Якісне оформлення сторінки значно підвищує шанси на успішне залучення фінансування.

Меценати та читачі можуть обирати проєкти, орієнтуючись на свої жанрові уподобання, рекомендації інших користувачів, рівень зібраної підтримки та змістовність опису. Для зручності оплати передбачено використання різних платіжних систем, що робить фінансування доступним для широкого кола користувачів.

Адміністратори платформи відповідають за загальну якість і безпеку контенту. Їхні обов’язки включають перевірку завантажених матеріалів для запобігання публікації плагіату, неякісної або забороненої інформації. Крім того, адміністратори здійснюють аналітику поведінки користувачів з метою вдосконалення функціональності системи та покращення користувацького досвіду. Це дозволяє підтримувати високий рівень довіри до платформи та задовольняти потреби всіх її учасників.

ВИСНОВКИ

У результаті розробки та впровадження програмного засобу «Видала Громада» можна зробити кілька важливих висновків. Передусім, проєкт має чітко виражену соціальну мету – він відкриває авторам можливість отримати фінансову підтримку для видання книжок без залучення великих видавництв. Це особливо актуально для молодих, незалежних письменників і авторів, що працюють у нішевих жанрах. Завдяки цьому платформа сприяє розвитку сучасної літератури та активізує читачів, заохочуючи їх до безпосередньої участі у культурному процесі.

Технічно проєкт поєднує краудфандинг із інструментами керування проєктами, що дозволяє авторам ефективно вести кампанії збору коштів, підтримувати зв’язок з меценатами та оновлювати інформацію про стан реалізації своїх ідей. Простий, інтуїтивний інтерфейс забезпечує зручність користування як для творців контенту, так і для читачів, а адаптація під мобільні пристрої потенційно відкриває доступ до ширшої аудиторії.

Серверна частина системи побудована з використанням сучасних вебтехнологій, зокрема фреймворку Ruby on Rails, інструментів кешування Redis та бібліотек для тестування, таких як RSpec і Capybara. Це забезпечує масштабованість, стабільну роботу та можливість швидкого розширення функціональності. У процесі розробки особлива увага приділялася якості продукту – поєднання ручного тестування з автоматизованими сценаріями дозволило досягти високої надійності платформи та зменшити кількість потенційних помилок у майбутньому використанні.

Платформа має потенціал до монетизації, зокрема через комісію з реалізованих проєктів, впровадження додаткових платних сервісів для авторів, а також співпрацю з видавництвами або книготорговельними мережами. З огляду на архітектуру проєкту, можливе також подальше масштабування: інтеграція з зовнішніми сервісами, створення мобільного додатку, впровадження нових механік підтримки авторів.

Разом із тим, перед платформою стоять і певні виклики. Найважливішими з них є залучення достатньої кількості активних користувачів, підтримка конкурентоспроможності серед подібних сервісів, а також забезпечення високого рівня безпеки персональних даних. Успішне вирішення цих завдань стане запорукою життєздатності та довготривалого функціонування платформи.

У підсумку, проєкт «Видала Громада» можна розглядати як перспективний з точки зору як соціального впливу, так і економічного потенціалу. Він створює сприятливе середовище для підтримки творчих ініціатив, розкриття літературного потенціалу авторів та активної участі читачів у процесі створення книжок, сприяючи тим самим розвитку культурного середовища загалом.

СПИСОК ДЖЕРЕЛ

1. Краудфандинг [Електронний ресурс] // Вікіпедія. – URL: https://uk.wikipedia.org/wiki/Краудфандинг (дата звернення: 20.05.2025).
2. Спільнокошт: платформа для громадських проєктів [Електронний ресурс] // Велика Ідея. – URL: https://biggggidea.com/ (дата звернення: 20.05.2025).
3. 20 найпопулярніших краудфандингових платформ [Електронний ресурс] // UAspectr. – URL: https://uaspectr.com/2022/03/14/crowdfunding-platforms/ (дата звернення: 20.05.2025).
4. Patreon: про платформу [Електронний ресурс] // Офіційний сайт Patreon. – URL: https://www.patreon.com/ (дата звернення: 20.05.2025).
5. Ruby on Rails [Електронний ресурс] // Офіційний сайт Ruby on Rails. – URL: https://rubyonrails.org/ (дата звернення: 20.05.2025).
6. Redis: офіційна документація [Електронний ресурс] // Офіційний сайт Redis. – URL: https://redis.io/documentation (дата звернення: 20.05.2025).
7. Booknet: літературна платформа [Електронний ресурс] // Офіційний сайт Booknet. – URL: https://booknet.com/uk (дата звернення: 20.05.2025).
8. SimpleCov: Ruby Code Coverage [Електронний ресурс] // Офіційний сайт SimpleCov. – URL: https://simplecov.io/ (дата звернення: 20.05.2025).
9. RSpec: Behaviour Driven Development for Ruby [Електронний ресурс] // Офіційний сайт RSpec. – URL: https://rspec.info/ (дата звернення: 20.05.2025).
10. Capybara: інтеграційне тестування вебзастосунків [Електронний ресурс] // Офіційний сайт Capybara. – URL: https://teamcapybara.github.io/capybara/ (дата звернення: 20.05.2025).
11. VS Code Documentation [Електронний ресурс] // Visual Studio Code Docs. – URL: https://code.visualstudio.com/docs (дата звернення: 20.05.2025).
12. Yakaboo: книжковий маркетплейс [Електронний ресурс] // Офіційний сайт Yakaboo. – URL: https://www.yakaboo.ua/ (дата звернення: 20.05.2025).

13. Devise Gem [Електронний ресурс] // GitHub репозиторій Devise. – URL: https://github.com/heartcombo/devise (дата звернення: 20.05.2025).
14. Sidekiq Gem [Електронний ресурс] // Офіційна сторінка Sidekiq. – URL: https://sidekiq.org/ (дата звернення: 20.05.2025).
15. Unbound: платформа краудфандингу для книг [Електронний ресурс] // Офіційний сайт Unbound. – URL: https://unbound.com/ (дата звернення: 20.05.2025).
16. PostgreSQL: документація [Електронний ресурс] // Офіційний сайт PostgreSQL. – URL: https://www.postgresql.org/docs/ (дата звернення: 20.05.2025).
17. Open Banking: що це таке [Електронний ресурс] // AIN.UA. – URL: https://ain.ua/2021/10/25/shho-take-open-banking/ (дата звернення: 20.05.2025).
18. Fondy: інструкція інтеграції [Електронний ресурс] // Офіційний сайт Fondy. – URL: https://fondy.io/ua/docs/ (дата звернення: 20.05.2025).
19. Stimulus JS [Електронний ресурс] // Stimulus Documentation. – URL: https://stimulus.hotwired.dev/ (дата звернення: 20.05.2025).
20. Hotwire: офіційна документація [Електронний ресурс] // Hotwire.dev. – URL: https://hotwired.dev/ (дата звернення: 20.05.2025).
21. YouTube Data API: офіційна документація [Електронний ресурс] // Google for Developers. – URL: https://developers.google.com/youtube/v3 (дата звернення: 20.05.2025).
22. Faker Gem [Електронний ресурс] // GitHub репозиторій Faker. – URL: https://github.com/faker-ruby/faker (дата звернення: 20.05.2025).
23. FactoryBot Gem [Електронний ресурс] // Thoughtbot. – URL: https://thoughtbot.github.io/factory_bot/ (дата звернення: 20.05.2025).
24. Pry: Ruby REPL [Електронний ресурс] // GitHub репозиторій Pry. – URL: https://github.com/pry/pry (дата звернення: 20.05.2025).

25. Dropzone.js: сучасне drag’n’drop завантаження файлів [Електронний ресурс] // Офіційний сайт Dropzone. – URL: https://www.dropzone.dev/ (дата звернення: 20.05.2025).
26. Docker: офіційна документація [Електронний ресурс] // Docker Docs. – URL: https://docs.docker.com/ (дата звернення: 20.05.2025).
27. Git: розподілена система керування версіями [Електронний ресурс] // Git SCM. – URL: https://git-scm.com/doc (дата звернення: 20.05.2025).
28. Чому тестування критично важливе у розробці [Електронний ресурс] // ITC.UA. – URL: https://itc.ua/ (дата звернення: 20.05.2025).
29. Atlassian: Тестування в Agile [Електронний ресурс] // Atlassian Docs. – URL: https://www.atlassian.com/agile/testing (дата звернення: 20.05.2025).
30. Importmap для Rails [Електронний ресурс] // Ruby on Rails Guides. – URL: https://guides.rubyonrails.org/working_with_javascript_in_rails.html#using-importmap (дата звернення: 20.05.2025).
31. Trix Editor [Електронний ресурс] // Офіційний сайт Basecamp. – URL: https://trix-editor.org/ (дата звернення: 28.05.2025).
32. ERB (Embedded Ruby) [Електронний ресурс] // Ruby-Doc.org. – URL: https://ruby-doc.org/stdlib-2.7.0/libdoc/erb/rdoc/ERB.html (дата звернення: 28.05.2025).
33. Pagy: легкий пагінатор для Ruby on Rails [Електронний ресурс] // Офіційний сайт Pagy. – URL: https://ddnexus.github.io/pagy/ (дата звернення: 28.05.2025).
34. Tailwind CSS: утилітарний CSS-фреймворк [Електронний ресурс] // Офіційний сайт Tailwind CSS. – URL: https://tailwindcss.com/ (дата звернення: 28.05.2025).
35. Annotate Models: Ruby Gem для додавання коментарів у моделі [Електронний ресурс] // GitHub. – URL: https://github.com/ctran/annotate_models (дата звернення: 28.05.2025).

36. Ruby Language, мова програмування [Електронний ресурс] // Ruby Official Site. – URL: https://www.ruby-lang.org/en/about/ – (дата звернення: 28.05.2025).
37. Why Ruby? [Електронний ресурс] // Ruby on Rails Guides. – URL: https://guides.rubyonrails.org/why_rails.html – (дата звернення: 28.05.2025).
38. Yukihiro Matsumoto: The Philosophy of Ruby [Електронний ресурс] // O'Reilly Media. – URL: https://www.oreilly.com/library/view/the-ruby-programming/9780596516178/ch01.html – (дата звернення: 28.05.2025).
39. Реляційні бази даних. Теорія і практика [Електронний ресурс] // W3Schools SQL Tutorial. – URL: https://www.w3schools.com/sql/sql_intro.asp – (дата звернення: 28.05.2025).
40. Entity Relationship Diagram (ERD) Tutorial [Електронний ресурс] // Lucidchart Blog. – URL: https://www.lucidchart.com/pages/er-diagrams – (дата звернення: 28.05.2025).
41. Database Normalization: A Step-By-Step Guide [Електронний ресурс] // Vertabelo Academy. – URL: https://academy.vertabelo.com/blog/database-normalization/ – (дата звернення: 28.05.2025).

ДОДАТКИ

Додаток А

Технічне завдання

У межах технічного завдання визначено основні функціональні вимоги, реалізація яких є критичною для забезпечення повної працездатності системи та відповідності її поставленим цілям.

Передусім, проєкт передбачає створення каталогу книжкових краудфандингових ініціатив. Цей каталог має відображати список доступних проєктів у форматі інтерактивних карток, кожна з яких містить ключову інформацію про книгу, включно з назвою, обкладинкою, жанром, коротким описом і графічним індикатором прогресу збору коштів. Навігація каталогом має бути зручною та безперервною завдяки динамічному довантаженню нових елементів без повного перезавантаження сторінки.

Функціонал аутентифікації користувачів охоплює процедури реєстрації, входу, виходу, а також реалізацію механізмів підтвердження електронної адреси під час реєстрації та відновлення паролю шляхом надсилання відповідного листа користувачеві. Такі заходи забезпечують базовий рівень захисту даних і контроль доступу до розширених можливостей платформи.

Створення нового книжкового проєкту реалізується у вигляді багатосторінкової форми, що розділена на три логічні вкладки: «Книга», «Нагороди» та «Контакти». У вкладці «Книга» автор надає обкладинку, додає ілюстрації, вказує жанри, посилання на повний текст (наприклад, у хмарному сховищі), а також формує розширений опис твору. У вкладці «Нагороди» визначаються базова ціна книги та додаткові винагороди для підтримувачів, залежно від розміру донату. Вкладка «Контакти» включає поля зворотного зв’язку та кнопку надсилання заявки, яка підлягає обов’язковій валідації.

Окрему частину функціональних вимог становить реалізація сторінки проєкту книги. На цій сторінці користувачі мають змогу ознайомитися з повною інформацією про ініціативу, зокрема з обкладинкою, мовою написання,

детальним описом, відеозверненням від автора, а також залишити коментар. Автор книги має доступ до функції редагування проєкту. Крім того, на сторінці доступна кнопка «Передзамовити», що відкриває модальне вікно для введення бажаної суми підтримки. Після підтвердження введеної суми користувач перенаправляється на сторінку платіжної системи Fondy, де може здійснити оплату банківською карткою. Таким чином, здійснено інтеграцію з зовнішнім фінансовим сервісом для безпечної та швидкої обробки платежів.

Важливим компонентом технічного завдання є інтерфейс адміністратора та розробника, який передбачає наявність прихованого меню. Це меню дозволяє під час тестування миттєво перемикати стан сесії, минаючи повторні дії автентифікації, що істотно спрощує процес налагодження системи.

Інтерфейс користувача повинен містити навігаційну панель із кнопками для входу до системи та створення нового проєкту. У нижній частині сторінки розташовується футер з інформацією про платформу. У разі спроби неавторизованого створення проєкту має з’являтися відповідне сповіщення про необхідність входу до системи.

Додаток В

Інструкція Користувачу

Основною сторінкою вебзастосунку є каталог книг (рис. 2.3), у якому представлено проєкти, що вже збирають кошти. Кожен книжковий проєкт супроводжується обкладинкою, назвою, жанром, коротким описом і візуальним індикатором прогресу збору коштів. Це дозволяє користувачам швидко зорієнтуватися в доступних варіантах і підтримати ті ініціативи, що їх зацікавили.

На нижній частині сторінки (рис. 2.4) розміщено кнопку для довантаження наступних книг каталогу, що забезпечує зручну навігацію без перезавантаження сторінки. Також внизу знаходиться футер із додатковою інформацією про платформу, а у верхній частині – навігаційна панель із кнопками «Видати книгу» та «Увійти». Обидві ці кнопки ведуть на сторінку входу, однак натискання кнопки «Видати книгу» супроводжується сповіщенням про необхідність авторизації (рис. В.1). Крім того, користувач має можливість зареєструватися (рис. В.2) або скористатися функцією відновлення паролю (рис. В.3). Усі ці дії супроводжуються автоматичними листами, що надсилаються через робочий мейлер для підтвердження електронної адреси або скидання паролю.

Сторінка створення запиту на збір коштів містить три основні вкладки: «Книга» (рис. 2.9), «Нагороди» (рис. 2.10) та «Контакти» (рис. В.4). У вкладці «Книга» автор може завантажити обкладинку, вибрати жанри, додати ілюстрації (рис. В.5), вказати посилання на саму книгу (наприклад, на Google Drive), а також заповнити розширений опис із використанням зручної текстової панелі. Вкладка «Нагороди» дозволяє встановити базову вартість книги для передзамовлення і запропонувати додаткові подарунки при вищому донаті. У вкладці «Контакти» передбачені відповідні поля для зворотного зв’язку й кнопка надсилання заявки.

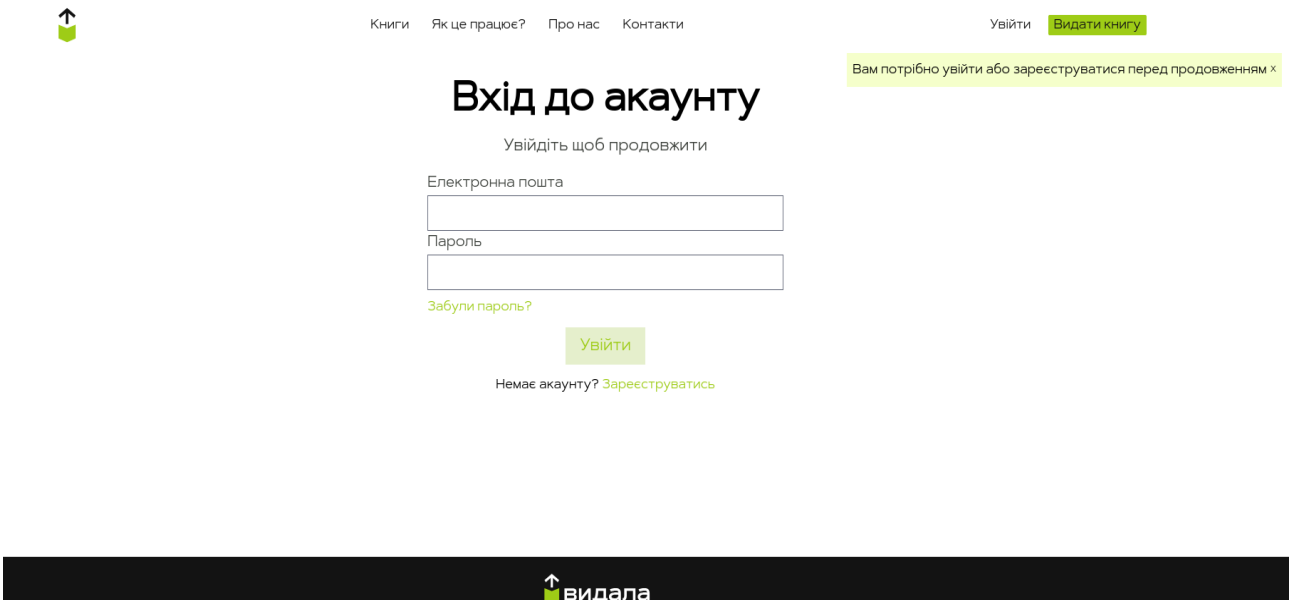

Рисунок В.1 – Сторінка входу в обліковий запис

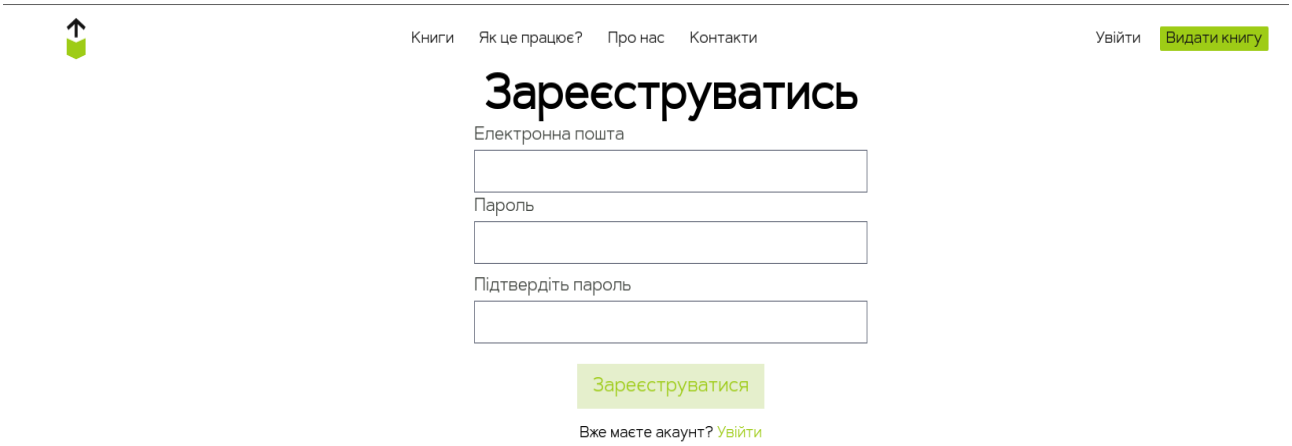

Рисунок В.2. – сторінка реєстрації

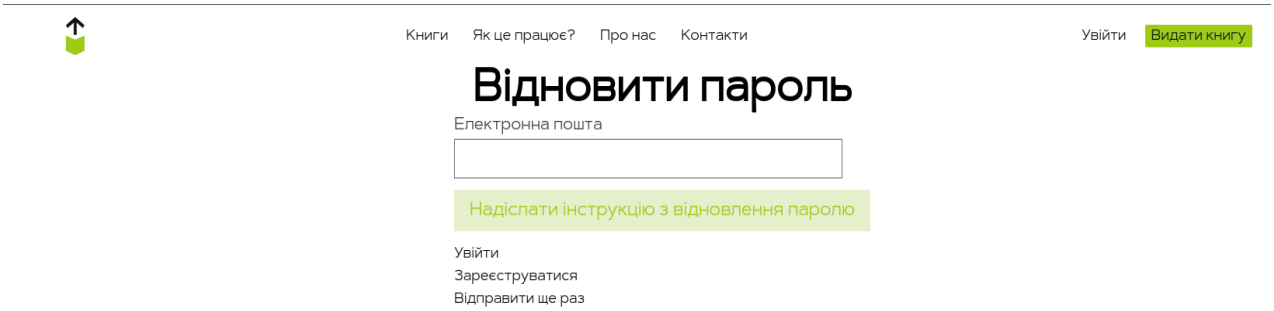

Рисунок В.3. – сторінка відновлення паролю

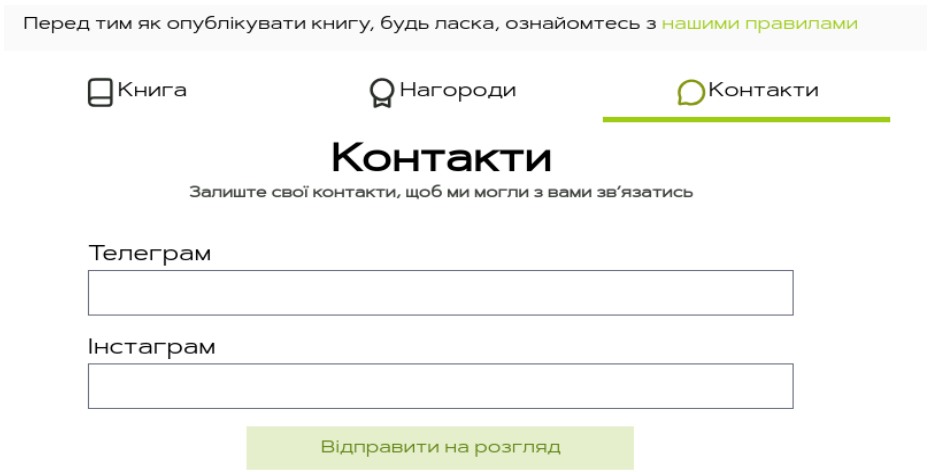

Рисунок В.4. – Вкладка контактів

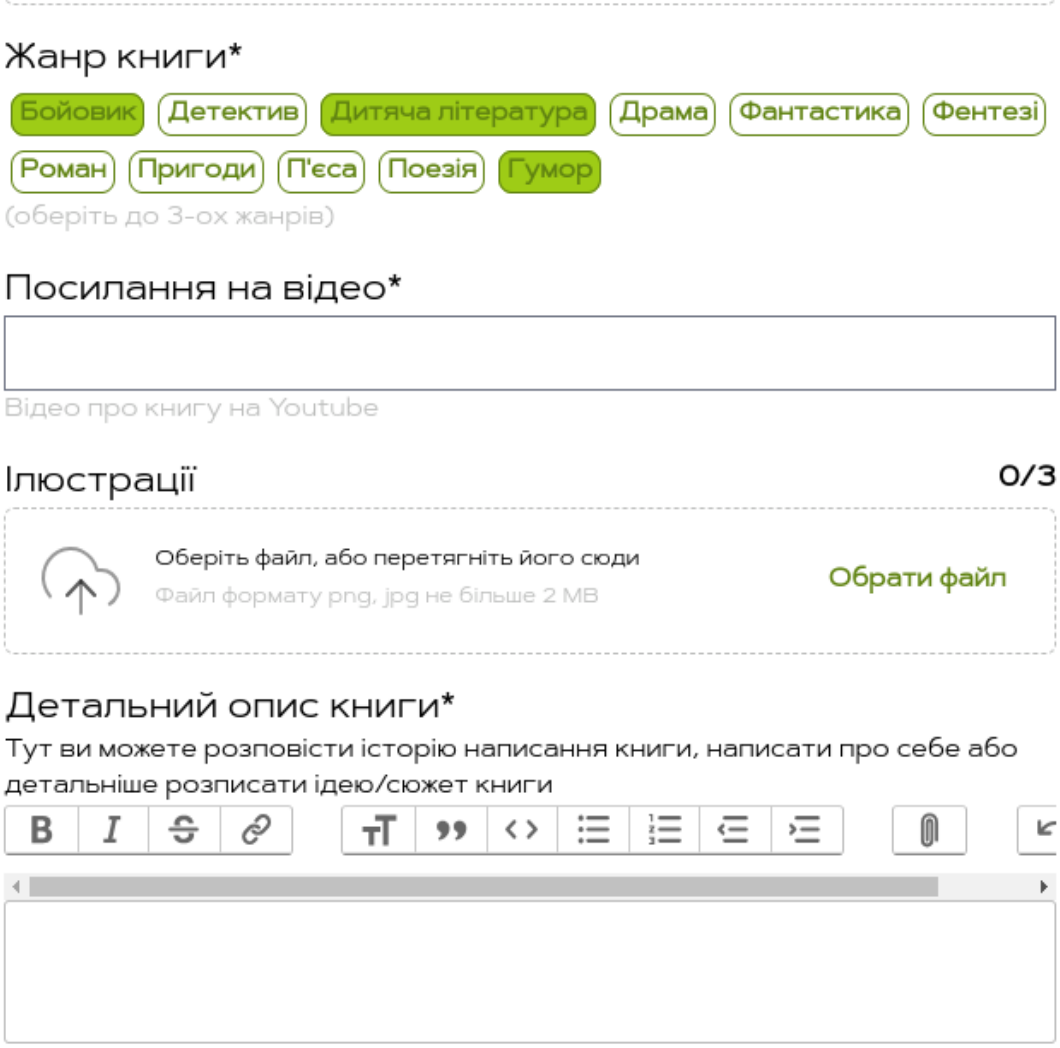

Рисунок В.5 – Поле для посилання на відео, вибір тегів жанру книги, завантаження ілюстрацій і можливість заповнення детального опису з додатковими інструментами

Після коректного заповнення всіх полів користувач автоматично переходить на сторінку з повною інформацією про книгу (рис. В.6), де проєкт уже доступний для підтримки.

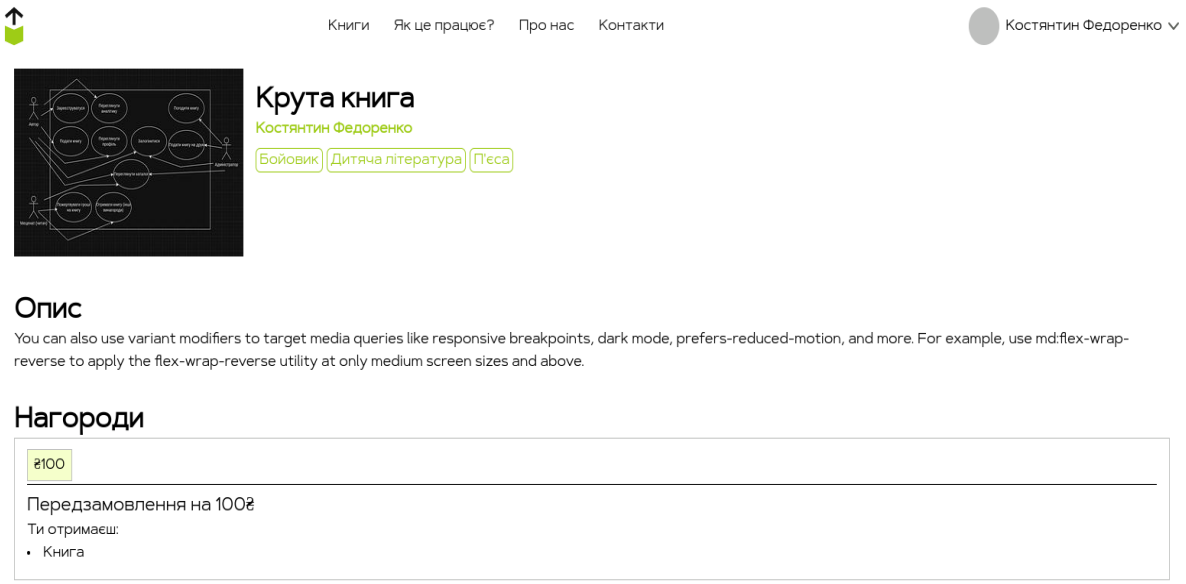

Рисунок В.6. – Сторінка книги

Для зручності тестування в режимі розробника в платформу вбудовано приховане меню (рис. В.7) у верхньому правому кутку. Воно дозволяє миттєво переходити до потрібного стану сесії, минаючи повторну авторизацію.

Під час перегляду сторінки книжки доступна опція передзамовлення (рис. В.8). При натисканні відповідної кнопки відкривається модальне вікно (рис. В.9), у якому користувач може самостійно вказати суму донату, яку бажає внести на підтримку автора. Після введення суми та підтвердження дії користувач автоматично перенаправляється на сторінку платіжного сервісу Fondy (рис. В.10), де може здійснити оплату зручним для себе способом.

Завдяки такій інтеграції з платіжною системою процес передзамовлення книги є простим, прозорим і швидким, що суттєво покращує користувацький досвід та підвищує ймовірність отримання авторами необхідної фінансової підтримки.

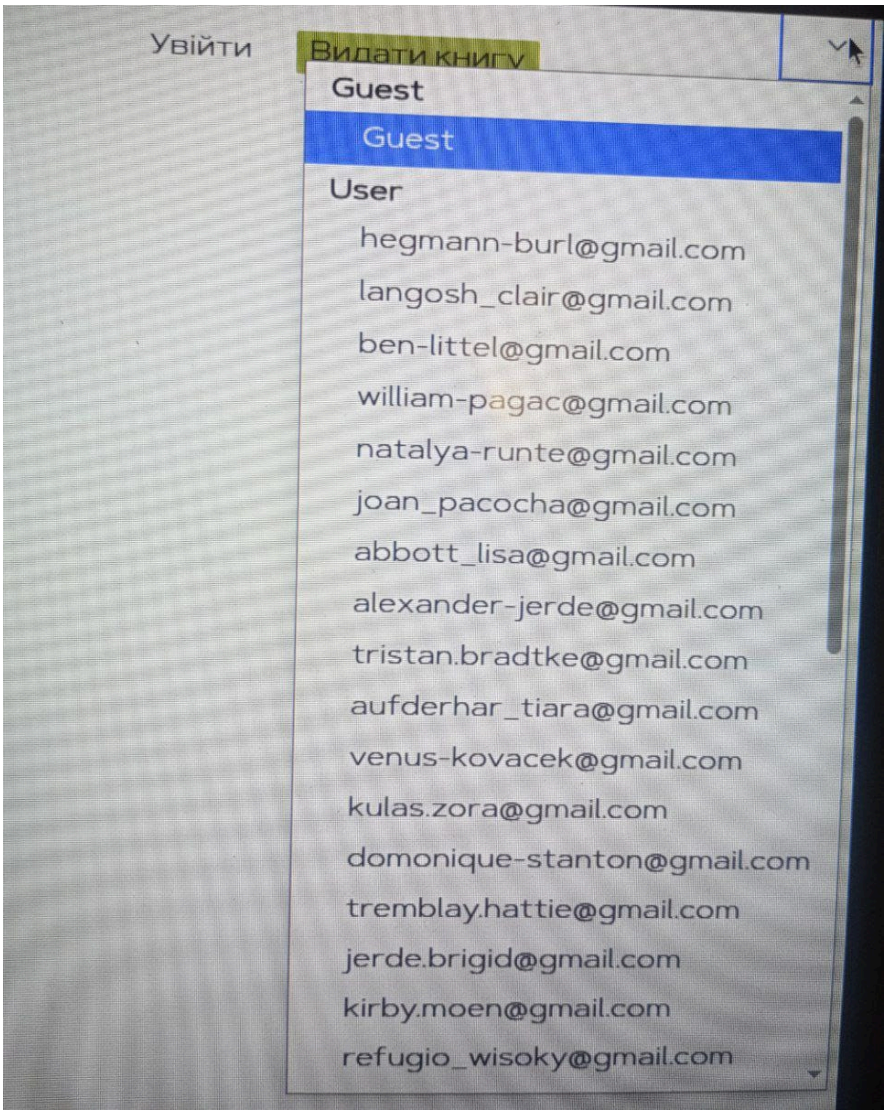

Рисунок В.7. – перемикання між користувачами у режимі тестування/розробника

Рисунок В.8 – Частина сторінки книги яка належить поточному користувачеві (кнопка редагування присутня)

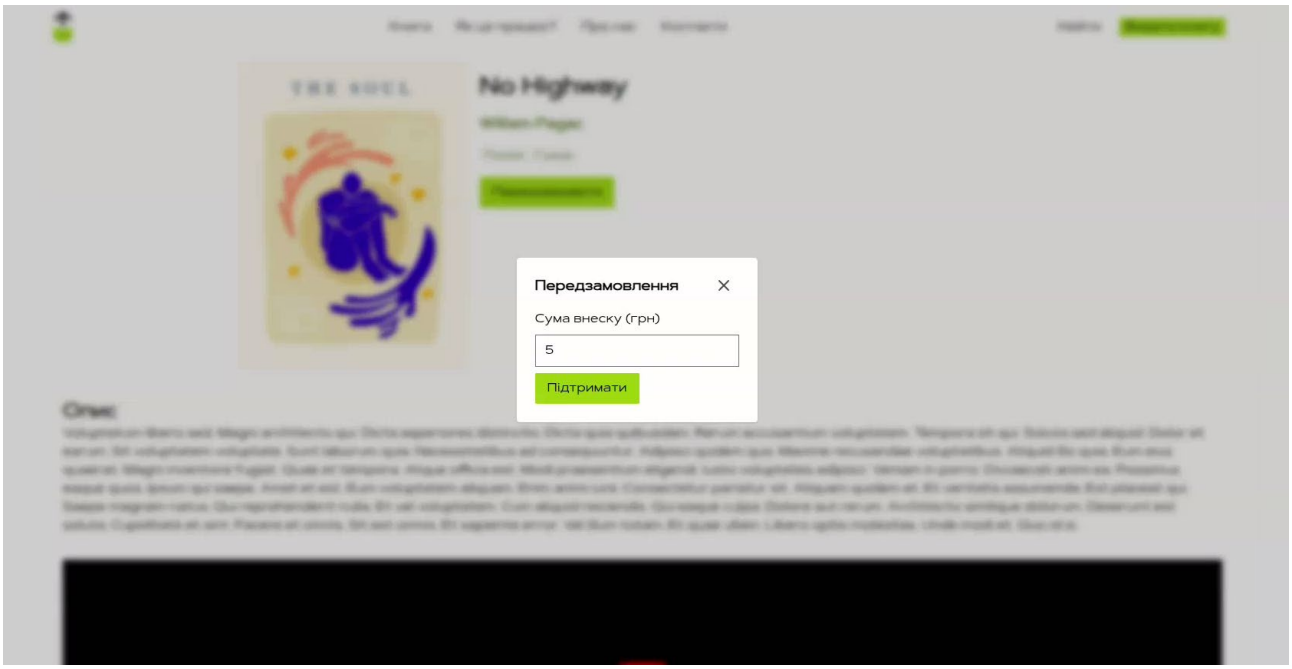

Рисунок В.9 – Модальне вікно передзамовлення

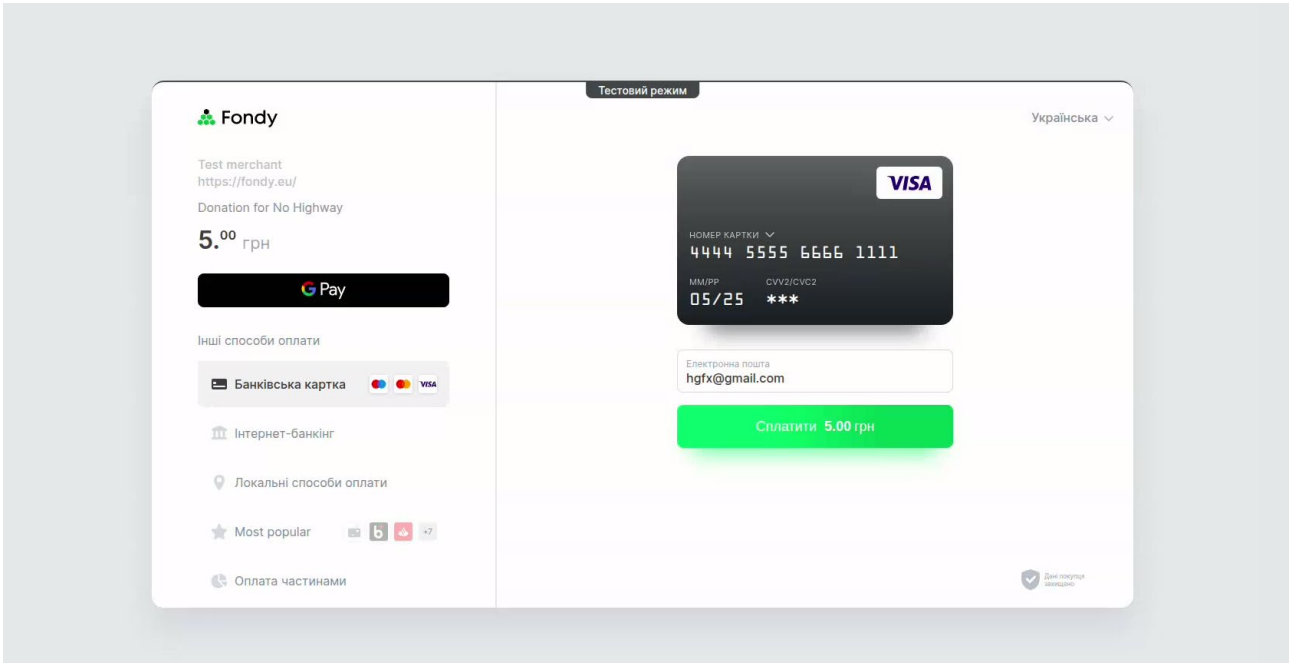

Рисунок В.10. – Спеціальна сторінка для оплати через Fondy

Також на тій же сторінці книжки є можливість дізнатися мову написання книги та переглянути відео про книгу від автора (рис. В.11), також окрема секція, що дозволяє додавати свої коментарі з приводу книжки (рис. В.12). Для автора книги доступна кнопка “редагувати” (рис. В.11).

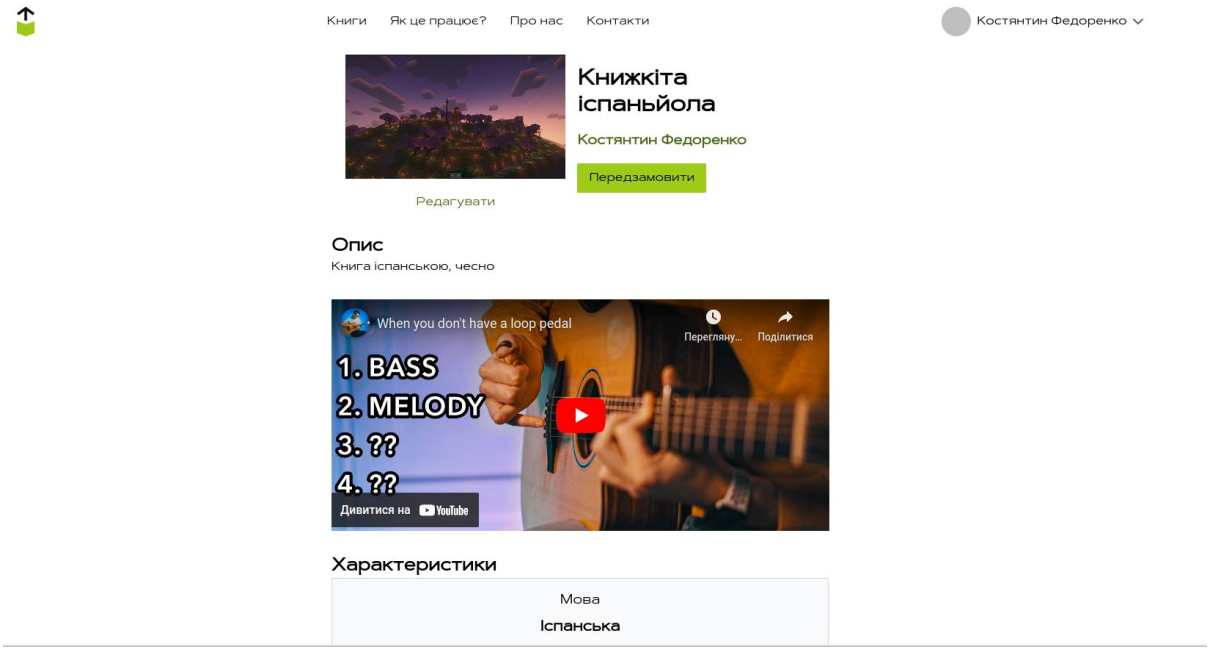

Рисунок В.11 – Фрагмент сторінки книги, де ми можемо бачити зокрема вказану нами раніше мову та відео з Youtube

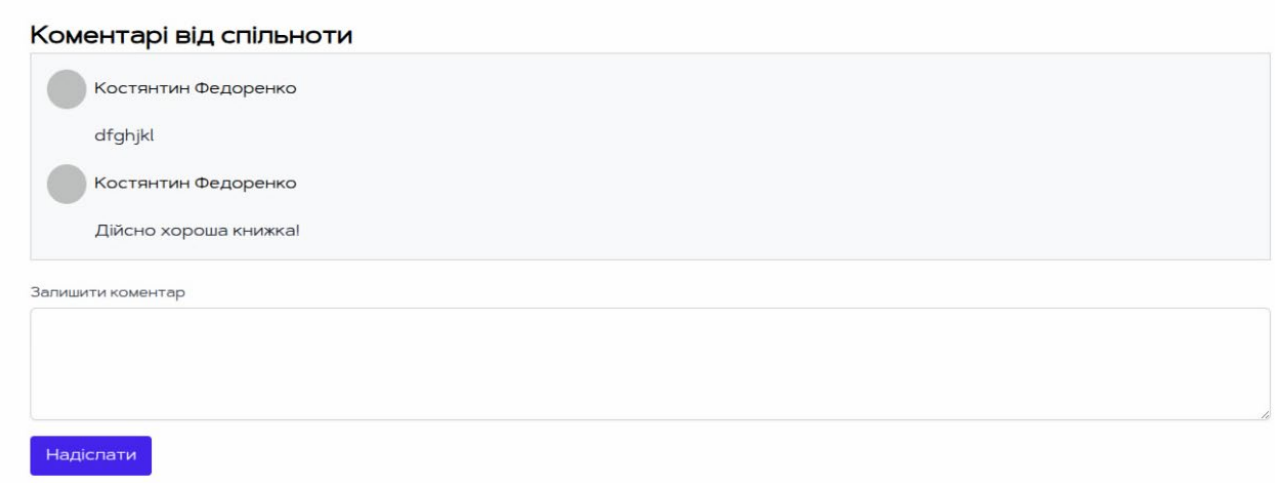

Рисунок В.12 – Секція сторінки книги з можливістю додавати коментарі

АНОТАЦІЯ

Федоренко К.О. Розробка та впровадження онлайн-платформи для краудфандингу книжкових проектів з функціями управління проєктами та підтримки авторів.

Спеціальність 112 – Комп'ютерні науки – Волинський національний університет імені Лесі Українки – Луцьк, 2025.

Ця робота зосереджена на розробці онлайн платформи краудфандингу для книжкових проєктів. Основною метою є створення функціонального вебзастосунку, що дозволяє авторам публікувати власні книжкові ідеї, збирати кошти на їх реалізацію через передзамовлення, а також забезпечує зручний інтерфейс для потенційних читачів, які можуть підтримати вподобані проєкти. В рамках дослідження проаналізовано аналогічні краудфандингові рішення, зокрема у сфері літератури, з метою виявлення типових функціональних моделей, недоліків і потреб користувачів. Особливу увагу приділено побудові ефективного технічного завдання, що охоплює ключові компоненти: каталог проєктів, реєстрацію й аутентифікацію користувачів, багатосторінкову форму створення проєкту, сторінку книги з функціоналом донату, інтеграцію з платіжною системою Fondy, а також адміністративні інструменти для тестування.

Платформа реалізована з використанням фреймворку Ruby on Rails та реляційної бази даних PostgreSQL. У роботі докладно описано структуру проєкту, особливості маршрутизації, обробки запитів та валідації даних. Фронтенд розроблено з урахуванням вимог до зручності інтерфейсу: реалізовано динамічне завантаження каталогу, підтримку Drag and Drop API для завантаження медіафайлів, адаптивний дизайн.

Серверна частина платформи була протестована у середовищі, максимально наближеному до умов продакшн-експлуатації. Розгортання відбувалося на університетському тестовому сервері з операційною системою Ubuntu 22.04.2

LTS. Ручне тестування дозволило оцінити не лише окремі функції, а й взаємодію між модулями в реальному мережевому середовищі.

У завершальній частині роботи наведено приклади ключових сценаріїв використання, а також запропоновано напрями подальшого розвитку проєкту, зокрема розширення механізмів пошуку та фільтрації, підтримку багатомовності, реалізацію електронної бібліотеки для збереження готових видань.

Ключові слова: вебзастосунок, краудфандинг, Ruby on Rails, PostgreSQL, Fondy.