Міністерство культури та інформаційної політики України
Харківська державна академія культури

Факультет соціальних комунікацій і музейно-туристичної діяльності
Кафедра інформаційної, бібліотечної та архівної справи

«**Автоматизація процесів індивідуального бібліографічного інформування користувачів бібліотек»**

Кваліфікаційна робота
здобувача 2 курсу
другого (магістерського) рівня вищої освіти
денної форми навчання
спеціальність: 029 «Інформаційна, бібліотечна та
архівна справа»
Гончарова Д. О.

Науковий керівник: д. п. н, проф. Соляник А.А.

Рецензент: к.т.н., доцент кафедри програмної
інженерії ХНУРЕ Мар'їн С.О.

Харків – 2023

ЗМІСТ

ВСТУП

Сучасний світ неможливо уявити без автоматизованих технологій обробки інформації, збільшення обсягів якої ускладнює процеси її збереження та структурування. Цифрова інформація зберігається в реляційних та нереляційних базах і сховищах даних, обсяг яких нині вимірюється терабайтами. Бібліотеки як базові комунікаційні посередники в системі «інформація — користувач» з метою оптимізації свої діяльності використовують автоматизовані бібліотечні інформаційні системи (АБІС), підтримують власні сайти, портали та акаунти в соціальних мережах [7]. За допомогою цих ресурсів вони суттєво розширюють мережу віддалених користувачів бібліотек та покращують якість їх інформаційного обслуговування [4].

Індивідуальне інформування користувачів – це тип бібліографічного інформування, яке спрямовано не персоналізацію обслуговування та оперативне задоволення індивідуальних запитів та інформаційних потреб бібліотечних абонентів [38]. Індивідуальне інформування допомагає користувачам постійно отримувати нову релевантну інформацію, якісною характеристикою якої є оперативність та комфортність її доставки [34]. Для допомоги користувачам можуть бути необхідними персональні консультації [10], підтримка з ними постійного зворотного зв’язку, моніторинг та оцінка оперативності та якості інформування [49].

Для підтримки зворотного зв’язку з користувачем, оцінки ступеня їх задоволеності якістю інформування працівниками багатьох наукових і публічних бібліотек створені блоги і аканти в соціальних мережах, зокрема: блог відділу абонемента Чернівецької ОУНБ «Книжковий простір», який інформує про кращі книги та нові надходження книг у бібліотеку; блог Миколаївської ОУНБ ім.О. Гмирьова підтримує «Книжковий континент». Ці блоги — своєрідний майданчик, де відвідувачі можуть вільно ділитися враженнями від прочитаного, отримувати допомогу або пораду у пошуку книг

та просто залишати свої відгуки про роботу бібліотеки. Популярними у користувачів є блоги «Порадник читача» (Рівненської ОУНБ), «Бібліотека без бар'єрів» (Херсонської ОУНБ імені О. Гончара), «Бібліотечна веселка» та «Читальники» (Волинської ОУНБ ім. О. Пчілки) та ін. [3].

Постійне збільшення обсягів інформації, яку бібліотеки мають надавати своїм користувачам, ускладнює завдання їх своєчасного та персоніфікованого інформування. Поки що не існує доступних за вартістю більшості бібліотекам технологій автоматизації цих процесів. Ще 20 років тому інформування відбувалась переважно вручну [47], але зараз цей процес покращується на основі застосування можливостей сучасних інформаційних технологій. Формат надання послуги має залежати від рівня розвитку технологій та інформаційних потреб користувачів [1], тому одним з комфортних форм є боти в месенджерах.

Дослідженням існуючих форм надання бібліотеками інформаційних послуг займалися українські бібліотекознавці І. Давидова [11-14], К. Бережна [2-3], К. Лобузіна [25-26], О. Мар'їна [28-32]. Особливості застосування інноваційних технологій в процесах створення та надання бібліотеками інформаційних послуг відображається в роботах М. Івашиної, Н. Каліберди [17-19], К. В. Лобузіної [25-26], О. Ю. Мар'їної [28-32], Г. М. Швецової-Водки та інших. У працях зарубіжних науковців висвітлено принципи автоматизації інформаційних систем [53-55], які були використано автором у розробці додатку до АБІС. Декілька публікацій, присвячених вирішенню проблем інтеграції сторонніх програмних застосунків з існуючими автоматизованими бібліотечними інформаційними системами в Україні, належать автору цієї магістерської роботи [6-8].

Мета дослідження – розроблення теоретико-методичних та організаційно-технологічних засад автоматизації процесів індивідуального інформаційного обслуговування користувачів бібліотек.

Завдання дослідження:

- визначити стан розробленості предметної області;

- охарактеризувати теоретико-методичні та організаційно-технологічні засади автоматизації індивідуального бібліографічного інформування користувачів бібліотек;
- з’ясувати сутність, складові та критерії оцінки системи автоматизації індивідуального інформаційного обслуговування користувачів бібліотек;
- визначити напрями та інструменти автоматизації процесів індивідуального бібліографічного інформування користувачів бібліотек.

Об'єкт дослідження: Інформаційні системи для бібліотечно-інформаційного обслуговування користувачів.

Предмет дослідження: особливості застосування автоматизованих інформаційних систем для індивідуального бібліографічного інформування користувачів бібліотек.

Методи дослідження обумовлені об’єктом і предметом кваліфікаційної роботи. Для досягнення мети та розв’язання визначених завдань застосовано комплекс взаємодоповнюючих методів дослідження: аналізу та синтезу, абстрагування, моделювання, системного підходу, узагальнення, прогнозування.

Структурно дипломна робота складається зі вступу, трьох основних розділів, висновків, списку використаних джерел.

РОЗДІЛ 1.
ТЕОРЕТИКО-МЕТОДИЧНІ ЗАСАДИ АВТОМАТИЗАЦІЇ ІНФОРМАЦІЙНОГО ОБСЛУГОВУВАННЯ КОРИСТУВАЧІВ БІБЛІОТЕК

1.1 Аналіз предметної області

Сучасна бібліотека допомагає зберегти національні цінності, культурні та наукові знання, формує простір для всебічного розвитку людини та популяризує читання [40]. Бібліотечна галузь не втрачає актуальності та затребуваності користувачів, тому її важливо підтримувати та розвивати.

Бібліотечна мережа України нараховує майже 40 тисяч бібліотек різного підпорядкування та форми власності [41]. Потужна мережа бібліотек потребує відповідного програмного забезпечення для автоматизації рутинних технологічних процесів та операцій.

Кожна бібліотека намагається оперативно інформувати своїх користувачів про нові надходження, інновації щодо надання базових інформаційних продуктів та послуг, але не всі поки що організували цю роботу із застосуванням автоматизованих технологій. В сучасній Україні задача інформування вирішується за допомогою RSS-стрічок, публікації матеріалу на сайті та в соціальних мережах, розсилок по електронній пошті. Практично всі існуючи в бібліотеках системи інформування читачів вже не відповідають сучасним стандартам розробки програмного забезпечення, а публічні рішення не використовуються через правила безпеки та конфігурації АБІС, що не мають виходу до глобальної мережі [6].

Сучасні користувачі потребують простий та зрозумілий у використанні застосунок, щоб отримувати та налаштовувати персоналізовані списки нових надходжень та іншої бібліографічної інформації. Це підтверджує сервіс google trends [42], його дані, проілюстровані на рисунку 1.1, свідчать, що тенденція читання та пошуку користувачами нових книг не зменшується в світі за останні

5 років. Люди цікавляться новими книгами та намагаються знайти їх через пошукові системи.

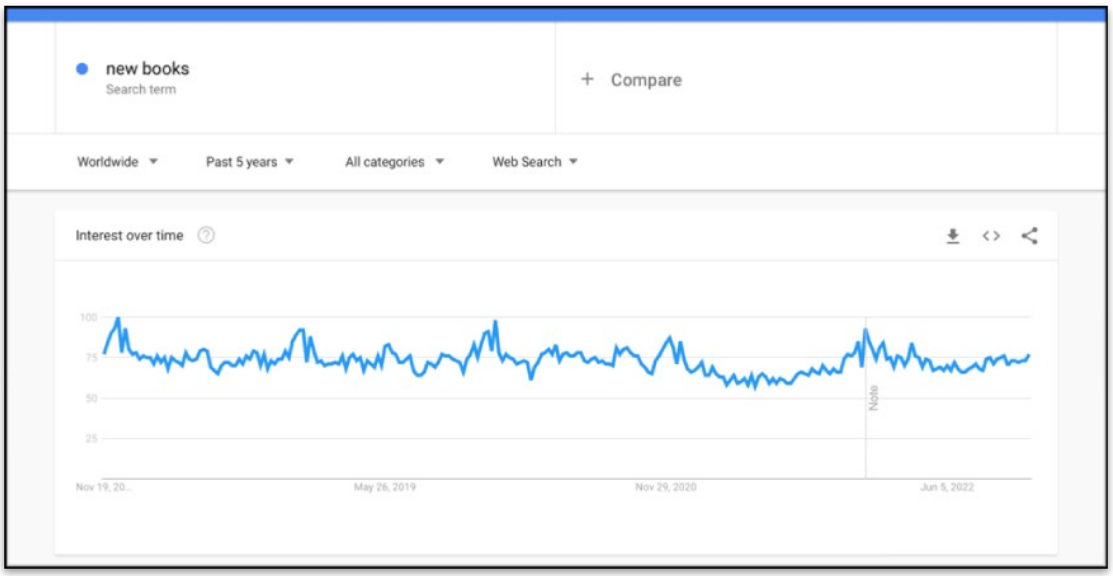


Рис. 1.1 – Динаміка пошуку нових книжкових видань

1.2 Виявлення проблем та огляд існуючих рішень

Для того, щоб створити оптимальну інформаційну систему, треба визначити проблеми в існуючих функціональних рішеннях. Для цього розглянемо рішення, які використовуються в бібліотеках зараз через характеристики функціоналу додатків Amlib, Aleph, Polaris, Koha, а також оцінимо існуючи методи сповіщення користувачів у базових бібліотечних сервісах.

1.2.1 Додаток Koha

Koha [57] – це одна з найпопулярніших автоматизованих бібліотечних інформаційних систем (АБІС) [48], це перша АБІС у світі з відкритим кодом, який розробляють інженери з різних країн. Цей застосунок має широкий функціонал для автоматизації рутинних бібліотечних процесів та операцій, проте ми розглянемо тільки рішення в сфері автоматизованого інформування читачів про нові надходження. Приклад інтерфейсу Koha ілюструє рис. 1.2

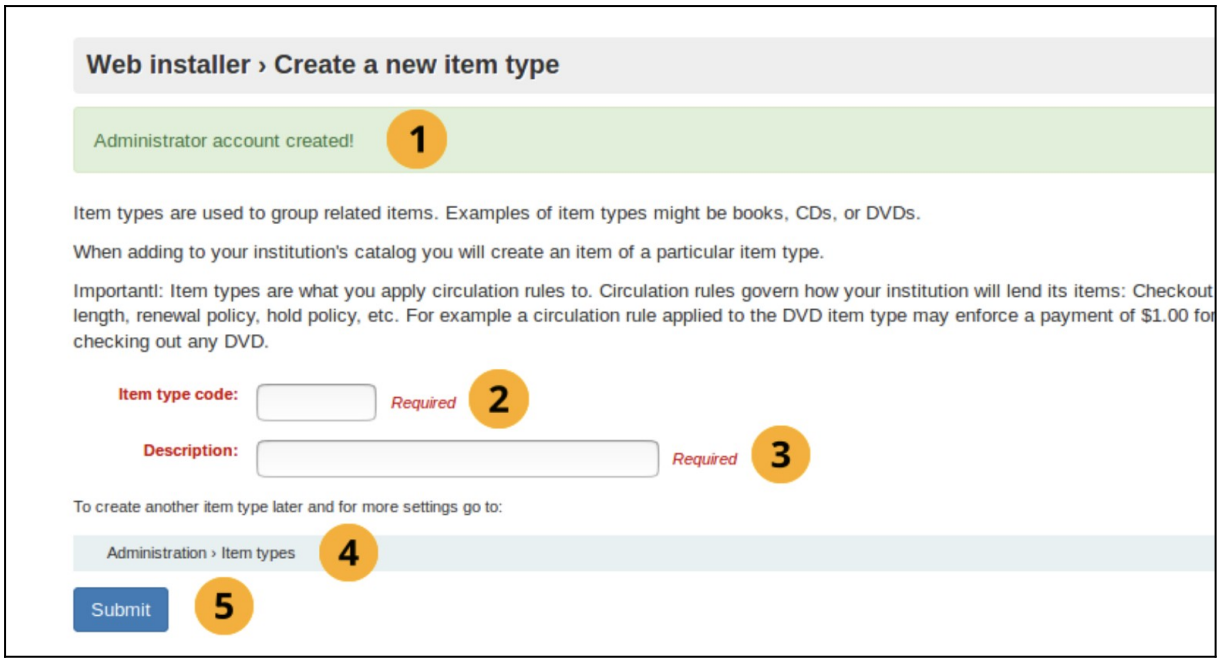


Рис. 1.2 – Інтерфейс Koha

Koha реалізована за допомогою клієнт-серверної архітектури та передбачає, що користувач зможе отримати дані про всі свої бібліотечні об'єкти на сервері з будь-якого пристрою з глобальної мережі. Ризики, які пов'язані зі зберіганням та обробкою персональних даних, призвели до того, що в сучасних бібліотеках клієнти та сервери, на яких зберігаються дані користувачів, знаходяться в одній захищеній локальній мережі, щоб не порушувати закон України [36-37]. Щоб системи мали право передавати дані в сторонні додатки, вони мають бути ліцензованими і користувач має погодитися на передачу своїх даних. Переважна більшість сучасних програмних застосунків розгортаються за допомогою хмарних середовищ, таких як GCP або AWS [60], та не мають змоги повноцінно і ефективно функціонувати в умовах відсутності доступу до глобальної мережі. А це означає, що інформація про нові надходження для відправлення у соцмережах повинна вручну експортуватися з АБІС. Koha має додатковий функціонал експортування та імпортування бібліографічних описів документів системи в різні формати файлів, такі як CSV та XML.

Недоліки АБІС Koha:

– Koha має функцію розсилки повідомлень на електронну пошту та при встановленні додаткових бібліотек може працювати з RSS-стрічкою, проте не має функцію інформування читачів за допомогою соціальних мереж;

– Koha зберігає особисту інформацію користувачів в одному місці, не має можливості інтеграції зі сторонніми застосунками;

– Koha працює тільки в локальній мережі організації;

– Koha не повністю підтримує українську мову;

– Koha не надає можливості редагувати персоналізовані вподобання користувачів;

– АБІС Коха буда створена для операційної системи Linux, нові версії можуть бути встановленими у Windows, але інсталяція під Windows вимагає встановлення та налаштування декількох бібліотек з кодом додатково, підтримуються ці бібліотеки окремо від Koha, тому виникає проблема сумісності версій.

1.2.2 Додаток Amlib

Amlib [51] – це АБІС, що має схожий с Koha функціонал та недоліки. Amlib може використовуватися як в невеликій бібліотеці з одним працівником, так і масштабуватися до системи для мільйонів користувачів та сотень адміністраторів. Amlib має готові версії програми для коледжів, університетів, юридичних установ, установ охорони здоров’я, корпорацій та використовується навіть в тюрмах Великобританії. Приклад інтерфейсу можна побачити на рисунку 1.3.

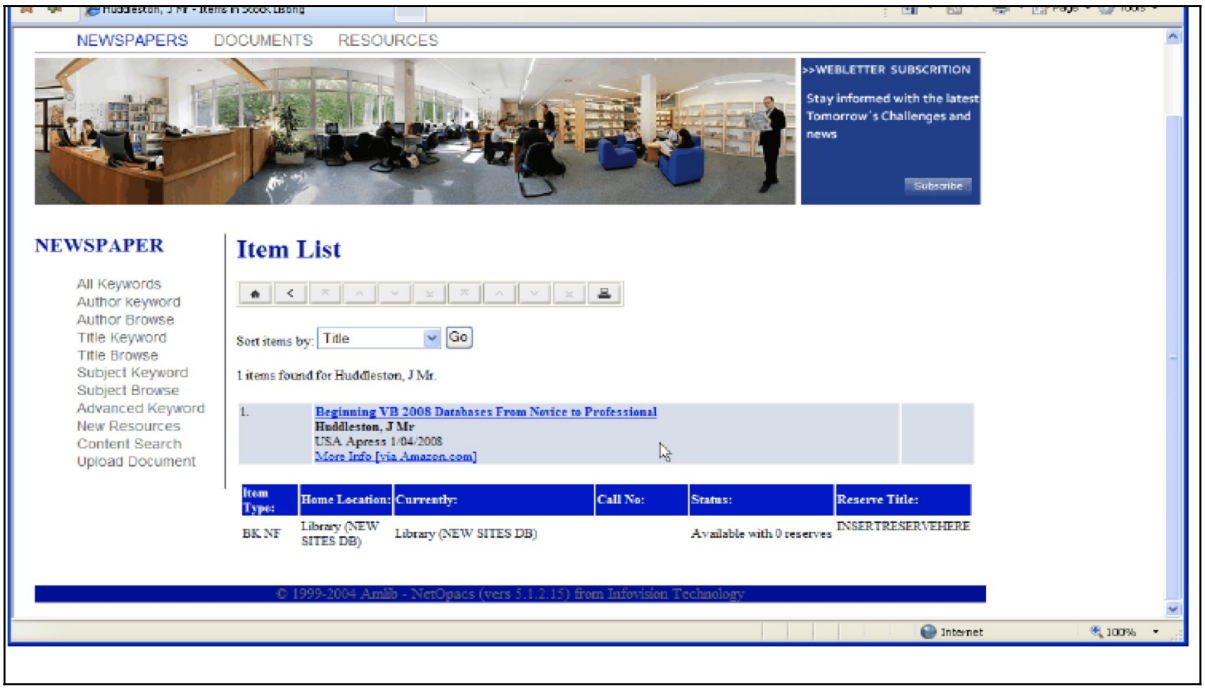


Рис. 1.3 – Інтерфейс Amlib

Недоліки АБІС Amlib:

– Amlib не має функції інформування читачів за допомогою соціальних мереж;

– Amlib працює лише у локальній мережі;

– Amlib не має української локалізації;

– Amlib повідомляє користувачів по пошті лише про те, що документ змінив статус, наприклад, прийшов час повертати книги та статус книги змінено на «прострочена», але не інформує читачів про нові надходження;

– Amlib потребує реєстрації та авторизації усіх акторів системи, та не захищає дані про користувачів додатково;

– Користувачі відмічають, що Amlib має складний для розуміння інтерфейс;

– Amlib не надає можливості редагувати персоналізовані вподобання користувачів.

1.2.3 Додаток Book-RSS

Book-RSS [52] – це програмний додаток, що надає можливість підписатися на RSS-стрічку Google Books. Результат цього додатку на сайті застосунку – це чотири посилання на стрічки з книгами, які можна побачити на рис. 1.4. RSS-стрічка – зручний інструмент для користувача. Її можна інтегрувати в браузери та месенджери, такі як Telegram або Viber [44]. Важливий плюс цього сервісу це швидкість та різноманітні способи отримання даних, відсутність додаткової реєстрації та авторизації.

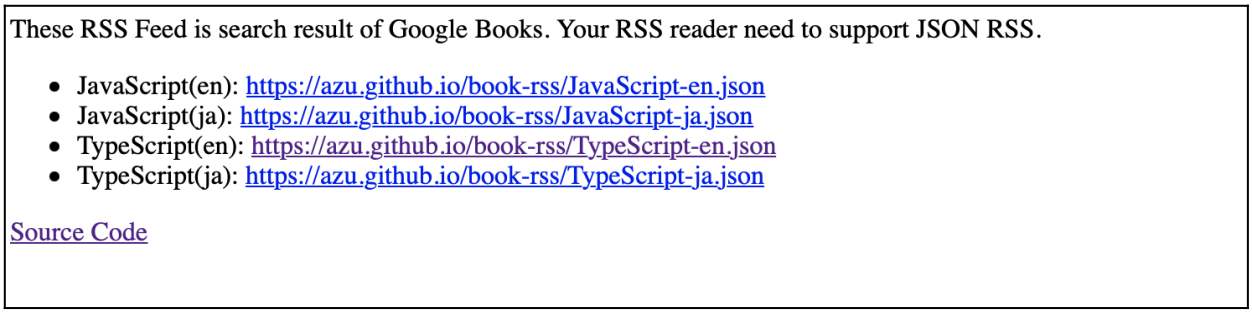


Рис. 1.4 – Інтерфейс Book-RSS

Недоліки:

– складність в налаштуванні, треба мати знання про використання RSS-стрічок, вміти користуватися, наприклад, telegram-ботами, щоб створити інтеграцію з telegram;

– неможливість персоналізації;

– неможливість отримувати повідомлення про нові надходження на електронну пошту.

З огляду на недоліки вищезазначених програмних додатків, цікаві сервіси запроваджено на сайті бібліотеки НФУ [35], який має спеціальну сторінку для інформування читачів про нові надходження. Більшість бібліотек України зараз мають власні сайти або сторінки в соціальних мережах, де публікують аналогічну інформацію. Інтерфейс сайту наукової бібліотеки можна побачити на рис. 1.5.

Рис. 1.5 – Інтерфейс сайту наукової бібліотеки НФУ

Недоліки:

– Відсутність автоматизації змушує постійно редагувати контент сайту;

– Відсутність будь-яких сповіщень про нові надходження змушує користувачів постійно перевіряти сайт бібліотеки, щоб отримати оновлену інформацію;

– Не має можливості сортування та фільтрування інформації про нові надходження по жанрах або даті;

– Сайт має багато помилок, не завжди відкриваються сторінки;

– Відсутня персоналізація даних.

На жаль, це типові недоліки для багатьох бібліотечних сайтів, які є каналом бібліографічного інформування користувачів.

1.3 Характеристика складових системи інформаційного обслуговування користувачів бібліотек

Розробка програмного застосунку потребує визначення усіх вимог до системи та її структурних елементів. Система індивідуального інформування користувачів поділяється на три основні підсистеми: клієнтська, серверна та підсистема оновлення додатку з АБІС.

Сформулюємо загальні вимоги до програмної системи:

- інтуїтивно зрозумілий інтерфейс;
- забезпечення безпеки персональних даних;
- обробка усіх можливих помилок впродовж роботи застосунку;
- стабільність;
- масштабованість;
- швидкість реагування на запити;
- коректність обробки інформації.

1.3.1 Вимоги до клієнтської підсистеми

Клієнтська частина відповідає за увесь графічний інтерфейс додатку, з яким працює користувач. Клієнтська частина не оброблює введені дані, але валідує їх та відправляє на серверну частину.

Сформулюємо функціональні вимоги до клієнтської частини застосунку:

- Окрема сторінка для налаштування вподобань тема та жанрів нових надходжень, якими цікавиться користувач;
- Окрема сторінка для підписки нового користувача;
- Окрема форма для додання соціальних мереж;
- Окрема сторінка — панель адміністратора;
- Реалізація повідомлень по електронній пошті.

Сформулюємо нефункціональні вимоги до клієнтської частини застосунку:

- Виконання будь-якої дії в додатку повинна бути інтуїтивно зрозумілим новому користувачу;
- Всі сторінки мають автоматично адаптуватися під розмір екрана пристрою користувача;
- Клієнтська частина має правильно валідувати та відображати інформацію.

1.3.2 Вимоги до серверної підсистеми

Серверна підсистема – це RESTful [55] API, що вміє отримувати нові книги з АБІС та обробляти запити клієнтської підсистеми, повертаючи коректну відповідь користувачеві та керуючі бізнес-логікою програмного застосунку.

Сформулюємо функціональні вимоги до серверної частини застосунку:

– Реалізація взаємодії зі СКБД;
– Реалізація взаємодії між АБІС та системою;
– Реалізація API для отримання надходжень літератури;
– Запити інформації (наприклад, до СКБД) повинні бути оптимізованими;
– Реалізація та підтримка токенної авторизації до сервісу з використанням електронної пошти;
– Реалізація взаємодії з соціальними мережами;
– Реалізація зберігання та керування статичними файлами;
– Серверна частина повинна правильно валідувати та обробляти інформацію.
– Реалізація генерації повідомлень користувачам про нові надходження;
– Реалізація панелі адміністратора застосунку.

Сформулюємо нефункціональні вимоги до серверної частини застосунку:

– Запити до системи мають оброблятися в обґрунтований час, якнайшвидше;
– Усі дані мають бути захищеними, а особливі приватні дані, такі як пароль – захешованими;

– Система має зберігати усі дані, та не дозволяти часткових транзакцій.

1.3.3 Вимоги до підсистеми оновлення додатку з АБІС

Підсистема оновлення додатку це додаткова джоба, що вміє структурувати дані, експортовані з АБІС по темі та відправляти їх на серверну частину, де створюються оповіщення для користувачів за допомогою API [5]. Ця підсистема не має даних про користувачів, не зберігає їх, та обробляє лише інформацію про нові надходження до бібліотеки.

Сформулюємо функціональні вимоги до підсистеми оновлення додатку з АБІС:

– Реалізація авторизації та відправлення запитів до серверної частини;

– Реалізація екпорту даних з АБІС.

– Визначення тематики книги та структурування книг за темою.

Серед нефункціональних вимог до підсистеми оновлення додатку з АБІС:

– Підсистема повинна працювати без збоїв, не допускати часткових оновлень чи дублювання даних;

– Підсистема має бути легко масштабованою на різні АБІС та різні розміри даних.

Аналіз існуючих інформаційних систем показав, що не існує ідеальної системи для сповіщення користувачів бібліотек про нові надходження, хоча таких систем багато і є попит на таку інформацію.

Недоліки існуючих систем можна поділити на дві групи: проблеми використання систем бібліотеками та проблеми використання систем користувачами.

Серед недоліків існуючих програмних додатків для користувачів, що погіршують досвід використання, можна виділити наступні:

– Немає повідомлень про нові надходження взагалі, треба постійно перевіряти додаток, щоб дізнатися інформацію;

– Інтерфейс додатку незручний та незрозумілий, складно навчитися їм користуватися;

– Для того, щоб отримати інформацію про нові надходження, треба авторизуватися в системі;

– Немає можливості обирати тематику книг, які подобаються;

– Додаток працює тільки у локальний мережі, щоб забезпечити безпеку приватних даних;

– Різні бібліотеки мають різні системи, що працюють за різними правилами;

Серед недоліків існуючих програмних додатків для працівників бібліотек, що погіршують досвід використання, можна виділити наступні:

– Щоб додати інформацію треба змінювати код додатку, вивчати систему;

– Немає можливості перенести дані з АБІС, треба вводити всі дані вручну.

Встановлені недоліки існуючих систем доводять, що бібліотеки потребують нової системи, яка буде ефективною й комфортною для користувачів. Так, вже на етапі постановки задачі можна виділити основні вимоги до нової системи, задовольнивши які система буде достатньо конкурентоспроможною:

– Сервіс має легко інтегруватися з різними бібліотеками, інтерфейс має бути однотипним;

– Інформація про нові надходження має бути імпортована з АБІС та не вписуватись в сервіс вручну;

– Приватна інформація має бути захищена та не зберігатись публічно;

– Користувачу достатньо мати лише електронну пошту, щоб використовувати систему.

– Тематика книги має сортуватися та фільтруватися автоматично;

– Система не має працювати з АБІС безпосередньо, бо АБІС знаходяться в локальних мережах бібліотек, а сервіс має бути публічним.

– Система повинна легко масштабуватися на багато бібліотек різного розміру та кількості читачів.

Пропонується наступна модель додатку для бібліографічного інформування читачів про нові надходження, яка вирішує проблеми існуючих систем та відповідає сформульованим вимогам:

– Користувач заходить на сайт бібліотеки, де є посилання на сервіс для отримання нотифікацій про нові надходження;

– Користувач проходить за посиланням та вводить адресу електронної пошти у форму, після чого йому пропонується перевірити електронну пошту;

– Користувач отримує на електронну пошту посилання, яке доступне лише обмежений час (декілька годин);

– З електронної пошти користувач переходить на сторінку з чекбоксами, де можна обрати тематику нових надходжень, яка релевантна його інформаційним потребам;

– Додатково можна додати соціальні мережі та активувати ботів в месенджерах, якщо це передбачено сервісною політикою бібліотеки.

– Працівник бібліотеки може в будь-який час експортувати дані з АБІС в іншу систему за допомогою хмарного сервісу. Хмарні обчислення це один з найефективніших видів обчислень, бо дозволяють легко масштабувати будь-яку систему та коштують відносно недорого. Хмарні обчислення знижують вимоги до пристрою користувачів, мають багато налаштувань безпеки, допомагають забезпечити стабільність роботи додатку.

РОЗДІЛ 2.
ФУНКЦІОНАЛЬНІ ОСОБЛИВОСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ ІНДИВІДУАЛЬНОГО ОБСЛУГОВУВАННЯ КОРИСТУВАЧІВ БІБЛІОТЕК

У цьому розділі надається інформація про дизайн та проектування архітектури програмного додатку. Умовно архітектуру додатку можна поділити на чотири частини: клієнтський інтерфейс, база даних, серверний шар з бізнес-логікою та хмарний додаток. Шар з бізнес-логікою отримує запити з клієнтської частини та хмарного додатку, робить запити до бази даних та повертає інформацію на клієнтську частину. Клієнтська частина відображає інформацію користувачу, реалізує графічний інтерфейс. База даних зберігає всю інформацію про користувачів та нові надходження. Хмарний додаток отримує дані, що експортовані з АБІС, структурує їх та відправляє на серверну частину.

2.1 Сутність і структура інформаційної системи

Діаграма прецедентів [15], або Use Case діаграма, що представлена на рис. 2.1, це така UML діаграма, яка візуально відображає можливі та сценарії взаємодії між акторами (користувачами додатку) і прецедентами (діями в додатку). Описані сценарії взаємодії описують бізнес логіку усього застосунку: діаграма прецедентів допомагає змоделювати програмний застосунок, бо при розробці для кожного прецедента програмувався окремий клас [63].

На діаграмі прецедентів програмного застосунку, що проектується, можна виділити два актори: користувач-читач бібліотеки та адміністратор-працівник бібліотеки. Користувач може ввести пошту в застосунку, а потім налаштувати соціальні мережі або тематику книг, які будуть йому надходити,

та може отримувати повідомлення. Користувач не має доступу до модерування сайту. Адміністратор може додавати на змінювати списки нових надходжень.

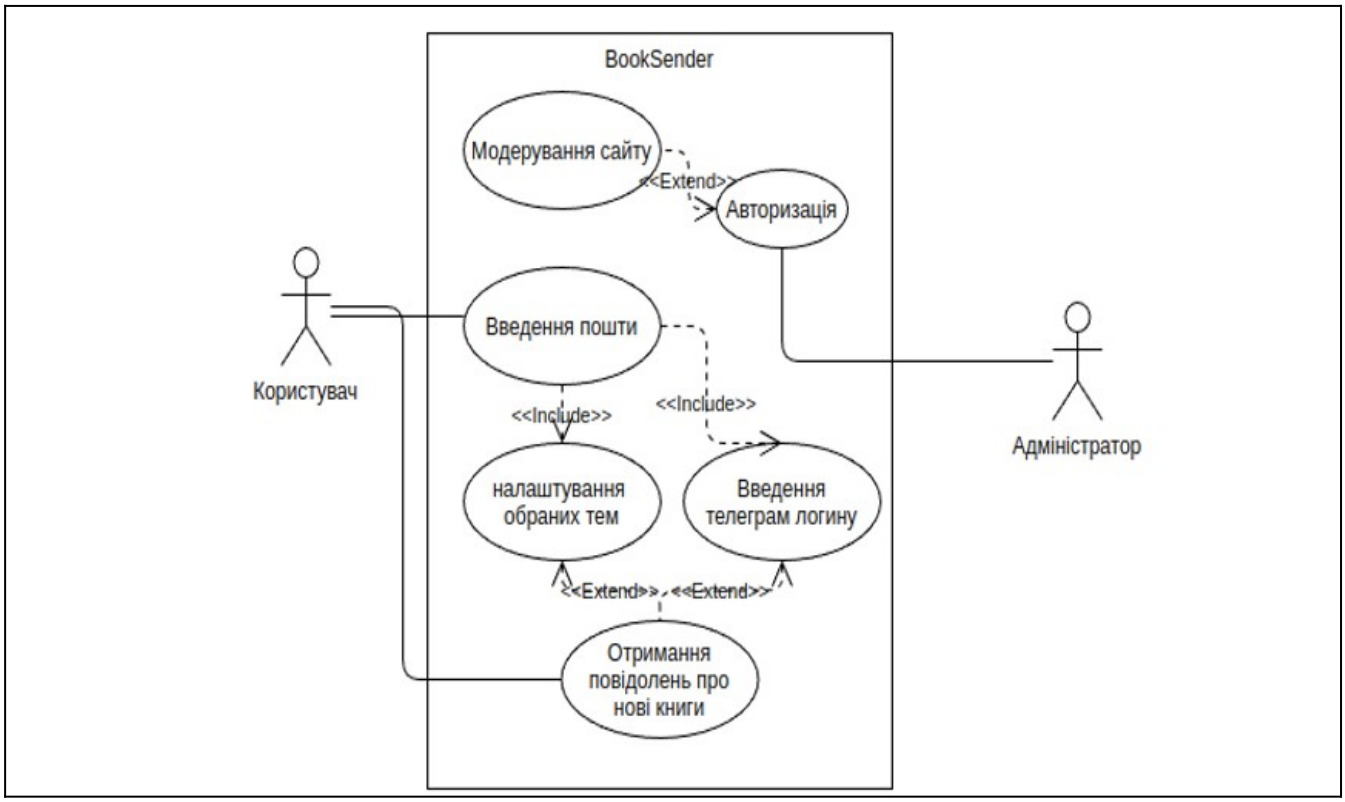


Рисунок 2.1 – Use Case діаграма

Моделювання об’єктів – дій в системі та зв’язків між ними для усіх акторів-типів користувачів відображає цілісну картину інформаційної системи [39]. Запропонована система допомагає змоделювати сутності в базі даних та архітектуру додатку, щоб розробити систему згідно до сформульованих вимог.

Аналіз предметної області та UML-проектування додатку допомагає виділити сутності для розробки бази даних. ER діаграма – це діаграма, яка дозволяє змоделювати сутності та їх поля, визначити зв’язки між сутностями [45]. Спроектована база даних, що була приведена до третьої нормальної форми, [43] щоб уникнути дублювання та інформаційної надмірності, приведена на рис. 2.2.

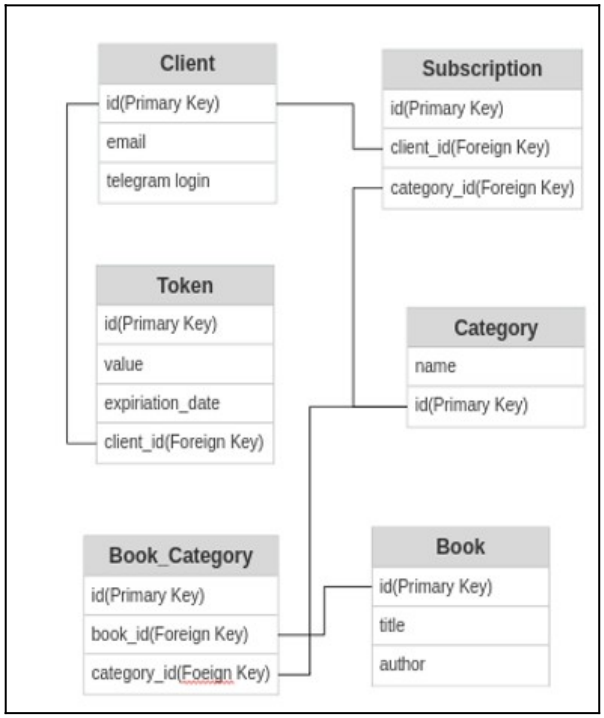


Рис. 2.2 – ER діаграма

Проектування ER-діаграми допомогло виділити наступні сутності, зв’язки та поля:

– Сутність Книга «Book»;

Сутність має наступні строкові атрибути: автор книги (author), назва книги (title) та автоматично згенерований ключ id;

– Сутність Категорія «Category»;

Сутність має наступні строкові атрибути: назва категорії (name) та автоматично згенерований ключ id;

– Сутність Категорія Книги «Book_Category»

Сутність має наступні атрибути-зв’язки з іншими сутностями: книга (book_id), категорія (category_id) та автоматично згенерований ключ id;

Сутність створена для підтримки зв’язку багато-до-багатьох між сутностями Книга та Категорія. Це реалізація правила, що книга може мати декілька категорій та одна категорія може належати багатьом книгам;

– Сутність Користувач «Client»;

Сутність має наступні строкові атрибути: електронна пошта (email), ідентифікатор соціальної мережі (telegram login) та автоматично згенерований ключ id;

– Сутність Токен «Token»;

Сутність має наступні атрибути: згенерована строка-токен (value), час, коли токен перестає працювати (expiration_date), зв’язок зі сутністю Користувач (client_id) та автоматично згенерований ключ id.

Сутність реалізує правило, що один користувач може мати багато токенів, деякі з яких можуть вже бути неактивними.

– Сутність Підписка «Subscription»;

Сутність має наступні атрибути-зв’язки з іншими сутностями: Користувач (client_id), Категорія (category_id) та автоматично згенерований ключ id;

– Сутність створена для підтримки зв’язку багато-до-багатьох між сутностями Користувач та Категорія. Це реалізація правила, що користувач може цікавитися декількома категоріями (тематикою) книг та одна категорія може цікавити багатьох користувачів одночасно;

Усі поля сутностей залежать винятково від одного ключа – автозгенерованого id [16]. У кожній сутності не існує поля, яке залежить від іншого неключового поля. З цього можна зробити висновок, що схема бази даних приведена до третьої нормальної форми та надана база даних зможе ефективно зберігати інформацію. Кодування даних буде у форматі UTF-8 [64]. База даних повинна повертати дані менше ніж за 2-3 секунди, тому було вирішено обрати базу даних PostgreSQL.

2.2. Архітектура бази даних та додатків

При проектуванні система була поділена на декілька частин, що властиво мікросервісній архітектурі [24]. Якісною відмінністю мікросервісної архітектури є те, що кожний сервіс працює у власному середовищі.

Середовища не пов’язані одне з одним та можуть розгортатися на різних хостах. Взаємодія між мікросервісами реалізується за допомогою REST HTTP запитів [53]. В системі плануються три сервіси: база даних на Heroku, хмарна джоба в AWS, клієнтська та серверна частини на третьому сервісі. Така архітектура забезпечує захист даних, зменшує зв’язність компонентів, що дозволяє уникати помилок.

З недоліків мікросервісної архітектури можна виділити наступні:

– Потреба додаткового тестування взаємодії між сервісами;
– Складність розгортання додатку;
– Час розробки збільшується.

Сервіс з клієнтською та серверною частиною перенавантажений функціоналом. Тому цей сервіс було вирішено поділити на декілька шарів, які можна побачити на рисунку: шар візуального відображення – це графічний інтерфейс клієнтської частини, шар бізнес-логіки – це класи з діаграми прецедентів, рівень даних – взаємодія з базою даних, отримання інформації з хмарного сервісу.

З недоліків багатошарової архітектури [42] можна виділити:

– Складність масштабування, треба буде змінювати багато коду, якщо в майбутньому буде створена нова клієнтська частина або змінений функціонал;

– Монолітна структура, яку складніше тестувати;

– Запит до системи переходить скрізь усі шари, навіть якщо це не потрібно (Рис. 2.3).

Проаналізувавши недоліки різних архітектур, було вирішено розробити додаток у вигляді двох мікросервісів та одного багатошарового моноліту. Такий вибір архітектури БД дозволяє зберігати дані на безпечному сервісі, який гарантує захист персональних даних, дозволяє найпотужнішій частині – обробці даних з АБІС бути у хмарному масштабованому застосунку, а це зменшує вимоги до обчислювальної потужності комп’ютера бібліотеки.

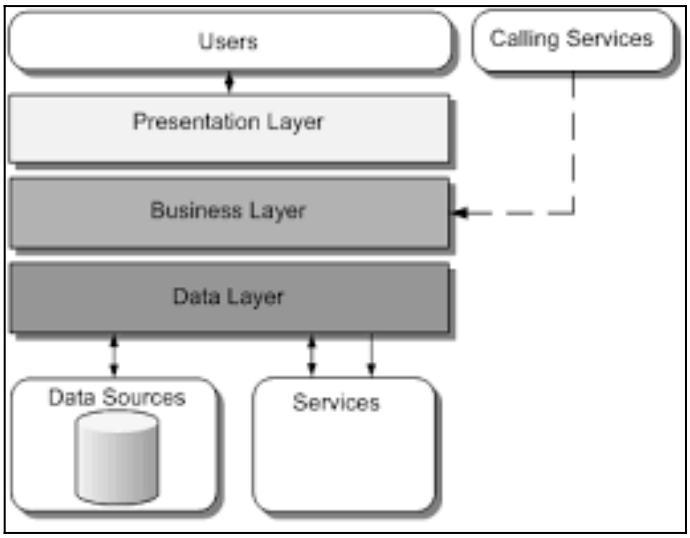


Рис. 2.3 – Багатошарова архітектура.

Таким чином, охарактеризована вище архітектура БД є найоптимальнішою та забезпечує ефективне використання даних як користувачем, так і персоналом бібліотеки.

РОЗДІЛ 3.
НАПРЯМИ ТА ІНСТРУМЕНТИ АВТОМАТИЗАЦІЇ ПРОЦЕСІВ ІНДИВІДУАЛЬНОГО БІБЛІОГРАФІЧНОГО ІНФОРМУВАННЯ КОРИСТУВАЧІВ БІБЛІОТЕК

3.1 UI/UX додатки

Важливою частиною програмного додатку автоматизованої інформаційної системи бібліографічного інформування є графічний інтерфейс користувача. Інтерфейс має бути зрозумілим, текст читабельним, його розмір — адаптуватися під екран мобільного пристрою користувача. Додаток має бути зручним у використанні та працювати швидко.

Розробка інтерфейсу починається з обрання кольорів для клієнтської частини. При цьому важливо зауважити, що панель «адміністратор» буде відрізнятися, щоб спростити розробку додатку. Обрані кольори для клієнтської частини можна побачити на рис. 3.1.

Рис. 3.1 – Кольорова палітра додатку.

Другим етапом розробки інтерфейсу є моделювання логотипу додатку, який створений за допомогою обраної кольорової палітри (рис. 3.2.). Логотип виконує навігаційну функцію та допомагає орієнтуватися користувачу в сервісних можливостях системи.

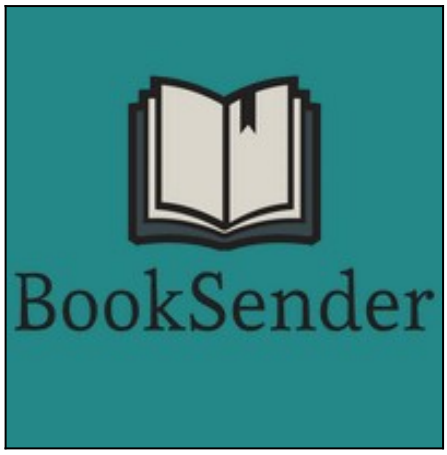


Рис. 3.2 – Логотип програмного додатку

Сторінка, з якої починається досвід використання додатку користувачем, це лендінгова сторінка сайту з інформацією про сервіс. Цю сторінку можна умовно поділити на дві частині: верхня частина на рис. 3.3 описує сервіс та пропонує перейти на наступну сторінку, щоб обрати категорії нових надходжень, про які може бути повідомлений користувач; нижня частина додатку, проілюстрована на рис. 3.4, – це блок з відгуками користувачів.

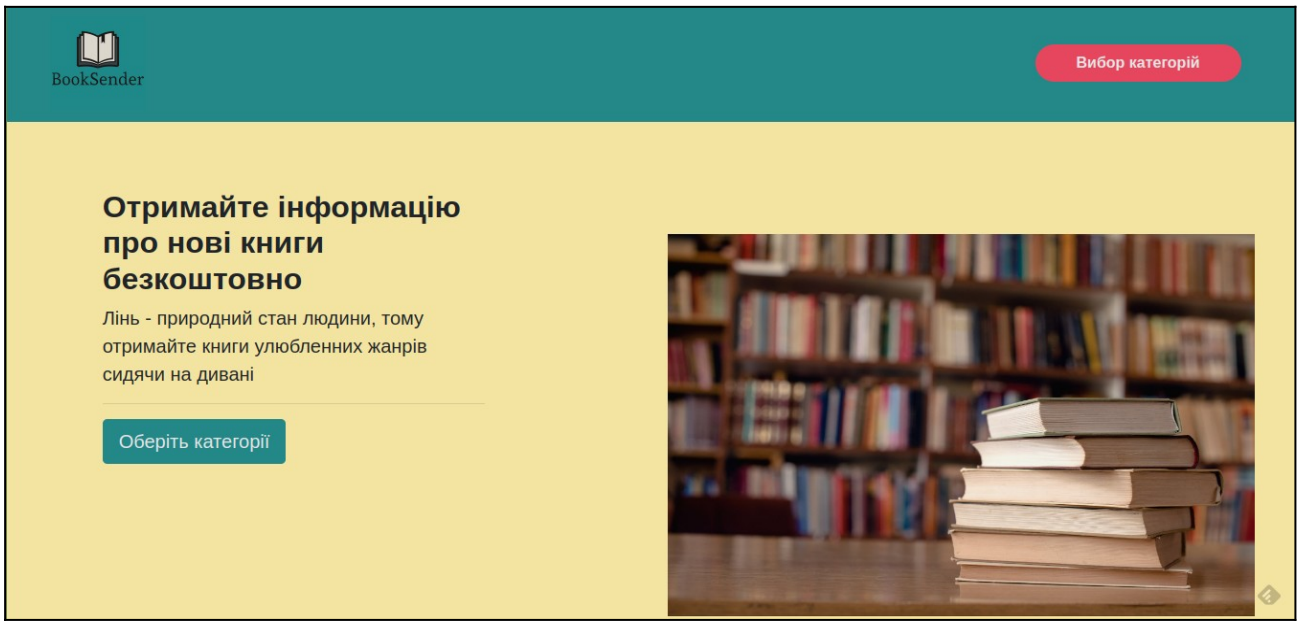


Рис. 3.3 – Лендінгова сторінка

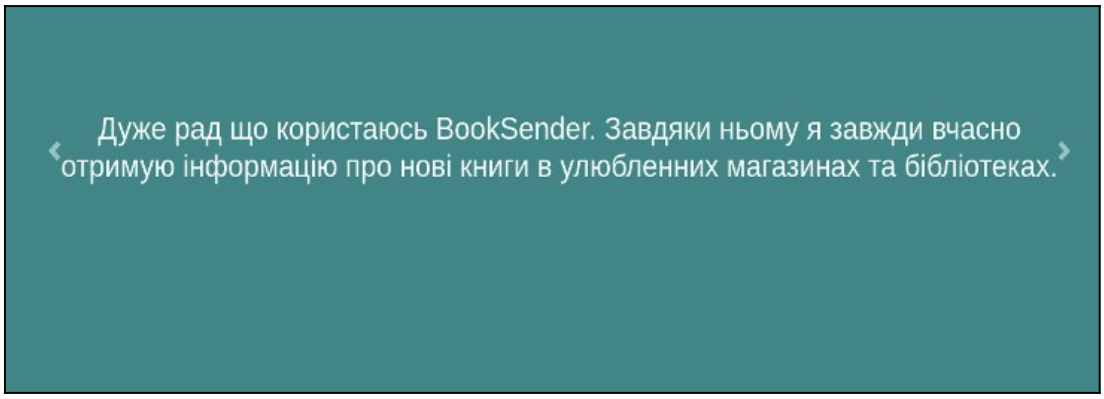


Рис. 3.4 – Карусель відгуків

З лендінгової сторінки доступна лише одна дія – перейти на сторінку, щоб почати процес обрання категорій нових надходжень. Ця сторінка надає можливість користувачеві ввести свій email, на який прийде повідомлення. Ця операція допомагає ідентифікувати користувача та захистити його персональні дані [21]. Інтерфейс сторінки проілюстровано на рис. 3.5.

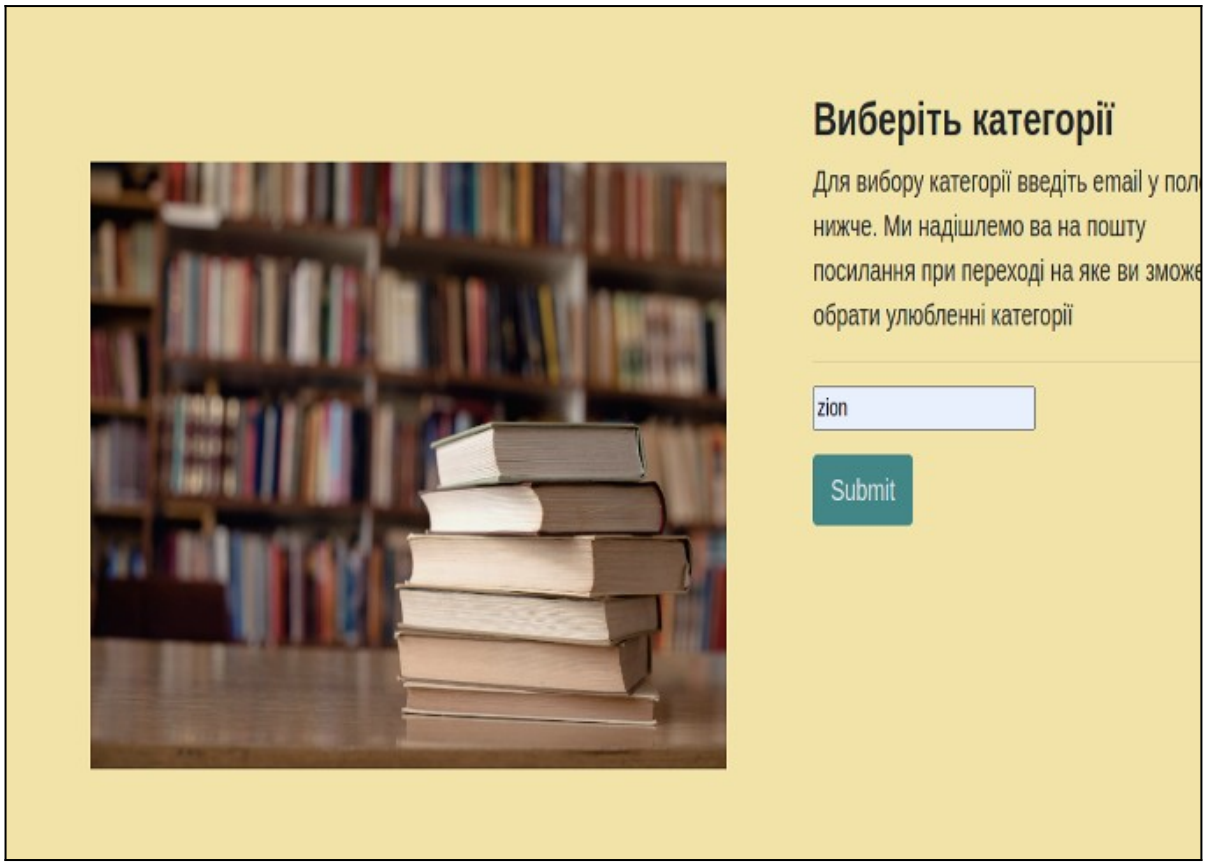


Рис. 3.5 – Інтерфейс сторінки ідентифікації користувача

Після введення власних даних користувач побачить підказку, що на email було відправлено посилання. Приклад повідомлення, яке надійде користувачеві, можна побачити на рис. 3.6.

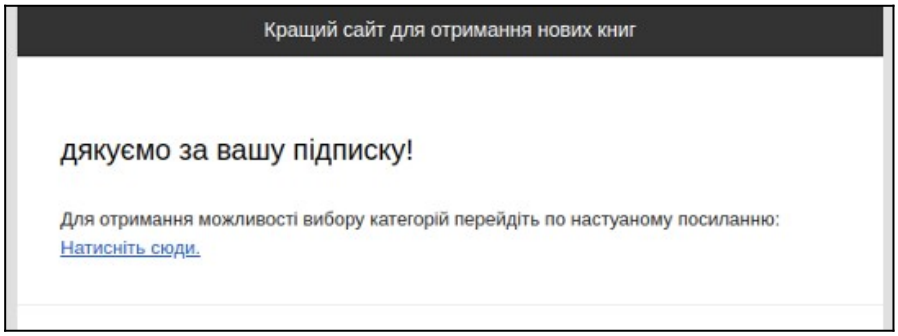


Рис. 3.6 – Приклад повідомлення на email користувача

Посилання діє обмежений час, проте дозволяє ідентифікувати користувача. Коли користувач додатку перейде по посиланню, то побачить сторінку, де він зможе обрати категорії-жанри (теми) нових надходжень, повідомлення про які він бажає систематично отримувати від бібліотеки. Приклад сторінки для редагування тематики інформаційних запитів користувача поданий на рис. 3.7.

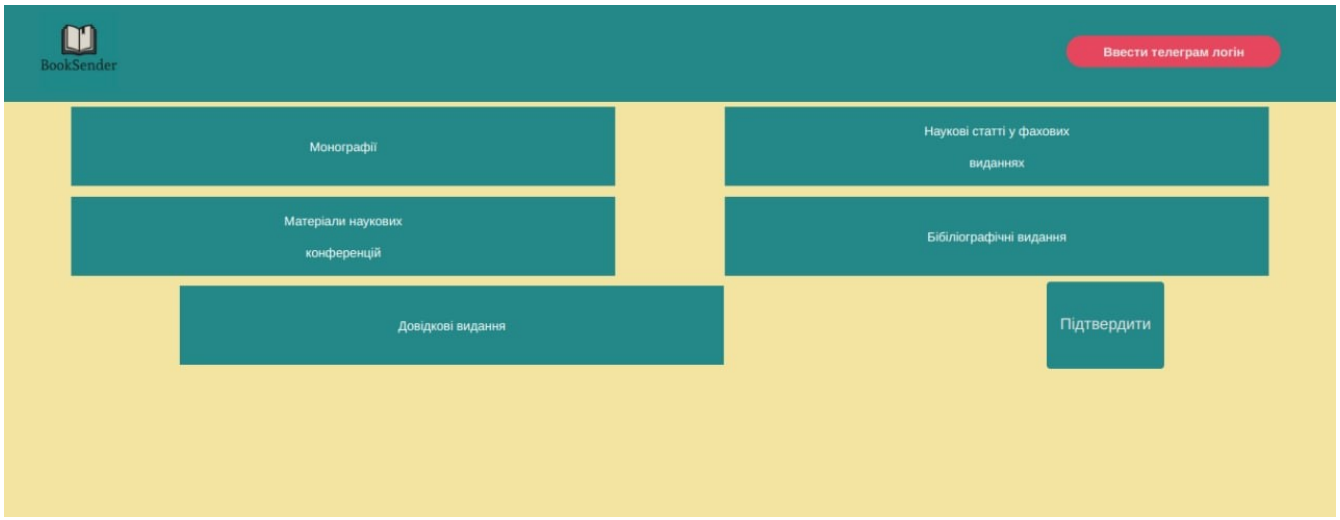
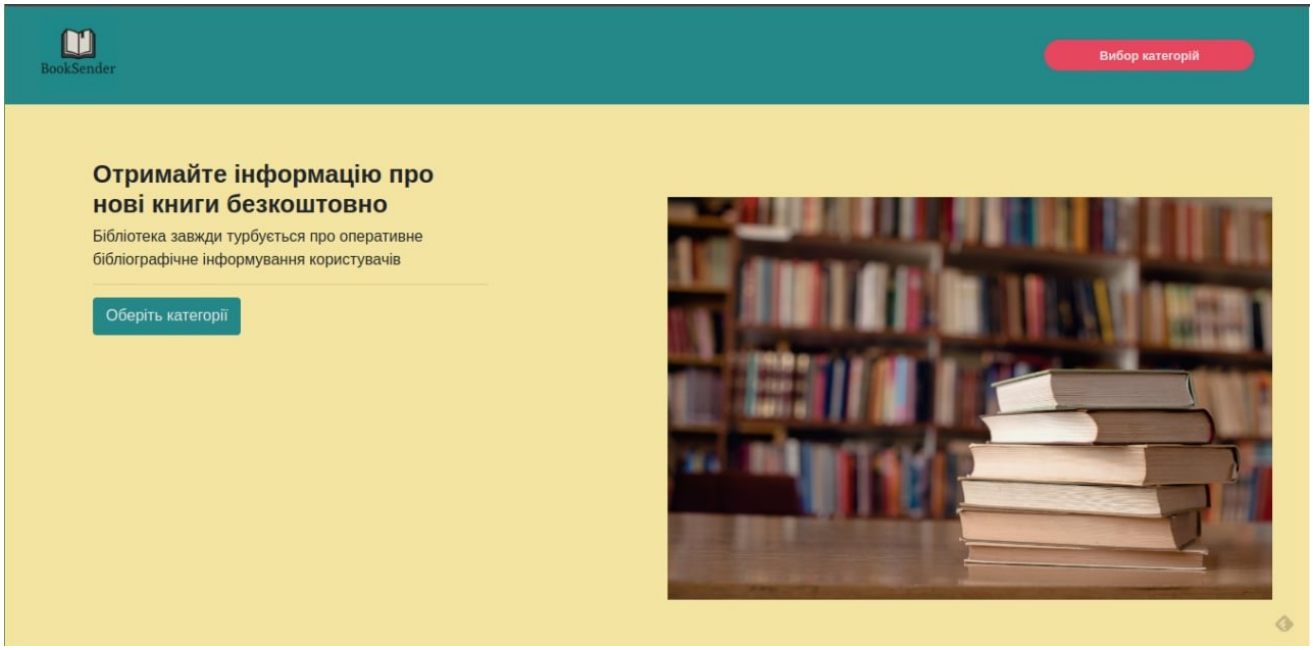


Рис. 3.7 – Інтерфейс обрання користувачем різних типів нових надходжень

Щоб обрати потрібну користувачеві категорію, треба просто натиснути на неї, щоб відмінити дію, треба натиснути ще раз. Після цього користувач буде повідомлений, що обрані категорії збережені додатком та перенаправлений на лендінгову сторінку. Приклад такого повідомлення проілюстровано на рис. 3.8.



Рис/ 3.8 – Інтерфейс підтвердження, що обрані категорії інформаційних повідомлень збережені

Коли в додатку з’являться ті категорії інформації, тематику яких обрав користувач, то на його електронну пошту прийде посилання. Приклад посилання з книгами можна побачити на рисунку 3.9.

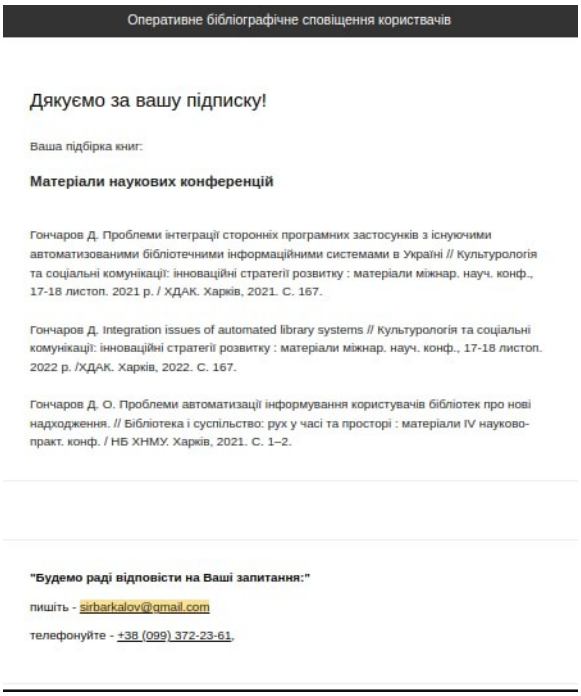


Рис. 3.9 – Нові надходження.

Адміністративна панель додатку дозволяє додавати-редагувати-видаляти усі сутності в базі даних напряму. Наприклад, на рис. 3.10 можна побачити додані дві книги. Сайт може працювати без панелі адміністратора, якщо книги експортуються напряму з АБІС.

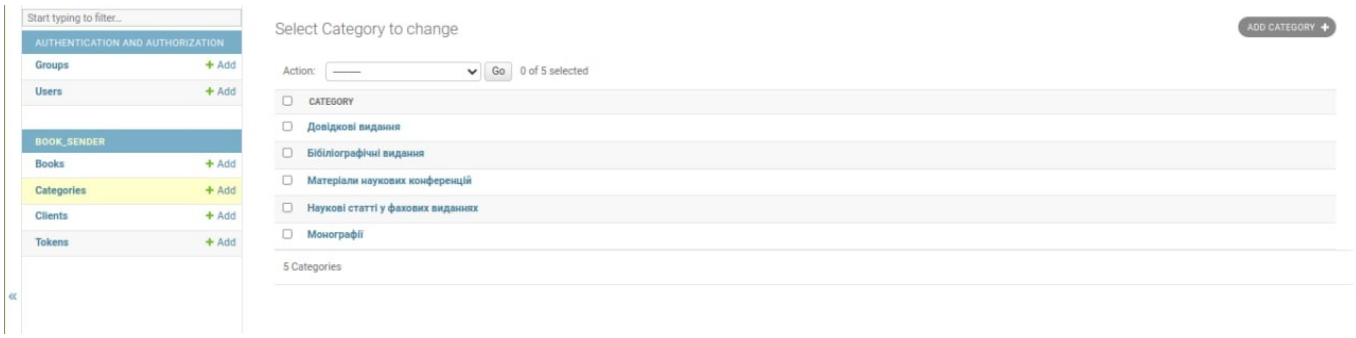


Рис. 3.10 – Адміністративна панель додатку

Таким чином, алгоритм дій користувача в додатку наступний: йому потрібно зайти на сайт бібліотеки, де вказане посилання на сервіс, перейти по посиланню та ознайомиться з інформацією, вказати свою електронну пошту та отримати посилання на неї від бібліотеки, перейти по посиланню, обрати

перелік необхідних йому типів нових надходжень, повідомлення про які йому необхідні, отримати їх та зберегти цей список. Виконавши ці дії, користувач буде отримувати бібліографічні повідомлення з кожним новим надходженням до бібліотечних фондів.

3.2 Оптимальні технологічні рішення

Серверна частина додатку створена за допомогою бібліотек Django та Django REST Framework [62], що використовує архітектуру Model-Template-View [50]. Ця архітектура реалізує обрану багатошарову архітектуру, Model – це шар взаємодії з даними, Template – це шар відображення, а View та доданий клас UseCase – реалізує шар бізнес-логіки. Запит до системи починається з View, View робить запити до Model, щоб отримати необхідні дані з бази даних, Template генерується на основі цих даних та повертається користувачу у вигляді веб-сторінки.

Важливою вимогою до додатку є безпека використання, тому була додана логіка токенної авторизації [9]. Коли користувач вводить електронну пошту, то автоматично генерується новий токен, який зберігається на клієнтській частині за допомогою механізму сесій. Приклад реалізації ідентифікації користувача можна побачити на рис. 3.11. Тут перевіряється, чи є актуальний токен на клієнтській частині, якщо є, то токен перевіряється в базі даних, якщо токен існує, то по токену знаходиться користувач. Якщо токена не існує, або токен вже не актуальний, то користувачу автоматично відкривається лендінгова сторінка, яка пропонує йому ввести електронну пошту.

```
def get(self, request):
    try:
        value = request.session['token']
    except KeyError:
        messages.error(request, 'Час дії посилання вичерпано. Ще раз введіть імейл')
        return redirect('home')
    token = Token.objects.filter(value=request.session['token'], expiration_time__gt=datetime.now())
    if token.exists():
        client = Client.objects.filter(token=token.first()).first()
```

Рис. 3.11 – Ідентифікація користувача

Для реалізації відправлення повідомлень про нові надходження використовується шаблон проектування «Спостерігач» [46], який реалізує механізм підписки. Тобто користувачі підписуються на категорії, а об’єкт-спостерігач дізнається, що якісь з категорій оновились новими виданнями та розсилає користувачам повідомлення. Приклад реалізації шаблону, а саме дію функції для розсилання користувачам повідомлень, можна побачити на рис. 3.12. Для кожного користувача перевіряються усі категорії, на які користувач підписаний, якщо знаходяться нові книги, то вони відправляються на пошту користувачеві.

```
@classmethod
def notify(cls):
    data = {}
    for client in Client.objects.all():
        changes = False
        for category in Category.objects.filter(client=client):
            books = Book.objects.filter(categories=category, is_sended=False).distinct()
            if books.exists():
                changes = True
                data[category.name] = books.all()

        context = {"data": data}
        html = get_template("email_books.html").render(context)
        if changes:
            send_email_message(subject="New books", recipient_list=[client.email], html=html,
                               from_email=settings.EMAIL_HOST_USER)
```

Рис. 3.12 – Нотифікація користувача

Коли користувач перший раз вводить пошту, то в базі даних створюється запис про реєстрацію нового користувача, якщо користувач вже хоч раз заходив в додаток, то користувач не реєструється, а шукається в базі даних. Після того автоматично видаляються всі старі токени, які вже не актуальні, та створюється новий, відправляється тимчасове посилання на електронну пошту. Реалізацію цього алгоритму дій можна побачити на рис. 3.13.

```
def post(self, request):
    user = Client.objects.get_or_create(email=request.POST['email'])[0]
    Token.objects.filter(client_email=user).all().delete()
    send_confirmation_link(user)
    messages.error(request, 'Посилання було відправлено на вашу email адресу')
    return redirect("home")
```

Рисунок 3.13 – Генерація тимчасового посилання

На рис. 3.13 немає коду для генерації нового коду, бо це працює автоматично. Щоб така логіка працювала автоматично, був перевизначений стандартний метод збереження даних в Model Token, який відображає сутність функціонування Токен у базі даних. Алгоритм реалізації функції перевизначення проілюстрований на рис. 3.14. Токен актуальний 24 години.

```
class Token(models.Model):
    expiration_time = models.DateTimeField(blank=True)
    value = models.CharField(max_length=255, blank=True)
    client_email = models.ForeignKey(Client, on_delete=models.CASCADE)

    def save(
        self, force_insert=False, force_update=False, using=None, update_fields=None
    ):
        self.expiration_time = datetime.now() + timedelta(days=1)
        self.value = uuid4()
        super().save(force_insert, force_update, using, update_fields)
```

Рис. 3.14 – Створення токену

Завдяки такому технологічному рішенню адміністратору системи не треба слідкувати вручну за кожною зміною токена. Для забезпечення унікальності та достатньої підміни розшифрування токену використовується бібліотека uuid [58].

Відправлення повідомлень по електронній пошті забезпечується внутрішніми можливостями Django та функцією send_email_message. Завдяки цієї функції можна додавати Template до повідомлень. Особливості реалізації цієї функції можна побачити на рис. 3.15.

```
def send_email_message(
    subject,
    recipient_list,
    html,
    text="",
    from_email=settings.EMAIL_HOST_USER
):
    msg = EmailMultiAlternatives(subject, text, from_email, recipient_list)
    msg.attach_alternative(html, "text/html")
    return msg.send()
```

Рис. 3.14 – Відправлення повідомлення користувачеві

Для створення Template, які описують шаблон повідомлення, краще застосовувати мову HTML та інструмент jinja2 [61], який використовує Django. Шаблон для відправлення повідомлення про нові надходження відбиває рис. 3.15.

```
<span class="email-text">
        Ваша підбірка книг:
        {% for category, books in data.items %}
                <h3>{{category }}</h3>
                {% for book in books %}
                        {{ book.author }}: {{ book.title }},
                {% endfor %}
        {% endfor %}
</span>
```

Рис. 3.15 – Шаблон повідомлення користувачу про нові надходження

Тимчасове посилання складається з двох етапів: перевірка токену та редірект на сторінку з категоріями, якщо токен валідний. Якщо користувач переходить, та його токен валідний, то клієнтська частина запам’ятовує токен за допомогою інструмента сесій. Реалізацію цієї логіки можна побачити на рис. 3.16.

```
class ConfirmEmailView(View):

    def get(self, request, token):
        my_token = Token.objects.filter(
            value=token,
            expiration_time__gt=datetime.now()
        )
        if my_token.exists():
            client = my_token.first().client_email
            my_token.delete()
            new_token = Token.objects.create(client_email=client)
            request.session['token'] = str(new_token.value)
            return redirect('choose_category')
        else:
            messages.error(
                request,
                'Час дії посилання вичерпано. Ще раз введіть імейл'
            )
            return redirect('home')
```

Рис. 3.16 – Логіка тимчасового посилання

Коли користувач переходить по тимчасовому посиланню, то система ідентифікує цього користувача по валідному токену і тоді наступним кроком відкриється сторінка з типами нових надходжень до бібліотечного фонду. На цю сторінку передаються усі типи видань, обрані користувачем як релевантні його інформаційними запитам і потребам. Для цього до шаблону передаються два масиви даних: з усіма категоріями та тільки з обраними. Якщо обрані типи видань змінюються, то другий масив оновлених даних відправляється на сервер. Реалізацію цієї логіки можна побачити на рис. 3.17.

```
client = Client.objects.filter(token=token.first()).first()
    client.categories.clear()
    for name in request.POST:
        if name != 'csrfmiddlewaretoken':
        category = Category.objects.filter(id=name)
        if category.exists():
            client.categories.add(category.first())
messages.error(request, 'Нові обрані категорії збережено'
```

Рис. 3.17 – Зміна інформаційних потреб користувача

Якщо користувач отримав повідомлення, що «Нові обрані типи видань збережено», то це означає, що повідомлення про нові надходження будуть надходити згідно до обраних ним пріоритетів. Взагалі, повідомлення про помилки або успішні повідомлення, реалізуються за допомогою інструмента Django messages. Реалізацію такого повідомлення можна побачити на рис. 3.18.

```
{% if messages %}
{% for message in messages %}
<div class="alert alert-primary alert-dismissible fade show mx-5" role="alert">
   {{ message }}
   <button type="button" class="close" data-dismiss="alert" aria-label="Close">
   <span aria-hidden="true">&times;</span>
   </button>
</div>
{% endfor %}
{% endif %}
```

Рис. 3.18 – Шаблон повідомлень

Наступні програмні рішення стосуються хмарної частини. Хмарна частина взаємодіє з серверною на основі API, який реалізований за допомогою Django REST Framework. API endpoint, що створений для отримання інформації про нові надходження, приймає реалізовані [56] дані у вигляді json файлу. Якщо нова книга належить до категорії, якої ще не існувало в додатку, то така категорія створюється автоматично. Якщо книга вже існує у додатку, то дані не дублюються, а оновлюються.

Алгоритм реалізації додатком отримання інформації про нові надходження можна побачити на рис. 3.19. Після кожного оновлення додатку новими надходженнями користувачам автоматично відправляються повідомлення.

```
for category_book in data[category]:
   serializer = BookSerializer(data=category_book)
   if serializer.is_valid():
      book_data = serializer.validated_data()
      current_book = Book.objects.get_or_create(title=book_data['title'])[0]
      current_book.author = book_data['author']
      current_book.categories.add(current_category)
      current_book.save()
```

Рис. 3.19 – Отримання інформації про нові надходження

Останнє програмне рішення стосується взаємодії с соціальним мережами. У поточній версії додатку реалізовані повідомлення лише за допомогою

telegram. Систему можна масштабувати новими соціальними мережами у майбутньому.

Для роботи з telegram ботами використовується додаткова бібліотека telebot [59]. Коли користувач вперше запускає бота, то перевіряється, чи був ідентифікатор користувача в телеграм доданий в систему, якщо був, то користувач буде отримувати повідомлення в телеграм. Якщо в telegram немає ідентифікатора користувача, то його потрібно додати в клієнтській частині. Нажаль telegram має ліміт кількості символів у повідомленні, тому для кожної категорії асинхронно надсилається окреме повідомлення. Реалізацію початку роботи з ботом можна побачити на рис. 4.20.

```
@bot.message_handler(commands=["start"])
def start(m, res=False):
    if Client.objects.filter(telegram_login=m.from_user.username).exists():
        bot.send_message(
            m.chat.id,
            'Hello. I start sending you notifications '
        )
    else:
        bot.send_message(
            m.chat.id,
             'Please go to our site and add telegram login to your profile '
        )
```

Рис. 4.20 – Телеграм бот.

Якщо зупинити роботу бота, чи видалити його, то бот не буде надсилати повідомлення. Поновити роботу бота можна, якщо ще раз оновити ідентифікатор користувача в клієнтській частині.

3.3 Вимоги до програмного забезпечення

Стандартне тестування додатку означає виконання усіх рівнів піраміди тестування, яку можна побачити на рис. 4.21. Найбільше в системі створено

модульних тестів, є тести взаємодії сервісів (API тести) та тестування графічного інтерфейсу.

Розробка програмного забезпечення має постійно супроводжуватися мануальним тестуванням, а здатність хмарного сервісу масштабуватися тестується за допомогою навантажувального тестування.

Модульне тестування виконувалось за допомогою інструмента для мови програмування python – pytest [27]. Інструмент дозволяє в декілька строк коду писати невеликі тести, вказавши вхідні та очікувані вихідні дані для кожної функції додатку. Фреймворк не підходить для складного функціонального тестування програм, але підходить для невеликих додатків. На серверній частині 15 тестів, результат проходження яких можна побачити на рис. 4.22.

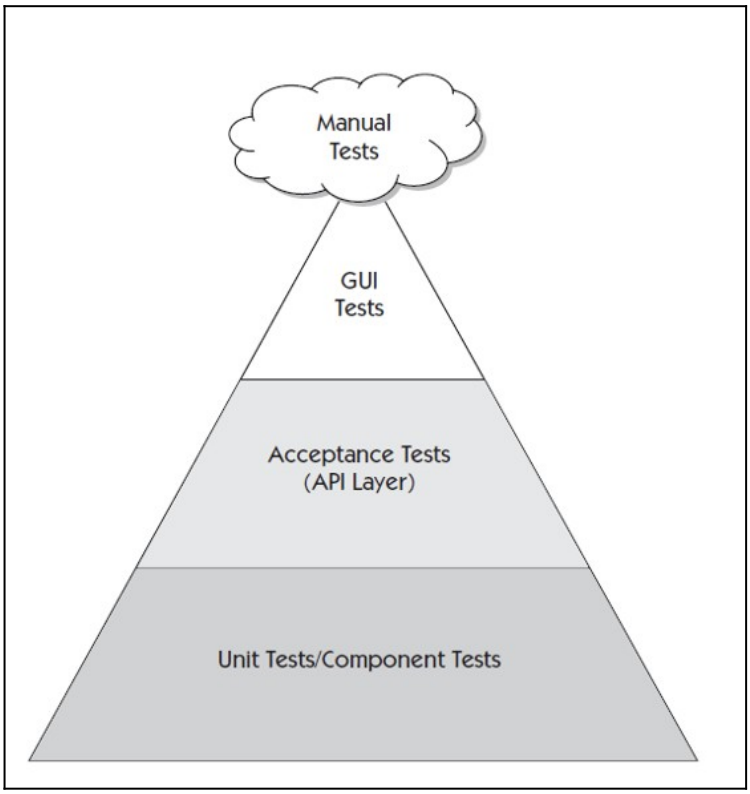


Рис. 4.21 – Піраміда тестів

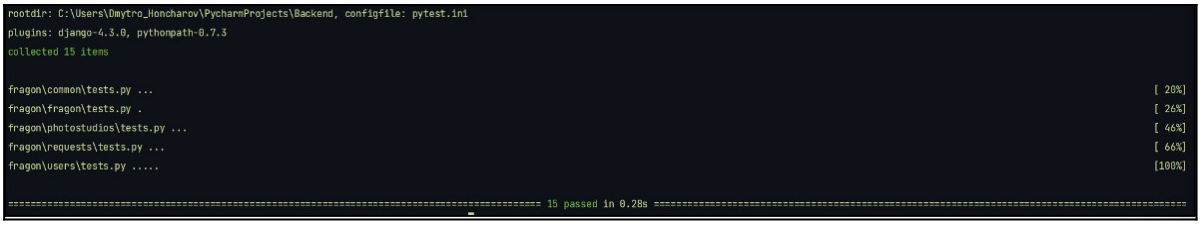


Рис. 4.22 – Успішність тестування

Модульні тести перевіряють генерацію токенів, додавання-видалення категорій та надсилання нових надходжень. Приклад модульного тесту, який перевіряє доступність лендінгової сторінки, можна побачити на рис. 4.23.

```
class HomeTestCase(TestCase):

    def test_health_check(self):
        c = Client()
        response = c.get("/")
        self.assertEqual(response.status_code, 200)
```

Рис. 4.22 – Модульний тест

Інтеграційне тестування потрібне для перевірки роботи REST API, тобто перевірку отримання нових надходжень. Для інтеграційного тестування використовувався Postman, який дозволяє відправити HTTP запит та перевірити коректність відповіді. Тест POST_New_books перевіряє правильність надсилання запиту з хмарного застосунку на сервер. Інтеграційний тест в Postman [33] можна побачити на рис. 4.23.

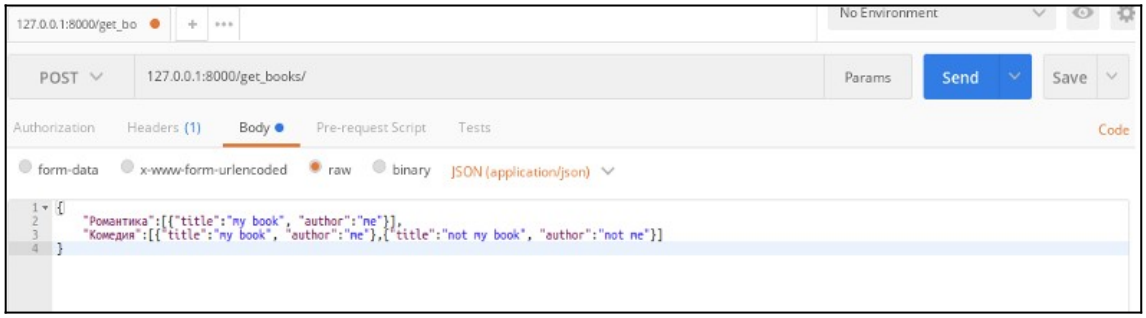


Рис. 4.23 – Інтеграційний тест Postman

Тестування інтерфейсу користувача було проведено за допомогою сервісу Google Lighthouse [54], який дозволяє виміряти доступність, оптимізацію та швидкість роботи застосунку. Успішність результатів тестування можна побачити на рис. 4.24.

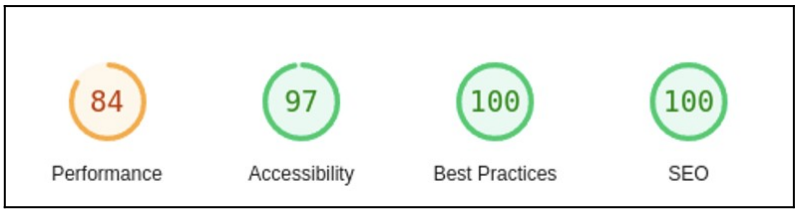


Рис. 4.23 – Google Lighthouse-тестування

Результат Performance – швидкість відповіді трохи нижче норми, але це пов’язано з тим, що сервісу потрібен кращий сервер. Це перевірено за допомогою навантажувального тестування сервісом Locust [20]. Кожну секунду створювалось 10 великих запитів на оновлення надходжень до бібліотеки від різних емульованих користувачів, система довела стійкість до навантаження. Якщо запитів зроблено забагато, або у випадок атаки на сервер, сервер автоматично перезавантажиться. Результат навантажувального тестування можна побачити на рис. 4.24.

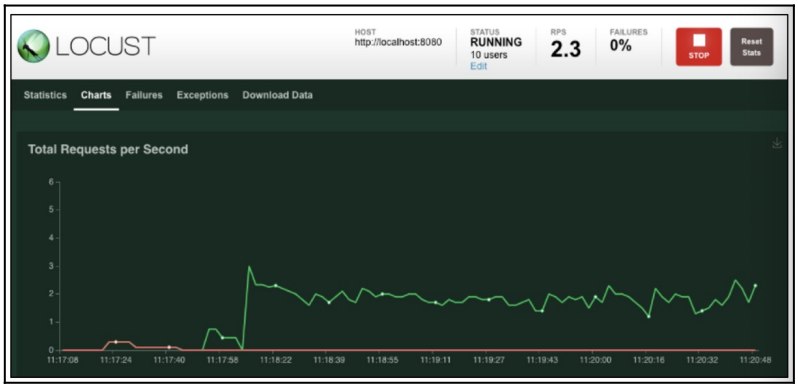


Рис. 4.24 – Locust тестування

Результат усіх видів тестування успішний, що означає, що сервіс відповідає встановленим вимогам та готовий до використання.

ВИСНОВКИ

В результаті виконання кваліфікаційної магістерської роботи були вивчені сучасні способи інформування читачів про нові надходження та створений новий програмний застосунок для автоматизації індивідуального бібліографічного обслуговування користувачів бібліотек, який функціонує на основі отримання повідомлень на електронну пошту та в месенджерах.

Аналіз існуючих програмних застосунків допоміг зрозуміти проблеми існуючих АІБС та сформулювати вимоги до нового додатку, які вирішують ці проблеми. Існуючі системи застарілі, потребують багато часу для підтримки та не здатні інтегруватися між собою, не мають достатньо можливостей для персоналізації вподобань читачів. Деякі системи надають широкий функціонал, але лише в локальній мережі або за допомогою складних для використання інструментів, таких як RSS-стрічка. Більшість публічних систем, таких як оприлюднення інформації на офіційному сайті бібліотеки або її аккаунті в соціальних мережах, потребують від працівників бібліотек великих витрат часу на введення даних, тому інформація на таких ресурсах може бути неповною або бути наданою через великий проміжок часу, бо система не інтегрована з АБІС.

Була поставлена задача на створення додатку, який відповідає наступним вимогам: його можна інтегрувати з різними бібліотеками, не створюючи ризиків для витоку персональних даних; він зможе працювати в сучасних месенджерах або надсилати дані по електронній пошті; він буде легким для застосування як для користувачів, так і для працівників бібліотек; його можна автоматично масштабувати для великої кількості користувачів; він має сучасний інтуїтивно зрозумілий інтерфейс, може інтегруватися з АБІС та не буде потребувати багато даних для ідентифікації користувача.

Після визначення технічних функціональних та нефункціональних вимог до додатку, була змодельована його архітектура, яка поділилася на два мікросервіси: базу даних та хмарний додаток і багатошаровий моноліт: клієнт-

серверний додаток. Була нормалізована схема бази даних, щоб уникнути дублювання та змодельована бізнес-логіка, спроектований графічний інтерфейс.

Були прийняті додаткові програмні рішення, щоб покращити безпечність використання додатку: доданий функціонал тимчасових посилань та логіка надсилання повідомлень. Користувачу, який бажає стати абонентом системи індивідуального бібліографічного інформування, необхідно зайти до системи через уведення адреси електронної пошти та отримати посилання, яке працює 24 години. Це швидкий та надійний спосіб ідентифікації користувача. Повідомлення користувачеві надсилаються, коли дані про нього експортуються з АБІС в хмарний додаток.

Модель інформаційної системи була реалізована у програмному додатку, який пройшов мануальне, інтеграційне, модульне, навантажувальне тестування, а також тестування інтерфейсу. Програмний застосунок відповідає вимогам, масштабується та готовий до впровадження в практику роботи бібліотек.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Автономова Н. Інформаційні продукти та послуги як результат виробничої діяльності бібліотек // Наукові праці Національної бібліотеки України ім. В. І. Вернадського. Київ, 2009. С. 255.

2. Бережна К. Динаміка техніко-технологічних трансформацій публічних бібліотек України // Вісник Харківської державної академії культури : зб. наук. пр. Харків, 2017. Вип. 52. С. 101-111.

3. Бережна К. Публічні бібліотеки України: вектори модернізації в інформаційному суспільстві : дис. канд. наук із соціальних комунікацій : 27.00.03. Харків : ХДАК, 2018. 228 с.

4. Бичко О., Кандул А. Сучасні тенденції інформування науковців університету // Книги і бібліотеки в глобалізованому світі. Хмельницький. 2019. С. 86.

5. Вільям С. В. Django for APIs.x. Санта Клара, Апресс, 2020. 99 с.

6. Гончаров Д. Integration issues of automated library systems // Культурологія та соціальні комунікації: інноваційні стратегії розвитку : матеріали міжнар. науч. конф., 17-18 листоп. 2022 р. /ХДАК. Харків, 2022. С. 167.

7. Гончаров Д. Проблеми автоматизації інформування користувачів бібліотек про нові надходження. НБ ХНМУ. Харків, 2021. С. 1–2.

8. Гончаров Д. Проблеми інтеграції сторонніх програмних застосунків з існуючими автоматизованими бібліотечними інформаційними системами в Україні // Культурологія та соціальні комунікації: інноваційні стратегії розвитку : матеріали міжнар. науч. конф., 17-18 листоп. 2021 р. / ХДАК. Харків, 2021. С. 167.

9. Гончарук Б. Г., Арсенюк І. Р. Моделювання процедури захисту веб-ресурсів та веб-сервісів // INTERNET-EDUCATION-SCIENCE (IES-2020) : Proceedings of the XII International scientific-practical conference, Ukraine, Vinnytsia, 26-29 May 2020. Винниця : ВНТУ, 2020. С. 84-87.

10. Гулик І. Індивідуальне бібліотечне обслуговування. Вінниця, 2011. 20 с.

11. Давидова І. Інноваційна політика бібліотек України: зміст та стратегії розвитку в інформаційному суспільстві : автореф. дис. д-ра наук з соціальних комунікацій : 27.00.03. Харків : ХДАК, 2008. 51 с.

12. Давидова І. Соціально-комунікаційна теорія бібліотечної діяльності: становлення та шляхи розвитку // Бібліотечний вісник. 2014. №2. С. 8-12.

13.Давидова І. Стратегії розвитку бібліотек України за умов інформаційного суспільства // Вісник Книжкової палати. 2008. №11. С. 24-27.

14. Давидова І. Трансформаційні та інноваційні зміни у бібліотечній сфері діяльності: теоретико-методологічні проблеми співвідношення // Вісник Книжкової палати. 2008. №4. С. 20-23.

15. Єфремов М. Ф., Єфремов, Ю. М., Єфремов, В. М. Проектування програмного забезпечення з використанням UML. Харків, 2016. 214 с.

16.Завадський І. Основи баз даних. Київ : Видавець Завадський, 2011. 256 с.

17. Каліберда Н. Інноваційні аспекти обслуговування читачів // Наукові праці Національної бібліотеки України. 2004. Вип. 47. С. 143-155.

18. Каліберда Н., Бровкін А., Формування та використання бібліотечно-інформаційних ресурсів: традиції і інновації. 2009. №1. С. 7-10.

19.Каліберда Н.Організаційні структури обслуговування в національних бібліотеках // Бібліотечний вісник. 1998. №. 4. С. 15-18.

20. Ковш М. Реалізація підсистеми для розподіленого високо-навантажувального тестування та аналізу результатів у реальному часі в системі CI/CD. Київ. 2021. 86 с.

21. Козачок В. А., Діхтяр М. В., Семко О. В. Особливості ідентифікації та авторизації в сучасних корпоративних інформаційно-телекомунікаційних системах. // Сучасний захист інформації. Харків, 2017. 7 с.

22. Колпак М. В. Система електронного документообігу підприємства // Bachelor's Thesis / КПІ ім. Ігоря Сікорського. Київ, 2020. С. 1-20.

23. Костирко Т. Бібліотеки вищих навчальних закладів у єдиному інформаційному просторі сучасного суспільства. Київ. 2002. С. 68-76.

24. Кучер В. В. Мікросервісна архітектура та її особливості. Житомир. 2018. С. 158-161.

25. Лобузіна К. Знання, сховища даних та експерти // Бібліотекознавство. Документознавство. Інформологія. 2012. Вип. 3. С. 27-35.

26. Лобузіна К., Онищенко О. С. Технології організації знаннєвих ресурсів у бібліотечно-інформаційній діяльності : монографія. Київ : НБУВ. 2012. 252 с.

27. Марущак Я. В., Коцюбинський В. Ю., Дрожнікова Ю. О. Аналіз інструментів для автоматизації тестування на мові програмування Python // Вісник ВНТУ. 2020. Вінниця. С. 1-2.

28. Мар'їна О. Бібліотека в епоху розвитку технологій Web 3.0 // Вісник Книжкової палати. 2015. №11. С. 18-20.

29. Мар'їна О. Бібліотеки в цифровому медіапросторі: монографія. Харків, 2017. 355 с.

30. Мар'їна О. Корпоративні бібліотечні проекти як засіб формування соціокомунікаційного середовища // Вісник Харківської державної академії культури. 2009. Вип. 47. С. 123-131.

31. Мар'їна О. Розвиток корпоративних бібліотечних проектів в Україні // Вісник Книжкової палати. 2010. № 3. С. 22-25.

32. Мар'їна О. Сучасні форми інформаційно-комунікаційної взаємодії в регіональних бібліотечних системах // Вісник Книжкової палати. 2009. №5. С. 20-23.

33. Мелкозьорова О., Гайкова В., Опис схеми API тестування програмного забезпечення // Комп'ютерні науки та кібербезпека. 2019. С. 38-45.

34. Мельник М., Головко Н., Горбенко О. Форми і методи роботи з читачами: як привернути увагу до бібліотеки. Вінниця, 2015. 20 с.

35. Нові надходження до бібліотечного фонду // НФУ : офіційний сайт. URL: https://lib.nuph.edu.ua

36. Про захист персональних даних : закон України // Відомості Верховної Ради України, 2010. № 34. С. 481.

37. Про основні засади забезпечення кібербезпеки України : закон України // Відомості Верховної Ради України, 2017. № 45. С.403.

38. Сашкова Л. Інформаційно-бібліографічна робота публічної бібліотека. Харківська обласна універсальна наукова бібліотека. Харків. 2015. С. 3.

39. Свінцицька О., Граф М. Метод Use Case в плануванні проєктів з інформаційних технологій // Технічна інженерія. №1(89). 2022. С. 77-84.

40. Сербін О. Бібліотечна галузь: зміцнення позицій. URL : https://ukurier.gov.ua/uk/articles/bibliotechna-galuz-zmicnennya-pozicij/

41. Сербін О., Ярошенко Т.О. Аспекти реформування сучасної бібліотечної освіти // Вісник Книжкової палати. 2015. №12. С. 12-13.

42. Сервіс Google тренди. URL: https://trends.google.com

43. Співак І. Проектування баз даних. Харків, 2021. 256 с.

44. Станкевич, Н. С. Телеграм-бот для пошуку інформації в RSS-стрічках // BS thesis / КПІ ім. Ігоря Сікорського. Київ, 2021. С. 36-38.

45. Стельмащук Р, Булатецька Л. В. Огляд хмарних сервісів для побудови ER-діаграм // Математика. Інформаційні технології. Освіта. Луцьк. 2022. 125 с.

46. Харченко М. Аналіз та використання патернів проектування. Київ. 2021. С. 30-43.

47. Швецова-Водка Г. М. Вступ до бібліографознавства : навч. посіб. для студентів напряму 6.020102 “Книгознавство, бібліотекознавство і бібліогр.” / Рівнен. держ. гуманітар. ун-т. 3-тє вид., випр. та допов. Рівне, 2011. 231 с.

48. Швецова-Водка Г. М. Документознавство : словник-довідник термінів і понять : навч. посіб. Київ : Знання, 2011. С. 319.

49. Шумко Л. Сучасний інформаційний сервіс у публічній бібліотеці : методичні рекомендації. Черкаси. 2012. 20 с.

50. Alchin M. Understanding Django // Pro Django, 2009. p.1-12.

51. Amlib Info. URL: https://www.amlib.co.uk

52. Book-RSS Github Repo. URL: https://github.com/azu/book-rss

53. Feng Xinyang; Snen Jianjing; FAN, Ying. REST: An alternative to RPC for Web services architecture // First International Conference on future information networks. IEEE, 2009. p. 7-10.

54. Heričko, T., Šumak, B., Brdnik, S. Towards Representative Web Performance Measurements with Google Lighthouse // 7th Student Computer Science Research Conference. Maribor, Slovenia. 2021. P. 39.

55. Hillar, Gaston C. Django RESTful Web Services: The Easiest Way to Build Python RESTful APIs and Web Services with Django. Packt Publishing Ltd, 2018.

56. Kellogg, G., Champin, P. A., Longley, D. Json-ld 1.1–a json-based serialization for linked data. Beihang. 2019. 215 p.

57. Koha Demo. URL: https://koha-community.org/demo/

58. Leach, P., Mealling, M., Salz, R. A universally unique identifier (uuid) urn namespace. 2005. P. 3.

59. Modrzyk, N. Building telegram bots: develop bots in 12 programming languages using the telegram bot API. Apress. 2018. P. 23-65

60. Rittinghouse, J.W. and Ransome, J.F., Cloud computing: implementation, management, and security // CRC press, 2017. P. 13-25.

61. Ronacher A. Jinja2 documentation. Welcome to Jinja2—Jinja2 Documentation (2.8-dev). 2008. 34 p. URL: mitsuhiko.pocoo.org/jinja2docs/Jinja2.pdf

62. Rubio D. REST services with Django // Beginning Django. Apress, Berkeley, CA, 2017. p. 549-566.

63. Vincent, William S. Django for APIs: Build web APIs with Python and Django. WelcomeToCode, 2022.

64. Yergeau, François. UTF-8, a transformation format of ISO 10646. 1998. 10 p. URL: https://datatracker.ietf.org/doc/html/rfc2279.