Universidad Autónoma de Madrid

Escuela Politécnica Superior

24|25

Bachelor thesis

Full-stack implementation of a book recommender system

María Paloma Ruiz Matesanz

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Bachelor as Ingeniería Informática Bilingüe

BACHELOR THESIS

Full-stack implementation of a book recommender system

Author: María Paloma Ruiz Matesanz
Advisor: Alejandro Bellogin Kouki

June 2025

Francisco Tomás y Valiente, n^o 1
Madrid, 28049
Spain

María Paloma Ruiz Matesanz
Full-stack implementation of a book recommender system

María Paloma Ruiz Matesanz
C\ Francisco Tomás y Valiente Nº 11

PRINTED IN SPAIN

Agradecimientos

En primer lugar, quiero agradecer a mis padres por su apoyo incondicional, su amor constante y la inmensa paciencia que siempre han tenido conmigo. Nunca podré expresar lo agradecida que estoy de tenerlos en mi vida.

A mis amigos, gracias por hacer que estos años sean realmente inolvidables.

A G y S, porque, a pesar de todo, hemos formado un equipo imparable.

A todos mis profesores, que habéis sido una guía y una fuente de conocimiento invaluable. Y, muy especialmente, a Alejandro Bellogín, por su dedicación sin límites, tanto en este trabajo como en todas las asignaturas en las que he tenido el privilegio de ser su estudiante.

Y, por último, a mi abuelo, que siempre me animó desde pequeña a ser ingeniera. Aunque no hayas podido verlo, lo he conseguido.

Resumen

Los libros han sido un pilar fundamental en el desarrollo de la humanidad, permitiendo la transmisión de conocimiento, cultura e ideas a lo largo del tiempo. Gracias a ellos, el saber ha trascendido generaciones, moldeando nuestra historia y pensamiento. Sin embargo, en la era actual, caracterizada por una sobreabundancia de información, encontrar el libro adecuado se ha vuelto un desafío cada vez mayor. En 2024, solo en España se publicaron más de 89.000 títulos [1], lo que amplía constantemente un universo literario que, lejos de facilitar la elección, la complica aún más.

Ante esta situación, este trabajo propone el desarrollo de una herramienta capaz de guiar a los lectores a través de este inmenso mundo literario, ayudándoles a encontrar lecturas acordes a sus gustos e intereses. Para ello, se ha llevado a cabo un proceso completo de desarrollo desde el principio. En primer lugar, se ha construido un conjunto de datos a partir de datos extraídos de la página web *Casa del Libro*, la mayor cadena de librerías de España [2]. Dichos datos han sido preprocesados y utilizados para calcular similitudes entre libros, elemento clave para la posterior implementación del sistema de recomendación.

A partir de esta base, se ha diseñado y desarrollado una aplicación web completa utilizando tecnologías como Python, PostgreSQL, Django y Nuxt. Todos los algoritmos de recomendación empleados han sido implementados desde cero, sin utilizar librerías externas específicas para este propósito, con el objetivo de comprender y controlar completamente su funcionamiento. Además, se ha puesto un especial énfasis en la experiencia de usuario, buscando crear una plataforma lo más accesible y amigable posible, apta para cualquier persona, independientemente de su perfil técnico, y disponible desde cualquier lugar y dispositivo.

Finalmente, una vez desarrollada la aplicación, se ha llevado a cabo un estudio con usuarios reales para recopilar feedback y evaluar la eficacia del sistema, cerrando así el ciclo de diseño, desarrollo y validación del recomendador.

Palabras clave

Sistema de recomendación, Scraping, Crawling, Filtrado de contenido, Similitud de coseno, Embeddings

Abstract

Books have been a key part of human development, transmitting knowledge, culture, and ideas over the centuries. They have allowed knowledge to transcend generations, shaping history and thought along the way. However, in today's world, overwhelmed by information, finding the right book has become a growing challenge. In Spain alone, more than 89,000 titles were published in 2024 [1], continuously expanding a wide literary universe that complicates rather than simplifies the selection process.

This project suggests the development of a tool to guide readers through this huge literary world and help them find books that match their tastes and interests. The starting point was creating a dataset from data extracted from Casa del Libro, Spain's largest bookstore chain [2]. This data was preprocessed and used to calculate similarities between books, a key step for implementing the recommendation system.

Building on this foundation, a full web application was designed and developed using Python, PostgreSQL, Django, and Nuxt. All recommendation algorithms were implemented from scratch to fully understand and control their behavior without relying on external libraries. Special attention was given to user experience, aiming to create a platform that is accessible and user-friendly for anyone, regardless of technical background, and available on any device and location.

Finally, once the application was complete, a user study was conducted to gather feedback and evaluate the effectiveness of the system, closing the loop of design, development, and validation of the recommender.

Keywords

Recommender system, Scraping, Crawling, Content filtering, Cosine similarity, Embeddings

Table of Contents

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Goals . . . 2

1.3 Work Structure . . . 2

2 State of the Art **3**

2.1 Recommender Systems . . . 3

2.1.1 Content based filtering . . . 3

2.1.2 Collaborative filtering . . . 4

2.2 Commercial Book Systems . . . 6

2.2.1 Goodreads . . . 6

2.2.2 Casa del Libro . . . 7

2.3 Data Collection for Book Recommendation . . . 9

2.3.1 Web crawling . . . 9

2.3.2 Web scraping . . . 9

2.4 Web Application Frameworks . . . 10

2.4.1 Django framework . . . 11

2.4.2 Nuxt framework . . . 12

3 Design and Implementation **13**

3.1 Project Structure . . . 13

3.2 Requirements Analysis . . . 14

3.2.1 Functional requirements . . . 14

3.2.2 Non-functional requirements . . . 16

3.3 Design . . . 16

3.3.1 Data preprocessing . . . 17

3.3.2 Web application backend . . . 18

3.3.3 Web application frontend . . . 19

3.4 Implementation . . . 19

3.4.1 Data preprocessing . . . 21

3.4.2 Web application backend . . . 24

3.4.3 Web application frontend . . . 25

4 Experiments and Results **27**

4.1 Data Analysis . . . 27

4.2 User Experience and Feedback Analysis 29
4.2.1 User demographics and reading preferences 29
4.2.2 Evaluation results 33
4.2.3 Usability assessment with the SUS 38
5 Conclusions and Future Work **39**
5.1 Conclusions 39
5.2 Future Work 40
Bibliography **44**
Appendices **45**
A Coverless Interface **47**
A.1 Register 47
A.2 Login 48
A.3 Home 48
A.4 Language selection 49
A.5 Genres selection 49
A.6 Language and genre-based recommendations 50
A.7 New releases 50
A.8 Search bar 51
A.9 Book detail page 51
A.10 Favorite books page 53

Lists

List of codes

3.1 Preload embeddings . . . 23
3.2 Genre similarity calculation . . . 24

List of equations

2.1 Cosine similarity . . . 4
2.2 Pearson correlation . . . 5

List of figures

2.1 Comparison between collaborative filtering and content-based filtering. . . . 6
2.2 Book rating comments, collected from the website *Casa del Libro*. . . . 8
2.3 Comparison between web crawling and web scraping . . . 10
2.4 MTV architecture diagram . . . 12

3.1 Dataflow . . . 14
3.2 Dataset collection design . . . 17
3.3 Database entity-relationship diagram . . . 18
3.4 Sequence diagram: User's favorite book list and recommended books . . . 20
3.5 Homepage of Coverless . . . 26

4.1 Number of books by genre . . . 27
4.2 Number of books by year . . . 28
4.3 Number of books by price range . . . 28
4.4 Number of books by publisher . . . 29
4.5 User consent and participant ages . . . 30
4.6 Participant demographics and reading habits . . . 30
4.7 Users favorite genres . . . 31
4.8 User strategies for book discovery . . . 31
4.9 Users criteria for book selection . . . 32
4.10 Book-related applications used by readers . . . 32

4.11 Relevance of reviews in book selection 32
4.12 User registration experience 33
4.13 User ratings on password selection difficulty 33
4.14 User login experience 33
4.15 User feedback on language and genre selection 34
4.16 User feedback on the recommendation system 34
4.17 User opinions on the new releases recommender 35
4.18 User experience navigating the book section 35
4.19 User feedback on the book section filters and sorting options 35
4.20 Ability to find relevant books in the book section 36
4.21 User feedback on the search functionality 36
4.22 Clarity and sufficiency of book information 36
4.23 User feedback on similar and same-author book suggestions 37
4.24 Access to user's favorite list 37
4.25 User feedback on favorite book features 38

A.1 Register Coverless page 47
A.2 Login Coverless page 48
A.3 Home Coverless page 48
A.4 Book language selection page 49
A.5 Book genre selection page 49
A.6 Language and genre-based recommendations 50
A.7 New releases page 50
A.8 Search bar 51
A.9 Search page 51
A.10 Book detail page 52
A.11 More from the same author 52
A.12 Recommender books 52
A.13 Favorite books page 53
A.14 Recommender based on the favorite books 53

List of tables

4.1 User ratings for app sections 38

1 Introduction

Books have been a key part of the development of humanity, transmitting knowledge, culture, and discoveries over the centuries. They have enabled ideas to transcend generations, shaping history along the way. However, in a world saturated with information, the art of finding the right book has become a significant challenge. This project aims to develop a tool that guides readers through this wide world of literature to find their ideal read.

1.1 Motivation

Books have always been a part of my life, from childhood to the present day. Reading has not only been a source of entertainment, but also a means of learning and personal growth. When the time came to choose a topic for my Bachelor's thesis, I saw the perfect opportunity to merge my two greatest passions: computer science and literature. Bringing these two worlds together was not only a natural choice, but also a way to offer a practical solution to a real need in today's literary landscape.

The publishing industry is currently experiencing fast and constant growth, with countless new titles released each month (e.g. see [3]). Although this abundance can be seen as positive, it often leaves readers feeling overwhelmed by the wide variety of options. With this in mind, I came up with the idea of developing a book recommender system: an intelligent tool to support readers in their decision-making process. Such a system can help users discover books aligned with their interests while also encouraging exploration beyond their usual preferences. Great books often go unnoticed simply because they never reach the right reader at the right time.

Recommender systems not only help readers but also offer important advantages to publishers and book distributors. By improving the visibility of titles and enabling more effective personalization, these systems enhance user engagement and promote a more diverse and inclusive literary market. In a highly competitive environment, offering accurate and meaningful recommendations can make a significant difference in both reader satisfaction and publisher success.

1.2 Goals

The main objective of this Final Degree Project is to design and implement a complete web application capable of delivering content-based book recommendations. To achieve this, the project will begin with a custom data collection process using web crawling and scraping techniques on the website *Casa del Libro*. The extracted data will then go through preprocessing to ensure consistency and clean entries with missing or incorrect information.

Once the dataset is prepared, the next step involves calculating similarity scores between books based on different attributes, which will serve as the basis for the recommendation algorithms. These algorithms, selected through previous research, will focus on content-based filtering methods.

The final stage consists of developing the full web application, including both the frontend and back-end components, and connecting them to the previously built database that stores all the processed information.

Before the development of this project, there was no previous hands-on experience with web scraping, web crawling or content-based recommendation algorithms. However, during the process, it has become a valuable opportunity to learn and apply new knowledge, leading to significant growth in areas such as software engineering, web application development, and database design and management.

1.3 Work Structure

This thesis is structured into five chapters, each of which details a phase of the project's development.

Chapter 1. Introduction: This chapter provides a brief overview of the thesis topic, describing the motivation behind the project, its main objectives, and the structure of the document.

Chapter 2. State of the Art: This chapter presents the theoretical basis necessary to understand the development of this thesis. It includes a review of existing recommendation systems and the different approaches used in both academic and commercial contexts.

Chapter 3. Design and Implementation: This chapter details the design and development process of the project. It includes an analysis of both functional and non-functional requirements, the architecture and technologies selected, and the reason behind key decisions. Also, it describes the implementation aspects of the developed system.

Chapter 4. Experiments and Results: An analysis of the collected data will be presented, along with the results of the user tests conducted on the web application.

Chapter 5. Conclusions and Future Work: Finally, the conclusions drawn from the project will be presented, along with potential directions for future improvements and extensions.

2

State of the Art

This chapter will explain the fundamentals of recommender systems, present some examples of commercial book recommender systems, and outline the theoretical foundations necessary to understand the development of this thesis.

2.1 Recommender Systems

A recommender system is an artificial intelligence or AI algorithm, usually associated with machine learning techniques, used to suggest additional items that may be useful or valuable to users. These can be based on various criteria, including previous purchases, search history, demographic information, and other factors [4]. They handle the problem of information overload that users typically face by providing them with personalized, exclusive content, and service recommendations [5].

To understand how recommender systems work, we need to understand the three main models that exist: those that choose recommendations based on content (Section 2.1.1), those that do the same using interactions (Section 2.1.2), and hybrid models [6], which produce recommendations based on combining the previous methods.

2.1.1 Content based filtering

Content-based filtering (CB) is a type of recommendation algorithm that suggests items similar to those a user has previously liked or preferred [7]. These recommendations are generated based on the features and characteristics of the items themselves, identifying similarities to predict what the user might enjoy next [8]. The main principle is that users tend to like thematically similar content.

Typically, CB recommendation works in three main steps:

1.– **Content Analyzer:** Performs data preprocessing and extracts relevant information from the items. This often involves representing item through metadata or using techniques such as named entity recognition or keyword extraction. If metadata is available, they can be directly used for further computation.

2.– **Profile Learner:** Builds a user profile based on the user's preferences and interactions with previous items.

3.– **Filtering Component:** Matches new items with the user profile and suggests the most relevant ones.

To find similarities between items, various distance or similarity metrics can be used, such as cosine similarity or Euclidean distance. A common and effective approach is cosine similarity, which measures the cosine of the angle between the item vector (A) and the user profile vector (B):

$$\text{sim}(A, B) = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{A}| \cdot |\mathbf{B}|} \tag{2.1}$$

The user and item vectors can be formed as follows (this is just an example):

- **Item vector (book vector):** Each item is represented as a vector composed of characteristics extracted from its metadata or content. These characteristics can include genres, language, keywords from the book's description, author information, or other relevant attributes. Each dimension of the vector corresponds to the presence, absence, or strength of a particular characteristic. For example, if we consider genres, a book that belongs to "fantasy" and "romance" genres could be represented as:

$$\text{Book}_1 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0.5 & 0 & 0.8 \end{bmatrix}$$

  where each number reflects the relevance or weight of a feature (such as 1 for strong presence and 0 for absence).

- **User vector:** A user vector is generated by aggregating the feature vectors of the items the user has positively interacted with. This aggregation can be performed by averaging the feature vectors or by calculating a weighted sum based on the user's ratings. In this way, the user's preferences are captured in a single profile vector, which represents the types of content they are most interested in [9].

Based on the similarity value (ranging from -1 to 1), items are ranked in descending order. Then, one of the following strategies is applied:

- **Top-N approach:** Recommends the top N items with the highest similarity scores.
- **Rating scale approach:** Sets a threshold and recommends all items whose similarity score exceeds the threshold.

If the user profile changes, the CB filtering technique still has the potential to adjust its recommendations in a very short period of time. However, content-based algorithms often struggle to give recommendations that are both novel and aligned with the user's interests. This is because they tend to suggest items that are very similar to those the user has previously liked, especially when recommender systems prioritize accuracy over factors such as diversity or the ability to surprise the user with new options [10].

2.1.2 Collaborative filtering

Collaborative Filtering (CF) systems are among the earliest and most widely deployed recommendation techniques. CF systems recommend items to users based on the preferences of other users with similar tastes [10]. The main principle of CF is that if two users have historically agreed on item preferences, they will continue to agree on items in the future.

CF approaches are typically divided into two categories: model-based and memory-based methods:

- **Memory-based CF (also known as nearest neighbor recommender systems):** approaches rely directly on the entire historical user-item interaction matrix to make recommendations. These approaches compute similarities between users or items and use those similarities to generate predictions and recommendations. For example, in a user-user CF model, recommendations are made by finding the most similar users based on their ratings and suggesting items that these similar users have liked.
    - **User-User CF:** the system calculates the similarity between users based on their historical interactions with items. The steps involved are as follows:
        1.– **Calculate user similarity:** We first compute the similarity between users, for example, using the Pearson correlation. This method quantifies how similarly two users rate the same items.

$$\text{sim}(u, v) = \frac{\sum_{i \in I_{uv}} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I_{uv}} (r_{v,i} - \bar{r}_v)^2}} \tag{2.2}$$

        where $r_{u,i}$ is the rating of user u for item i, and $\bar{r}_u$ is the average rating given by user u.

        2.– **Identify items rated by both users:** After calculating the similarity score between two users, we identify items that have been rated by both users, which helps us evaluate their level of agreement.

        3.– **Make predictions:** Based on the similarity computed, recommendations can be made by taking the weighted sum of ratings from similar users. The higher the similarity, the more influence a user's rating will have on the prediction.

        To generate recommendations, it is necessary to select "neighboring" users. Different strategies can be used to select these neighbors:

        - **Selecting only a few users (Threshold approach):** Users above a certain similarity threshold are selected.
        - **Selecting a fixed number of neighbors (Top-N approach):** The top N most similar users are selected based on their similarity values.
        - **Using clustering to select neighbors:** Users are clustered into groups, and recommendations are made using users within the same cluster.
    - **Item-item CF:** works in a similar manner, but the focus is on the similarity between items rather than users. Here, the system computes the similarity between items based on the ratings provided by users. Once similarities between items are calculated, the system recommends items that are most similar to those that the user has previously rated highly.
- **Model-based CF:** These techniques build predictive models of user-item interactions by learning underlying patterns from historical data. These models typically transform users and items into a shared latent factor space, capturing hidden features that explain observed interactions. Once trained, model-based systems can instantly generate recommendations by predicting a user's preference for hidden items, without having to scan the entire dataset at query time [11].

These are some of the challenges we may face when using CF:

- **Data sparsity:** Due to the massive number of items online and the limited number of ratings provided by users, the user-item interaction matrix becomes extremely sparse. This makes it difficult to find meaningful similarities between users or items, leading to weak recommendations.
- **Cold-start problem:** A specific case of sparsity, cold-start happens when new users or new items enter the

system. Since there is little to no historical data available, generating accurate recommendations becomes problematic.

- **Scalability:** As the number of users and items increases rapidly, traditional collaborative filtering algorithms struggle to process such large datasets efficiently, demanding significant computational resources [12].

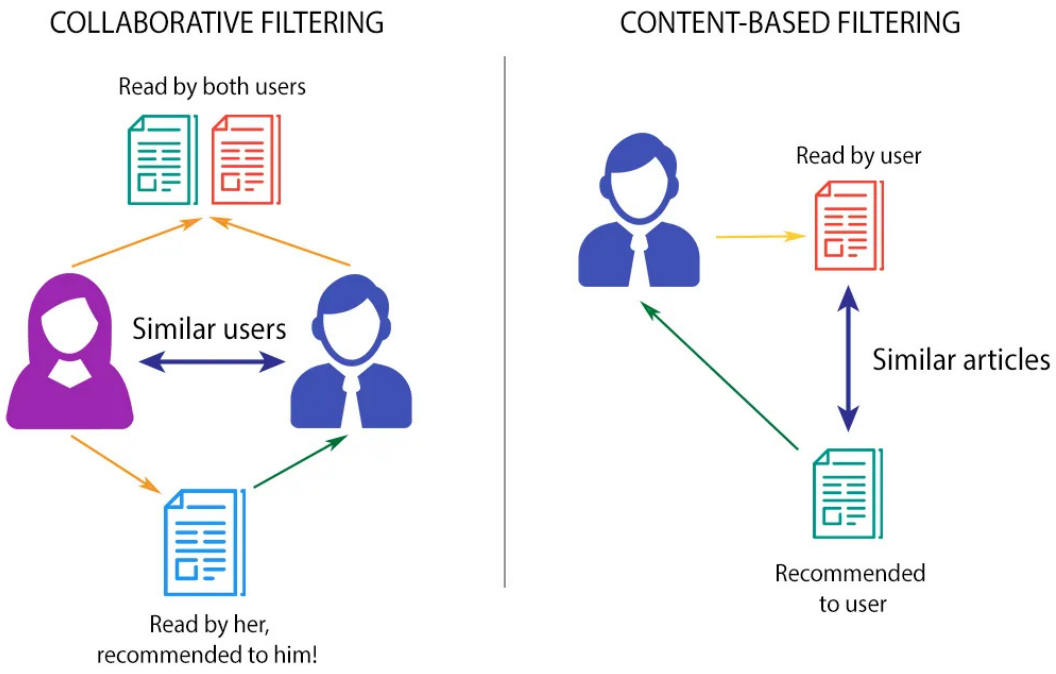


Figure 2.1: Comparison between Collaborative Filtering and Content-Based Filtering [13].

2.2 Commercial Book Systems

In this section, we focus on analyzing commercial book recommender systems that are currently available on the market. Unlike research-oriented models or academic proposals, these platforms are actively used by readers and give actual examples of how recommender systems are implemented at scale. We examine how they collect and use user data, the strategies they apply to personalize recommendations, and the limitations they face.

This study does not aim to provide an exhaustive review, but rather focuses on the analysis of a selection of popular and widely recognized websites.

2.2.1 Goodreads

Launched in January 2007, *Goodreads* has established itself as one of the most popular platforms for readers around the world. With a database that includes 20 billion data points [14], its recommender system stands out as one of its most valued features.

In 2011, *Goodreads* acquired *Discovereads.com*, a company specializing in book recommendation algorithms. This acquisition allowed them to leverage their vast database, which at the time included 100 million book ratings from 4.6 million users, to identify reading patterns and offer excellent personalized recommendations [15].

In order to initiate personalized recommendations, *Goodreads* requests that users rate a minimum of 20 books. This process helps mitigate the “cold start" problem. By obtaining these initial ratings, the system gathers the necessary data to better understand the user’s preferences, ensuring that the recommendations are more relevant and personalized from the start.

Goodreads uses a system of ‘bookshelves,’ or virtual shelves, which allows users to organize their books by genres, interests or custom categories. These shelves function as tags or filters that help book categorization. For example, when a user adds a book to a shelf like ‘Historical Fiction’, it is associated with that subgenre, reflecting the reader’s preferences. Thanks to this feature, *Goodreads* collects information about each user’s tastes and can generate more personalized recommendations. If a user has several books classified under ‘Historical Fiction’ and has rated these titles positively, the system will suggest other books that other users have tagged in the same category and also have good ratings.

In this way, the *Goodreads* system combines the two main approaches presented before: collaborative filtering by comparing the ratings and preferences of a user with those of other readers who have similar tastes, and content-based filtering by analyzing the characteristics of those books, such as genre, author, and tags, to offer suggestions aligned with the user’s interests. In addition to its main recommender system, *Goodreads* offers other options for discovering books, such as classification by genres, lists of users’ favorite books by year and genre, recommendations of newly released books, and lists created by users themselves.

2.2.2 Casa del Libro

Being one of the biggest bookstore chains in Spain [16], *Casa del Libro* could not fall behind in the field of book recommendations.

While both *Casa del Libro* and *Goodreads* use hybrid recommender systems, they have significantly different approaches. In addition to filtering by genre, this platform incorporates other recommendation methods designed to diversify the user experience.

One of the main recommendation methods is the ‘Bestsellers’ list [17], which suggests famous books based on their popularity on the platform. This list is organized by genres, making it easier for users to find relevant titles based on their interests.

Another key tool is the “Recommended" section [18], which selects books based on user interactions

such as clicks, purchases, comments, ratings and searches, providing a more personalized experience.

A particularly interesting system is the “Author Recommendations" [19], where various authors suggest books they have read and consider valuable. This approach adds a unique and human touch to the recommender system, allowing readers to discover works recommended by recognized figures in the literary world.

Since it is a commercial platform, user purchasing behavior can alter recommendations. For example, a user might purchase books as gifts for others, which could influence the data used to generate future suggestions, without necessarily reflecting the buyer’s preferences.

Another critical point is that some of the ratings available on the platform are more related to the shopping experience than to the quality of the book itself (see Figure 2.2). This can influence the recommendations, reducing their usefulness for readers seeking genuine opinions about the titles.

This also occurs in other systems such as *Amazon* or *Fnac*, as they are platforms for selling books and other products. In these environments, the reviews that books receive do not always refer exclusively to their literary content, but often focus on logistical aspects or the overall shopping experience, such as the package condition or the speed of delivery.

This can become a problem when generating recommendations based on ratings, such as ‘favorites’, ‘top rated’, or ‘most popular’, since a high rating does not necessarily mean that the user enjoyed the book’s content, such as the plot, characters or the author’s style, but rather that the book arrived in good condition, the delivery was fast, or the overall purchase experience was positive.

One possible solution to this problem would be to apply Natural Language Processing (NLP) techniques to the reviews, in order to identify and separate those referring to the actual content of the book from those focused on external factors. In this way, the resulting ratings and recommendations would be much more accurate and truly reflect the literary quality of each work.

Figure 2.2: Book rating comments, collected from the website *Casa del Libro* [20].

2.3 Data Collection for Book Recommendation

For a recommender system to function accurately, it is essential to have a complete and up-to-date database. Although some public datasets are available, such as those from *Goodreads* [21] or *Amazon* [22], the most recent among them is at least five years old, making them unsuitable for the objective of this project of using current data. The *Book-Crossing* dataset [23] was also reviewed; however, it was ultimately discarded due to its age and lack of content in the desired language (Spanish).

An alternative approach considered was the use of public APIs, which offer the advantage of retrieving structured data directly from servers, thus eliminating the need for web scraping. The *Goodreads* API, in particular, would have been a valuable resource. However, it has not been publicly available since December 2020 [24].

It was decided to build a custom dataset using automated information extraction as no suitable and up-to-date data source was found in an accessible format. Therefore, web scraping and web crawling techniques were used to collect essential information about books, including titles, authors, genres, languages, and publishers.

2.3.1 Web crawling

Crawling is a process where a web robot, also known as a crawler, spider, or bot, systematically navigates the web, exploring, and analyzing the content of web pages. This process begins with the selection of an initial URL, from which the crawler extracts information and detects links to other pages. As it progresses, it follows these links, crossing the website structure and collecting data automatically [25].

A key concept in crawling is crawl depth, which determines how deep within a website's hierarchy the crawler will go. For example, with a crawl depth of level 1, the crawler will only scan the main pages. If the crawl depth is level 2, the crawler will also include pages linked from the main pages, etc. The deeper the crawl, the more pages of the site will be analyzed, allowing for a more comprehensive data collection. However, a greater depth also significantly increases the time and resources required for the process.

2.3.2 Web scraping

This process consists of extracting the HTML content from web pages to filter the required information and store it. Once the web crawl is performed, the content is identified and extracted. Finally, the extracted content must be cleaned and formatted. The information is then post-processed and stored in structured data files [26].

Python libraries such as Selenium, Beautiful Soup, and Requests make this task easier.

- **Selenium:** This package is used to automate the Python web browser interaction. It requires a driver to interface with the chosen browser. For example, Firefox requires geckodriver, while Chrome requires chromedriver [27].
- **Beautiful Soup:** This library simplifies the scraping of information from web pages. It sits above an HTML or XML parser, providing Pythonic idioms for iterating, searching, and modifying the parse tree [28].
- **Requests:** This library allows you to send HTTP requests easily. There is no need to manually add query strings to your URLs or to encode your PUT and POST data manually [29].

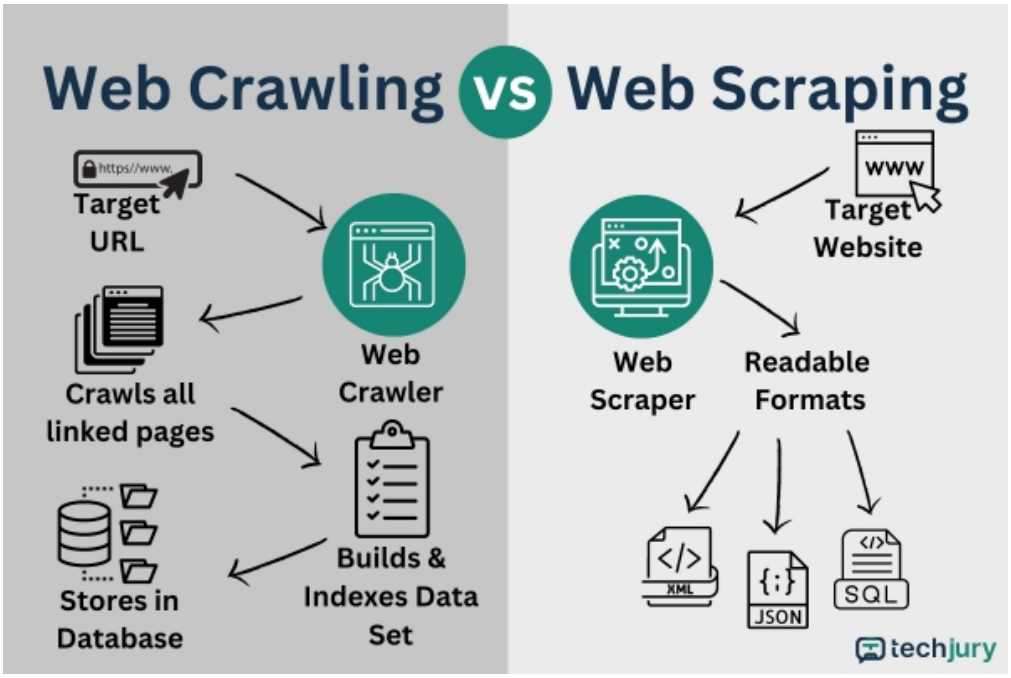


Figure 2.3: Comparison between web crawling and web scraping [30].

2.4 Web Application Frameworks

One of the most important aspects of building an online recommender system is developing a web application that allows users to interact efficiently with the platform. To ensure a smooth and optimized experience, it is essential to structure the application into three main components:

- **Backend:** Is the logical core of a web application, responsible for managing business logic, processing data, and ensuring the proper functioning of the system. Its main function is to receive, process and return information to applications and websites, facilitating navigation and ensuring the integrity, security and efficiency of various functionalities [31].
- **Frontend:** Is the part of web development (or an application) that surrounds everything the user sees and interacts with. It serves as the connection between the application and the user, allowing smooth communication and an intuitive experience [32].
- **Database:** Is not only responsible for storing information but also for organizing it and establishing logical connections between the data. Its role is to efficiently manage, process and provide information, ensuring that both the backend and the frontend can access it as quickly as possible. This allows for a smooth user experience and optimal data handling within the system.

The following section outlines the technologies used in the development of the application, specifically the use of Nuxt as the frontend framework and Django as the backend framework, including their integration with the database.

2.4.1 Django framework

Django is a high-level Python web framework designed to promote fast development and a clean, practical approach to web application design [33]. It manages the underlying code for web applications, handling the request-response system through the Model-View-Template (MVT) architecture [34].

In Django, **models** act as the interface between the database and the server code, serving as the definitive source of information about the application's data. They represent the logical structure behind the entire application and are responsible for maintaining and managing the data [35].

Each model is generally assigned to a single database table, and its attributes correspond to the fields of that table. Through object-relational mapping (ORM), Django transforms database tables into classes and objects in Python code, allowing interaction with the data in an intuitive and structured way. Django supports relational databases such as MySQL, PostgreSQL, among others, facilitating efficient data management [36].

Views are responsible for processing user requests and generating appropriate responses. They handle the logic that connects models to the user interface (HTML, CSS, JavaScript). Each view in Django is a function or class that receives a request and returns a response. To manage requests, Django uses a URL mapper that maps each URL to its corresponding view function, allowing users to access different content based on the web address they visit.

A **template** is a file that defines the presentation of a web page in the browser. It combines static HTML parts with special syntax that allows dynamic content to be inserted flexibly.

In addition to Django, frameworks like Flask (a lightweight Python microframework for simpler or custom applications [38]) and Express.js (a minimalist Node.js framework for rapid development [39]) are also popular. However, Django was chosen due to its "batteries included" approach, which provides built-in tools for core functions such as user authentication, ORM and security (CSRF, XSS protection) [40]. Unlike Express.js, which requires manual setup for many components, Django enables the developer to focus more on the recommendation logic than on configuring the infrastructure, making it ideal for personalized applications. In addition, since Django was covered in our coursework, I was able to use my previous knowledge and apply best practices from the beginning.

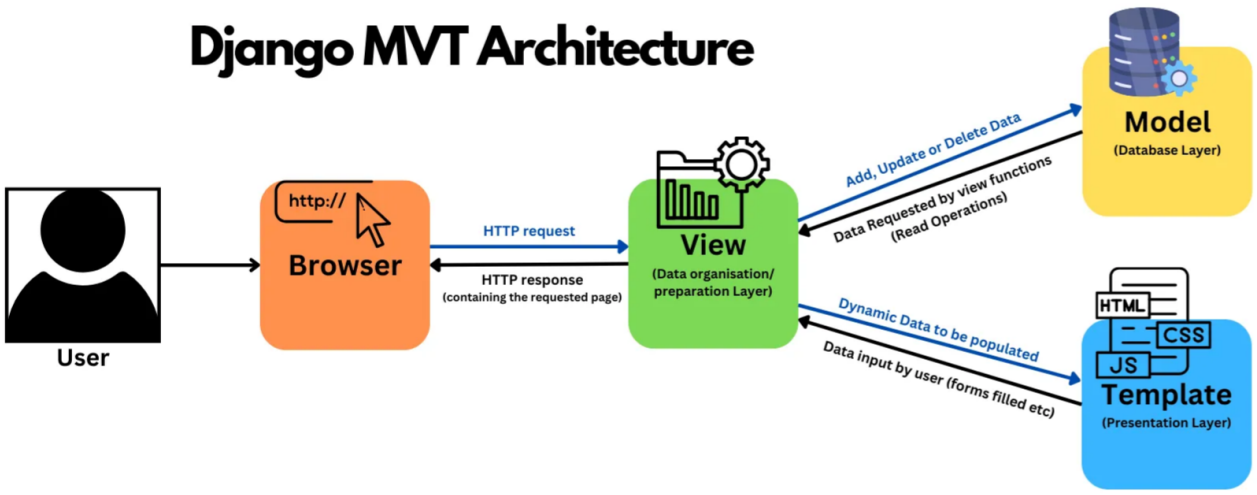


Figure 2.4: MTV Architecture Diagram [37].

2.4.2 Nuxt framework

Nuxt.js is an open-source framework based on JavaScript, designed to simplify and update the development of applications with Vue.js. It provides an optimized structure, as well as features like the ability to combine JavaScript, HTML and CSS on the client side, as well as integration with Vuex for efficient state management of the application [41].

Nuxt.js enhances the performance of Vue.js by simplifying server-side rendering (SSR), eliminating the need for complex configurations. It also allows for static site generation, leveraging the benefits of SSR without its operational overhead [42].

In addition, it optimizes performance through code splitting, loading only the necessary packages on the client side to reduce the application size. Its modular architecture facilitates code reuse, and its auto-import system automates the management of components [43].

The reason Nuxt.js was chosen for this project is that, since I was already familiar with Vue.js (having studied it previously), this framework provides a comprehensive solution for building optimized applications with SSR and SSG, allowing for more efficient work than using Vue.js alone [44]. Moreover, Nuxt.js greatly simplifies the configuration process and provides a ready-to-use structure, unlike Next.js (a React framework), which requires a part of the infrastructure to be configured manually [45]. Moreover, Nuxt.js's modular approach and automatic code splitting feature not only enhance performance but also contribute to long-term maintainability and scalability of the application.

3

Design and Implementation

This chapter describes the functional and non-functional requirements, as well as the decisions made regarding the design and development of the implementation. The source code of the application is available in the following repository: `git@github.com:palomaaruiiiz/TFG.git`.

3.1 Project Structure

The project is structured into three modules, as shown in Figure 3.1:

Data Processing: This module is responsible for scraping data from *Casa del Libro*, preprocessing it, and calculating similarities based on that information. It consists of the following submodules:

- **Dataset Collection:** Composed of two parts:
  - **Crawler:** It manages the retrieval of book URLs. It ensures that the collected URLs are valid and correspond to actual books.
  - **Scraper:** It collects the required data from the previously obtained URLs. It sends requests to the server to gather all the necessary information to create the dataset.
- **Data Pre-processing:** This submodule is responsible for cleaning the data and removing books that lack essential information (such as author, cover, synopsis, etc.).
- **Similarity Processing:** This submodule calculates the similarity between all books based on their genres, authors, language, publisher and synopsis.

Web Application Backend: This module contains the functions that define the internal logic of the application, as well as the database system.

- **Database:** This submodule is responsible for storing and managing the data in the dataset. All data is inserted and organized in a way that keeps it interconnected (books, authors, publishers, languages, etc.), allowing for more efficient and faster access to the information.
- **Recommendation algorithms:** This submodule includes the recommendation algorithms that use the previously computed similarities to generate the top recommended books.

Web Application Frontend: This is an interactive platform that visually displays the results obtained from the web application backend. In addition to showing personalized suggestions, the application provides a user-friendly interface where users can explore recommended books, view details about each title, and adjust their preferences to receive more accurate recommendations aligned with their interests.

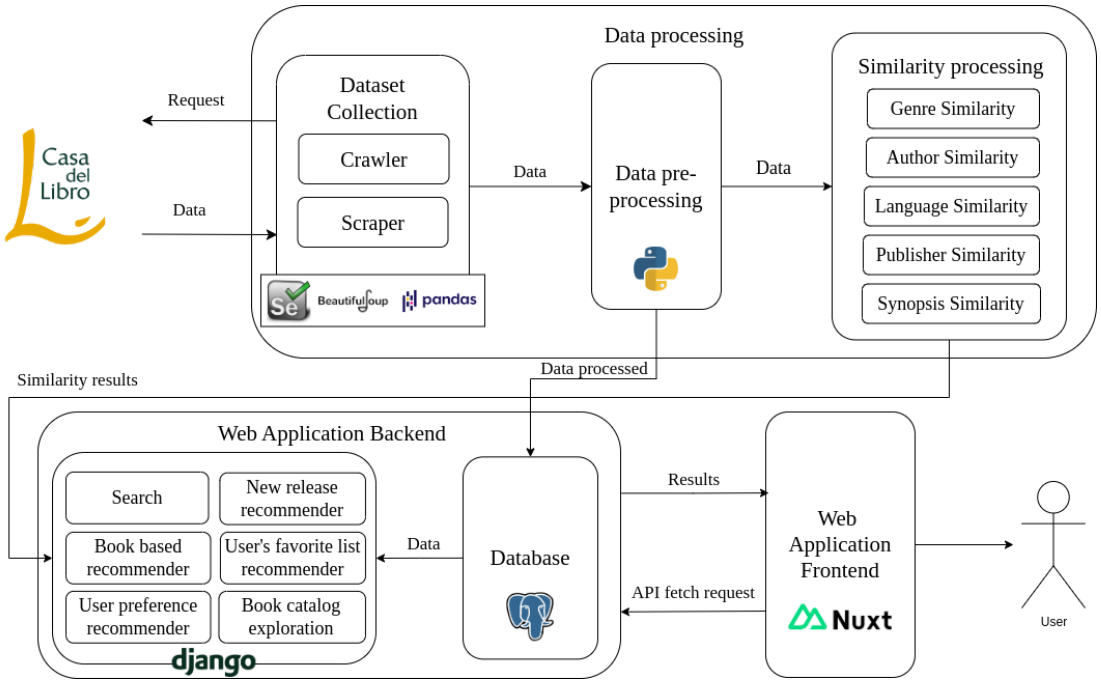


Figure 3.1: Dataflow of the project, organized by modules

3.2 Requirements Analysis

3.2.1 Functional requirements

Data Preprocessing

FR-DP-1.– The data must be sourced from real, reliable platforms, specifically from *Casa del Libro*'s website.

FR-DP-2.– The dataset will be collected via automated crawling and scraping.

FR-DP-3.– The dataset must allow adding new books anytime to stay up to date.

FR-DP-4.– The dataset must include key details such as title, author(s), publisher, language, price, page count, genre(s), ISBN, publication date and the book's synopsis, among other relevant information.

FR-DP-5.– The system must provide mechanisms to identify and remove duplicate entries, to ensure the dataset remains clean and accurate.

FR-DP-6.– The system must compute the similarity between books using the following criteria: author, publisher, genres, language and synopsis.

FR-DP-7.– Book similarities must be calculated in advance and stored (e.g., in files) for efficient use during recommendations.

FR-DP-8.– The system must use language models (such as Sentence-BERT) to generate vector representations of the synopses and apply cosine similarity to measure semantic similarity.

FR-DP-9.– The system must compute similarity values for categorical attributes as follows:

- **FR-DP-9.1.–** For author, language, and publisher, the similarity is binary (1 if they match, 0 otherwise).
- **FR-DP-9.2.–** For genres, the similarity must be computed approximately through text comparison and common genre relationships.

FR-DP-10.– Similarity calculations must be performed concurrently or in parallel, using threads or background

processes to improve preprocessing performance.

FR-DP-11.– The system must allow recalculating and updating similarities periodically, especially when new books are added to the database.

FR-DP-12.– The system must combine the different similarity scores to generate recommendations for books similar to a given one or to a user's favorite list.

Web Application Backend

FR-WAB-1.– The database will store a separate table for each entity in the dataset: books, authors, publishers, genres and languages. Each table will include its corresponding unique identifiers and associated names.

FR-WAB-2.– Each book in the database will be associated with one or more authors, establishing a many-to-many relationship between books and authors.

FR-WAB-3.– Each book in the database will be associated with one or more genres, establishing a many-to-many relationship between books and genres.

FR-WAB-4.– Each book in the database will be associated with a single publisher, establishing a relationship of many-to-one between books and publishers.

FR-WAB-5.– Each book in the database will be associated with a single language, establishing a many-to-one relationship between books and languages.

FR-WAB-6.– The database must allow efficient book searching using filters such as title, author, genre, language, price, among others.

Web Application Frontend

FR-WAF-1.– The application will allow users to register by providing a name and a password.

FR-WAF-2.– The system will allow users to log in using their credentials.

FR-WAF-3.– The system will allow users to select a preferred language and multiple genres. Based on these preferences, personalized book recommendations will be given.

FR-WAF-4.– The system will also recommend newly released books to users.

FR-WAF-5.– The system will allow users to view all the information of each book stored in the database, including title, author, genre, synopsis, number of pages, year of publication, publisher, language and price.

FR-WAF-6.– Each book detail page will include a recommender that displays books similar to the selected one.

FR-WAF-7.– The system will allow users to filter the books using the following criteria:

- **FR-WAF-7.1.–** Title (A-Z and Z-A)
- **FR-WAF-7.2.–** Price (low to high, high to low and custom price range)
- **FR-WAF-7.3.–** Year of publication (most recent or oldest)
- **FR-WAF-7.4.–** Genre
- **FR-WAF-7.5.–** Language
- **FR-WAF-7.6.–** Publisher

FR-WAF-8.– The system will include a search bar that allows users to search for books by title, author or ISBN.

FR-WAF-9.– Authenticated users can add or remove books from their favorites list.

FR-WAF-10.– Based on the books in the user's favorites list, the system will recommend similar books.

FR-WAF-11.– The system will allow users to log out.

3.2.2 Non-functional requirements

Data preprocessing

NFR-DP-1.– The process of creating and updating the dataset must be fully automated, ensuring that data collection through scraping and crawling techniques is performed without manual intervention.

NFR-DP-2.– The system must be scalable, allowing the dataset to expand with new books without compromising performance or data integrity as the volume of information grows.

NFR-DP-3.– The system must have mechanisms to handle missing or incomplete data efficiently, such as the ability to discard books that lack essential information (author, ISBN, genres, etc.).

Web application backend

NFR-WAB-1.– The database must be optimized for performance, ensuring fast query response times even as the dataset grows in size.

NFR-WAB-2.– The database should support efficient indexing to speed up data searches and retrieval.

NFR-WAB-3.– Recommendation generation must respond within 1 second under typical load, by leveraging parallelism and avoiding redundant computations.

NFR-WAB-4.– The system must support efficient similarity processing and recovery even as the number of books increases significantly.

NFR-WAB-5.– The system must organize similarity calculations and storage logic in a modular way, allowing easy adjustment of weights or similarity logic for each criterion.

NFR-WAB-6.– In case of missing or corrupted similarity files, the system must fall back to basic content-based recommendations (e.g., by genre or author).

NFR-WAB-7.– The system should respond to user requests (e.g., filtering, searching, recommendation) in under 2 seconds under normal load.

NFR-WAB-8.– The system must be scalable, allowing an increasing number of books without impacting the performance.

Web application frontend

NFR-WAF-1.– The interface should be intuitive and easy to navigate, providing a user-friendly experience for all users.

NFR-WAF-2.– The frontend must be responsive, ensuring compatibility and optimal display across different devices and screen sizes, including desktops, tablets and smartphones.

NFR-WAF-3.– The user interface must load and render quickly, aiming for initial page load times under 3 seconds on average connections.

3.3 Design

This section will explain the decisions made in the different modules of the application related to its design. In addition, it will describe how the different components in each module are connected, supporting the explanation with diagrams and images.

3.3.1 Data preprocessing

To start building the recommender system, it is important to create a complete dataset with all the necessary information about the items, in this case, books. As mentioned in Section 3.2.1, a custom data collection process was used to make sure that the dataset is as up to date as possible. Also, it is important to have a clear and efficient design for this process, as it simplifies the later development of the database.

The data collection process has been divided into three main submodules: **dataset collection**, **data preprocessing**, and **similarity processing**. We will begin by describing the **dataset collection** submodule, which consists of two parts: the **crawler** and the **scraper**.

The **crawler** was designed to identify only URLs that contain information about books. Since a large number of URLs are irrelevant to the objectives of this project, a filtering mechanism was implemented to drop irrelevant pages. In addition, the crawler can discover links nested within other pages. This is especially important given the typical structure of websites like *Casa del Libro*.

Meanwhile, the **scraper** was designed with a focus on efficiency, extracting all the necessary information from each item in the shortest time possible. Special attention was given to addressing the challenges posed by the dynamic and complex structure of the website, ensuring accurate data extraction and minimizing potential errors throughout the process. The operation of these two parts is presented in Figure 3.2.

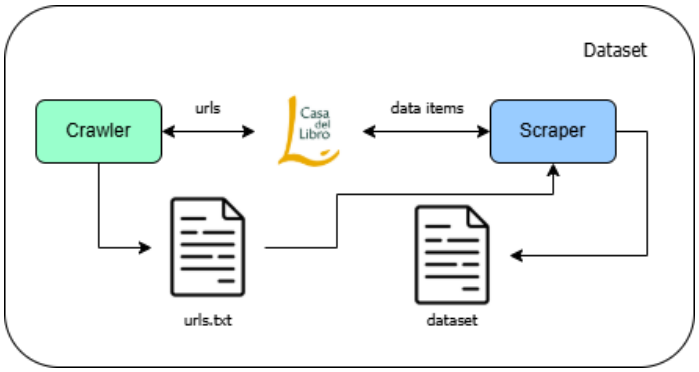


Figure 3.2: Dataset collection design

The **data preprocessing** submodule prepares the raw data collected by the dataset collection submodule for insertion into the database. It reads a CSV file and processes each book entry.

This step makes sure that only valid and structured data is stored, which enables accurate similarity processing in the next module. Finally, the **similarity processing** submodule calculates how similar books are to each other using different criteria: authors, genres, language, publisher and synopsis.

The system follows a modular design. Depending on the chosen criterion it uses a specific function to compare books. For example, author similarity checks if books share the same authors, while synopsis similarity uses sentence embeddings to compare the meaning of their descriptions.

The results are saved in text files, where each line shows the similarity between a pair of books.

3.3.2 Web application backend

To fully understand the backend design of the web application, it is essential to first examine the database structure where all the collected dataset information is stored. This basis supports the backend's core functions by organizing data efficiently and defining the relationships between entities such as books, authors, genres, publishers, languages and users.

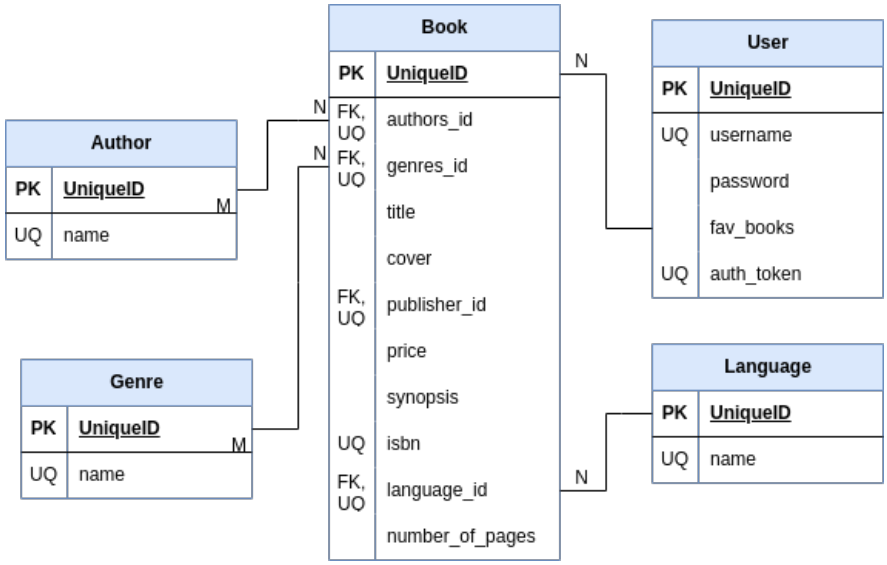


Figure 3.3: The database entity-relationship diagram, where PK, FK, and UQ stand for Primary Key, Foreign Key, and Unique, respectively, illustrates the structure and relationships of the data.

As shown in Figure 3.3, the extracted data from the dataset is stored in related tables (Book, Author, Genre, Publisher, and Language). In addition, we have the User table, which stores the information provided by users during registration (such as the username and password). Moreover, it keeps a list of books that the user marks as favorites while using the application.

Building on this data foundation, the backend manages data storage, recommendation algorithms, and API endpoints that serve data to the frontend. This includes:

- Managing requests for book details, filtered lists and search results.
- Handling book search requests by ISBN, author or title.
- Computing personalized recommendations through various parts:
  - Book-based similarity using multiple attributes weighted for relevance.
  - Filtering of user preferences based on selected language and genres.
  - New release sorted by publication date.
  - Favorite-based recommendation combining multiple similarity scores.
- Optimizing queries to efficiently retrieve related books, authors and genres.
- Returning serialized data including detailed similarity scores for recommendations.

Each submodule is designed to support the frontend's functionalities while ensuring performance and scalability.

3.3.3 Web application frontend

The *Coverless* application's frontend focuses on providing a user-friendly interface that allows users to explore books and access personalized recommendations easily.

The **book catalog exploration** submodule enables users to browse the entire collection of over 5,000 books. Users can access a general list of books and use filters or search tools to find specific titles, authors or ISBNs. When a book is selected, the frontend requests detailed information from the backend and displays a full page with relevant details such as title, author, synopsis, publisher, language, genre and price. Moreover, this page shows a list of other books written by the same author and recommends books related to the selected title.

The **search** submodule provides a user-friendly search bar where users can type queries to quickly find books by title, author or ISBN. As the user types, the system dynamically shows relevant suggestions under the input field.

The **book-based recommender** submodule provides personalized suggestions by allowing users to pick a book and then displaying similar titles. Similarity is measured based on multiple attributes including author, genre, synopsis, publisher and language. This feature helps users discover new books closely related to their interests.

The **user preferences recommender** provides recommendations based on the user's preferred reading languages and genres. After users specify their preferences, the frontend sends requests to the backend to receive filtered and ranked lists of books that match those criteria.

To highlight recent content, the **new releases recommender** displays the 12 newest books on the homepage. Each book preview includes a cover image, title and author, all linked to the book's detailed information page. This module allows users to quickly find the latest additions without needing to apply filters or perform searches.

Finally, the **user's favorite list recommender** lets logged-in users add books to their favorites list. The frontend retrieves and shows personalized recommendations based on these favorites, helping users find new titles aligned with their tastes. The functioning of this recommender system is shown in the sequence diagram (Figure 3.4).

Together, these modules provide a complete and interactive frontend experience, making it simple and enjoyable to explore and discover books.

3.4 Implementation

This section will provide a detailed explanation of the implementation of the modules described in Section 3.3. In addition, the implementation decisions made throughout the process (related to the code, libraries, etc.) will be discussed.

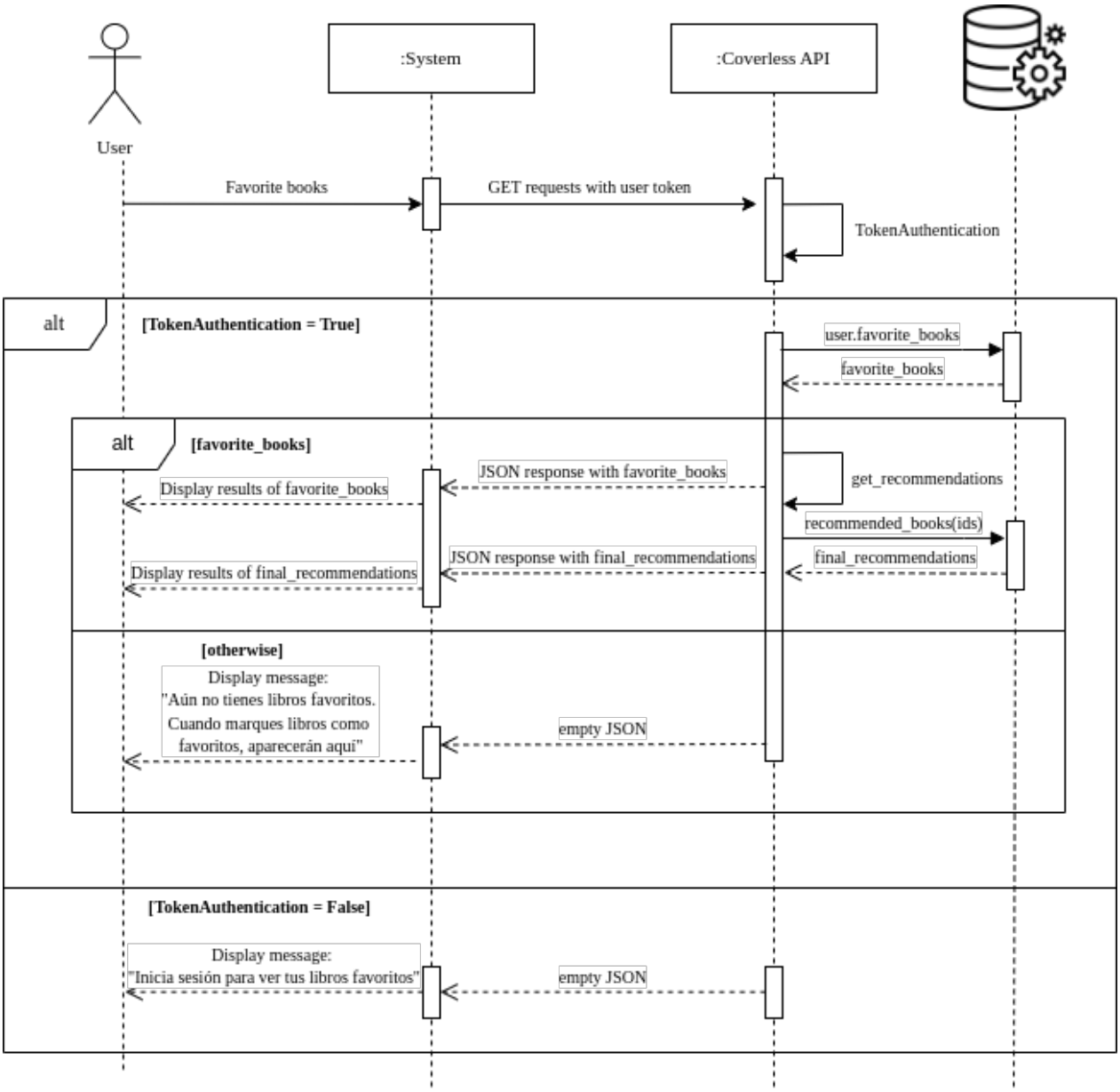


Figure 3.4: Sequence diagram: User's favorite book list and recommended books

3.4.1 Data preprocessing

As mentioned in Section 3.3.1, the data collection process consists of three main submodules, whose implementation will be discussed in detail below.

Dataset collection

This section explains the implementation of the dataset collection module, which is responsible for collecting raw book data directly from the source website. It consists of two parts: the crawler, which identifies and collects URLs of book detail pages, and the scraper, which visits those URLs to extract structured information.

The **crawler** systematically navigates the paginated sections of the website, starting from a base URL and increasing the page number to collect individual book page URLs.

- **HTML fetching and parsing:** The crawler sends an HTTP GET request using the `requests` library for each page URL. It randomizes the `User-Agent` header with each request to reduce the risk of being marked as a bot. The HTML response is parsed with `BeautifulSoup` to build a structured representation of the page.
- **URL identification:** The crawler uses a regular expression pattern created to identify book URLs by matching the URL structure used in *Casa del Libro* (`'/libro[\w-]+/\d{13}/\d+$'`) and validate the existence of a 13-digit ISBN, which uniquely identifies books. This method filters out irrelevant links and improves accuracy.
- **Avoiding duplicates and ensuring persistence:** Valid book URLs are stored in a list to avoid duplicates during crawling. Furthermore, each URL found is saved to an external text file, enabling data persistence and allowing the process to resume without issues if interrupted.
- **Controlled pagination handling:** The crawler generates URLs for subsequent pages by increasing the page number in the URL path. It continues this process until reaching either the maximum number of URLs desired or a fixed page limit, avoiding endless crawling loops.
- **Randomized delay between requests:** To limit server load and mimic natural browsing behavior, the crawler inserts random sleep intervals between requests within a configurable range. This method reduces the risk of being blocked and follows best practices for responsible scraping.

This module generates a structured list of book URLs, which the **scraper** then processes to extract data from dynamic book detail pages.

- **Dynamic rendering and synchronization:** The scraper uses waits, such as `WebDriverWait` combined with `expected_conditions`, to make sure that important page elements are completely loaded before extracting data. This guarantees access to asynchronously loaded content, such as book descriptions or updated prices.
- **User-Agent randomization and headless mode switching:** The scraper randomly selects user-agent headers for each session and changes between headless and visible browser modes to reduce detection risk. This helps simulate real user browsing and avoid anti-scraping mechanisms.
- **Element locating approach:** Data extraction is based on a combination of CSS selectors and XPath queries, with exception handling for cases such as `NoSuchElementException` and `TimeoutException`. This allows the scraper to handle missing or changed elements without crashing.
- **Handling interactive content:** The scraper detects sections with expandable text (such as truncated summaries) and clicks on “See more” buttons to reveal and collect the full content.

- **Data extraction focus:**
  - *Visual content:* Downloads the book cover image and converts it to a Base64 string for easy storage within tabular data.
  - *Metadata:* Extracts important text fields such as title, author(s), publisher, ISBN, language, edition year, number of pages and genre tags.
  - *Price information:* Collects the current price, handling different formats and currency symbols.
- **Incremental data saving:** Extracted data is stored in a `pandas.DataFrame` in memory and saved to a CSV file on disk. This approach prevents data loss if the process is interrupted.

This setup enables reliable and accurate extraction of book data from complex, dynamically generated pages.

Data preprocessing

A custom module was developed to load information from a CSV file into the corresponding Django models to populate the system's database with the collected book data. The script, implemented in Python using the Django ORM and the pandas library, performs several steps to ensure data integrity, normalization and consistency.

The module begins by executing Django's migrate command to ensure that the database schema is up to date. Then, it checks for the existence of the specified CSV file path. If the file is found, the data is loaded into a pandas.DataFrame, and column names are standardized to match the expected model field names.

To avoid issues during import, rows with missing values in critical fields, such as Title, Author, ISBN, Genre, Language, Synopsis or Publisher, are dropped. The Price field is also cleaned by removing currency symbols and converting the values to numeric format, ensuring compatibility with the database schema. Similarly, the number of pages and year of publication are cast to integers after handling invalid entries by converting them to null values, which are then replaced with zeros.

For each row in the dataset:

- If a book with the same ISBN already exists, it is deleted to avoid duplication.
- Related entities such as `Publisher` and `Language` are created or retrieved using `get_or_create`.
- A new `Book` instance is created with the cleaned data.
- The `Author` and `Genre` fields, which may contain multiple values separated by commas or conjunctions, are split and normalized. Each related object is also created or fetched using `get_or_create`, and associated with the book using the corresponding many-to-many relations.

Similarity processing

This module implements a system to calculate the similarity between books based on different criteria such as authors, language, genres, publisher and synopsis. It is designed as a custom Django

management command, enabling its execution via the command line.

The implementation uses the `handle` method to track the execution depending on the chosen criterion. Each criterion corresponds to a specific similarity function, which is then passed to a general method named `calcular_similitud_libros`. This method iterates through all pairs of books and calculates the corresponding similarity, storing the results in a text file named according to the similarity model.

The module uses Python's `ThreadPoolExecutor` from the `concurrent.futures` module to improve performance when processing a large number of books. Each thread is responsible for processing one book against all others, allowing multiple threads to run in parallel. This multithreading strategy optimizes computation time and efficiently uses available CPU resources.

Code 3.1: Code to preload embeddings for book synopsis

```
1  def precargar_embeddings(self, libros):
2      """Preload embeddings for all book summaries."""
3      self.model = SentenceTransformer(
4          'paraphrase-multilingual-MiniLM-L12-v2',
5          device='cuda' if torch.cuda.is_available() else 'cpu'
6      )
7      sinopsis = [str(libro.sinopsis) if libro.sinopsis else "" for libro in libros]
8      self.embeddings = self.model.encode(sinopsis, show_progress_bar=True, batch_size=32,
           convert_to_numpy=True)
9      self.libro_idx_map = {libro.id: idx for idx, libro in enumerate(libros)}
```

When the selected criterion is `synopsis`, the system uses a pre-trained language model called `paraphrase-multilingual-MiniLM-L12-v2` from the `SentenceTransformer` library to transform book synopses into numerical vectors (embeddings) as shown in Code 3.1. Then the system computes the cosine similarity between these vectors using the `cosine_similarity` function from `scikit-learn`. To make this process more efficient, it also calculates the embeddings for all synopses and stores them in memory before the comparison starts. An index map is also created to retrieve the book embedding using its id.

Other criteria use simpler logic. For example, the **author similarity** function returns 1 if both books have the same authors, 0.5 if they share some authors, and 0 otherwise. The similarity functions for **language and publisher** are binary: they return 1 if the corresponding attribute is the same and 0 otherwise. For **genre similarity**, the system computes the average of the maximum similarity between each genre in one book and the genres of the other. This similarity is calculated using the `SequenceMatcher` from the `difflib` library, and the computation is split into two threads to process both directions at the same time.

The similarity scores are written to a file inside a directory. Each line in the file contains a JSON dictionary with the IDs of the two books and the computed similarity value.

Code 3.2: Code to calculate the similarity between book genres

```
def calcular_similitud_generos(self, libro1, libro2):
    """Calculates average similarity between genres of two books."""
    genres1, genres2 = libro1.genres.all(), libro2.genres.all()
    if not genres1 or not genres2:
        return 0.0

    def max_sim(g, others):
        return max(0.5 *SequenceMatcher(None, g.name, o.name).ratio() +
                   0.5 *SequenceMatcher(None, o.name, g.name).ratio()
                   for o in others)

    sims1 = [max_sim(g1, genres2) for g1 in genres1]
    sims2 = [max_sim(g2, genres1) for g2 in genres2]

    return (sum(sims1) + sum(sims2)) / (len(sims1) + len(sims2))
```

3.4.2 Web application backend

The Django framework has been chosen as the backend technology for the development of the application. Django uses the Model-Template-View (MTV) pattern, designed to separate logic, data, and presentation while simplifying development. Its default features such as an ORM (Object-Relational Mapper), admin panel and strong authentication system make it especially suitable for data-centric applications like recommendation systems.

The backend of the application is structured into three main modules: the **database**, the **API endpoints** and the **API views**, as described in Section 3.3.2. For the **database** layer, PostgreSQL was selected due to its strong scalability features, although it requires a more complex setup compared to other database engines. The relationships between entities were carefully designed and implemented to optimize query performance. In addition, unnecessary data retrieval was avoided by limiting the amount of column data returned in each query, preventing database overload and improving response times.

To ensure that the application is accessible from any device and not limited to local execution, it was **deployed using Neon**, a cloud-based PostgreSQL platform [46]. Neon was chosen over alternatives like Render [47] or Supabase [48] due to its usability, seamless database upload process and lack of expiration limits, making it a reliable long-term hosting solution.

The **API endpoints** were developed using Django REST Framework to handle all backend interactions with the frontend. These endpoints include main functionalities such as user registration, authentication (token-based login and token refresh) and retrieval of the current user's data. Users can also manage their favorite books through endpoints that allow adding, removing and checking if a specific book is marked as a favorite.

A personalized book recommendation endpoint was implemented, which computes suggestions based on the user's favorite books by combining similarity metrics: synopsis, genres and authors. These recommendations are filtered using a weighted scoring system. Also, fallback logic is included to ensure results even when similarity data is insufficient.

The API also includes a search endpoint that enables users to find books by title, author or ISBN. In addition, a paginated listing endpoint supports sorting and filtering by price range. Together, these endpoints form a comprehensive and efficient interface between the frontend and the recommendation engine.

The **API views** implement the logic behind these endpoints and are designed to be modular and reusable.

- **User Views:** Handle registration, login, logout and profile retrieval. Token-based authentication is implemented using JWT tokens, for secure and stateless sessions. Permissions are required to limit certain actions to authenticated users only.
- **Book Views:** Include listing, detail retrieval and search capabilities. Pagination and filtering mechanisms are implemented to allow efficient browsing by price, genre, author and other attributes. Filtering is done at the database level using query parameters to reduce data transferred and improve response times.
- **Favorites Views:** Allow users to add or remove books from their favorites list and check if a book is already marked as favorite. These views validate ownership and implement access controls to make sure that users interact only with their own data.
- **Recommendation View:** Implements a complex logic combining multiple similarity metrics (such as text similarity of synopses using embedding models, genre overlap and author matches) with a weighted scoring system to generate personalized book recommendations. The view also incorporates fallback strategies to guarantee recommendations even when user data is limited.

The **API views** are carefully implemented focusing on maintainability and scalability. Serialization ensures that only selected fields are included in API responses, implementing a consistent and controlled data structure. By limiting the data sent to clients, this approach improves efficiency and prevents potential leaks of sensitive or unrelated details.

3.4.3 Web application frontend

For the frontend interface, the **Nuxt.js** framework was chosen due to its powerful ecosystem and flexibility. Built on top of Vue.js, Nuxt provides advanced features such as server-side rendering (SSR), static site generation (SSG) and automatic routing provided by default. These features contribute to improved performance and a better user experience, especially important for applications such as book recommendation systems.

To make the frontend accessible from any device, the application was **deployed on Render**. It is connected to the PostgreSQL database hosted on Neon, as explained in Section 3.4.2, ensuring full integration between frontend and backend.

The project makes use of the modular architecture of Nuxt 3, which simplifies the integration of various technologies and tools, making it easier to develop and maintain.

- **Server-Side Rendering (SSR):** Implemented to optimize performance on first load and support effective search engine indexing.
- **State Management with Pinia:** Pinia, the official state management library for Vue 3 [49], is integrated via the Nuxt Pinia module to efficiently manage application state such as user authentication status, favorite books and UI state.
- **Styling with Tailwind CSS:** Tailwind CSS [50] is used to style the UI with functional classes that support fast and responsive design.
- **Authentication Handling:** The *auth-init.client.ts* plugin handles client-side authentication state, ensuring smooth integration with the backend's JWT token mechanism.
- **HTTP Client Configuration:** The application communicates with Django REST API endpoints using the native Fetch API, configured with runtime environment variables for secure and efficient requests.
- **Image Optimization:** The @nuxt/image module is used to provide optimized image loading and responsive image support.

Overall, the combination of Nuxt.js, Pinia, Tailwind CSS and optimized API integration results in a efficient, scalable and user-friendly frontend application that complements the backend architecture.

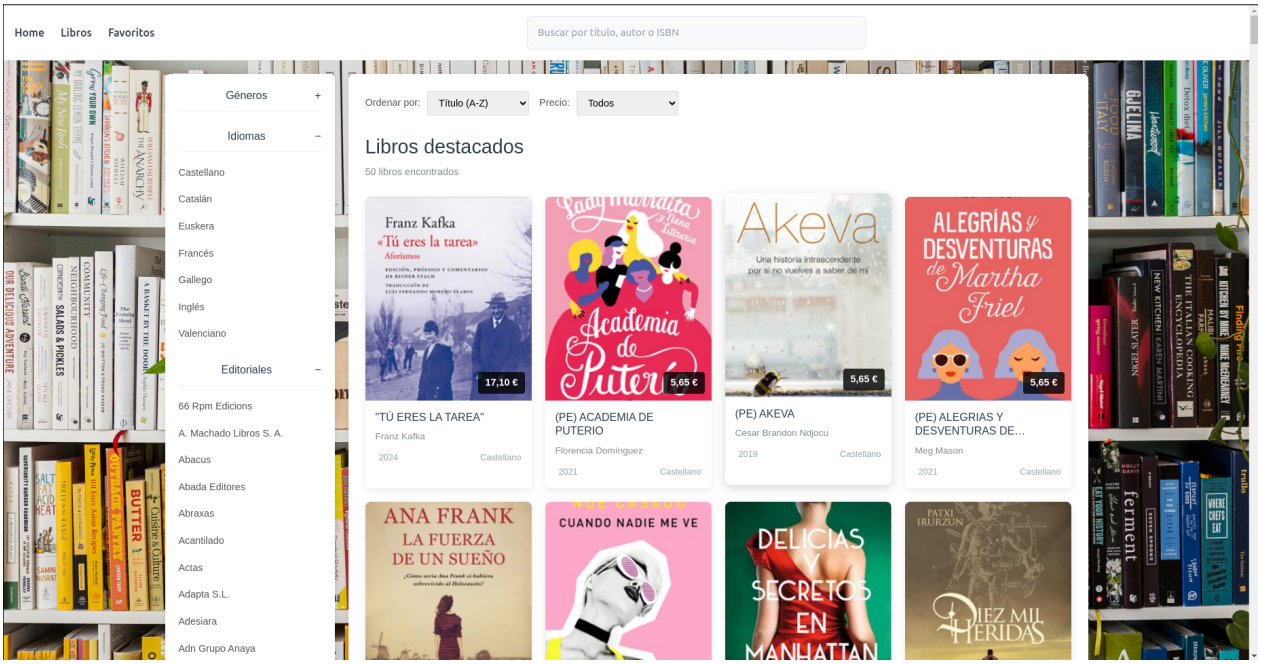


Figure 3.5: Homepage of Coverless

4

Experiments and Results

This chapter presents the results of the experiments conducted on the application. First, a brief analysis of the data collected from *Casa del Libro* will be provided. Next, we will discuss the results obtained from evaluating Coverless with real users.

4.1 Data Analysis

To begin the data analysis of the books extracted from *Casa del Libro*, we note a total of 5,124 entries. From this dataset, we examine key characteristics, starting with a comparison by **genre**, as shown in Figure 4.1. Genres with fewer than five books have been excluded to improve clarity and readability.

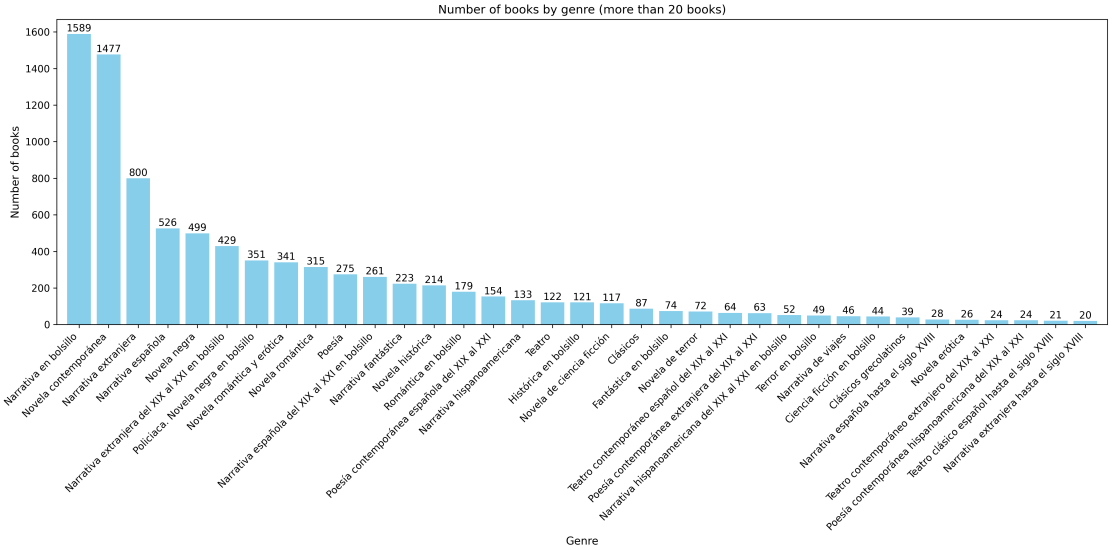


Figure 4.1: Number of books by genre

The graph indicates that *narrativa de bolsillo* is the most frequent genre, which is reasonable since it includes all pocket-sized editions, regardless of their specific content. Moreover, there is a notable predominance of foreign narrative over national, a difference that proves to be particularly significant.

Figure 4.2 shows the distribution of the **number of books published per year**. It is obvious that the greater part of the publications are recent, reflecting a focus on current titles. Furthermore, preliminary

data for 2025 suggest that this year may exceed the number of publications from the previous year, despite not yet being complete.

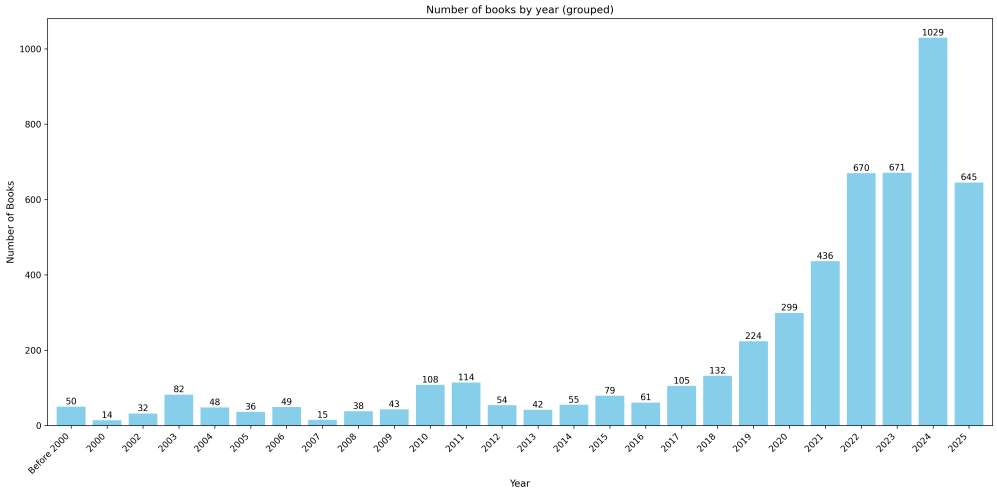


Figure 4.2: Number of books by year

The analysis has also been extended to include the **average prices** of the books over time. It is important to note that this graph reflects both physical and digital (e-book) editions. The data has been grouped by publication year, allowing us to observe trends in book pricing from 2010 to 2025.

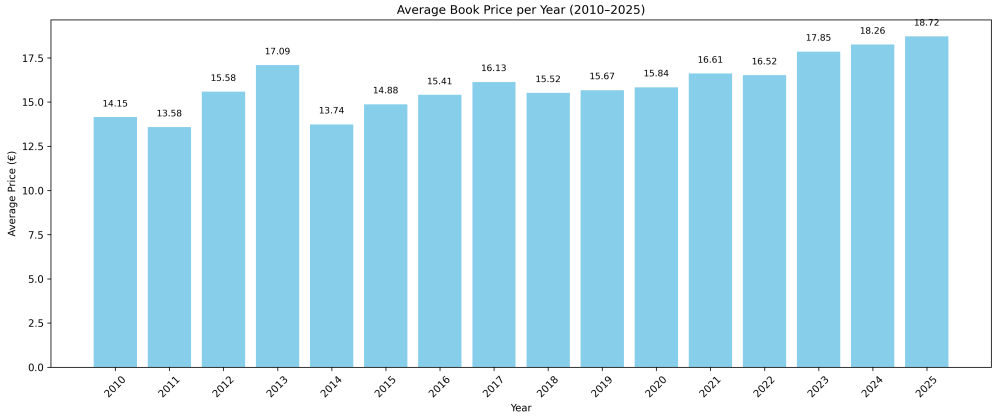


Figure 4.3: Number of books by price range

Following this, we analyze Figure 4.4, which compares the number of books across different **publishers**. To improve the clarity of the graph, publishers with fewer than 20 books have been excluded.

Debolsillo stands out as the publisher with the highest number of titles in the dataset as shown in Figure 4.4. This publisher belongs to the *Penguin Random House* [51] group, as does *Alfaguara*, which ranks fourth in terms of the number of publications. On the other hand, *Booket*, also known for its pocket-sized editions and part of the *Planeta* group [2], holds the second position, highlighting the competition between these two major publishing groups within this market segment.

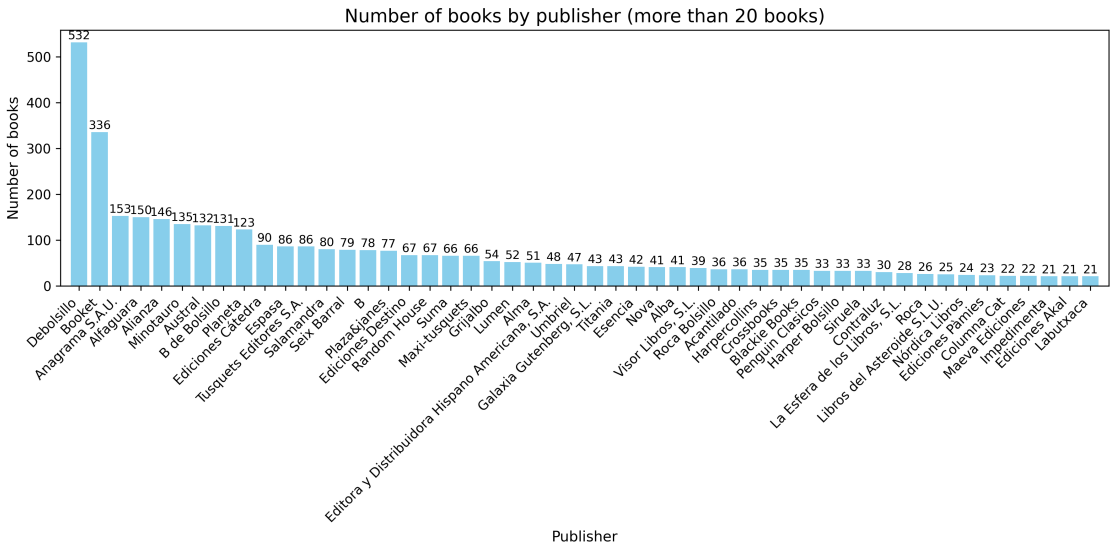


Figure 4.4: Number of books by publisher

4.2 User Experience and Feedback Analysis

This section describes the study carried out to collect user feedback on the *Coverless* application. A total of 19 Spanish-speaking participants were invited to test the platform and share their opinions, which is why the survey was conducted in Spanish. Participants were chosen from diverse backgrounds, with different levels of technical knowledge, interests and reading habits. This diversity was intended to ensure a more representative and realistic range of feedback.

The participants were given full freedom to explore and test the application on different types of devices, including both computers and mobile phones. They could access it from anywhere by visiting the URL: `https://coverless-app.onrender.com/`. To gather and organize their responses, a Google Forms survey was created. The form focused on key aspects such as usability, user experience, and reading habits. In addition, some personal data, including name, age and gender, was collected, always with the participants' informed consent.

In the following subsections, the results obtained from the survey will be analyzed in detail.

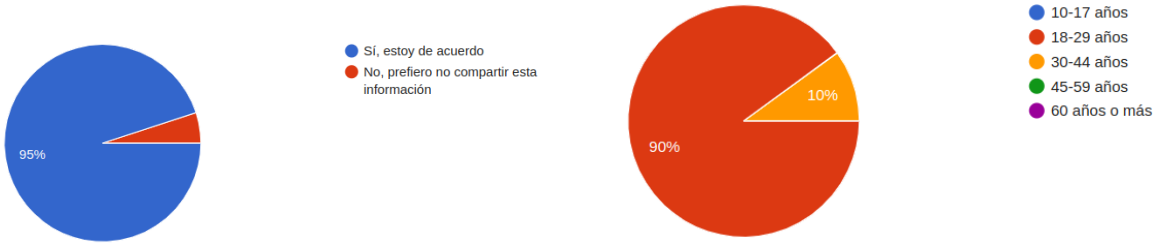
4.2.1 User demographics and reading preferences

User consent

Before starting the survey, participants were required to give their consent for the processing of their personal data, as mentioned above. Obviously, those who did not give their consent were not allowed to complete the form. See Figure 4.5(a).

Age of the Participants

Most Coverless testers were aged 18–29 (90%), with the remaining 10% in the 30–44 range. It was not possible to include older users, despite attempts. However, Coverless remains accessible and intuitive for users of all ages. See Figure 4.5(b).



(a) User consent (b) Age of the participants

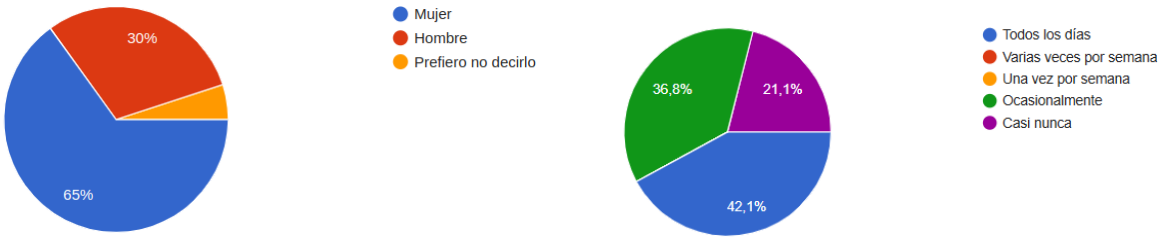
Figure 4.5: User consent and participant ages

Gender of the Participants

In this case, the gender distribution of the participants was relatively balanced, with approximately 30% identifying as male and 65% as female. The remaining 5% preferred not to tell their gender. See Figure 4.6(a).

Frequency of reading

They were asked about the frequency with which they read to have an idea of their reading habits. A total of 42.1% reported reading daily, 36.8% stated that they read occasionally and the remaining 21.1% indicated that they rarely read. See Figure 4.6(b).



(a) Gender of the participants (b) User reading frequency

Figure 4.6: Participant demographics and reading habits

Users favorite genres

Users were asked to indicate their favorite literary genres in order to have an idea of their preferences. The most popular genre was Fantasy, receiving 13 votes, followed by Romance with 10 votes and Thriller/Suspense/Mystery with 9 votes. It is important to note that participants were allowed to select

multiple genres, rather than being limited to a single choice.

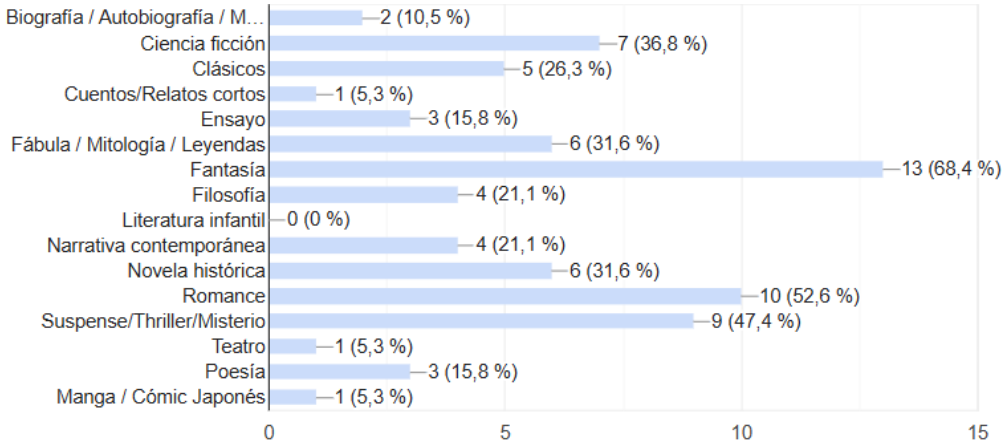


Figure 4.7: Users favorite genres

User strategies for book discovery

Users were asked about the methods they usually use to discover new books. The most frequently selected option was through social media platforms (such as Instagram, TikTok, YouTube, etc.), which received 15 votes. The second most common method was to receive recommendations from friends or family members. Lastly, the third most selected option involved following book influencers, such as BookTubers, Bookstagrammers and BookTokers, individuals who regularly share book reviews and literary content online.

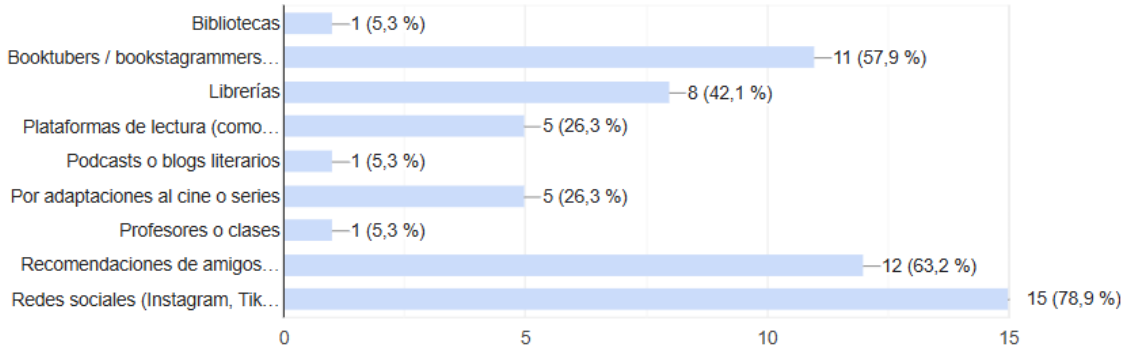


Figure 4.8: User strategies for book discovery

Users criteria for book selection

Users were asked to indicate which aspects they consider most important when selecting a book. The option most selected was the theme or genre, with 17 votes. This was followed by recommendations they had received, with 14 votes, indicating that a significant portion of users rely on suggestions from others. The third most voted factor was the synopsis, with 12 responses. These results are shown in Figure 4.9.

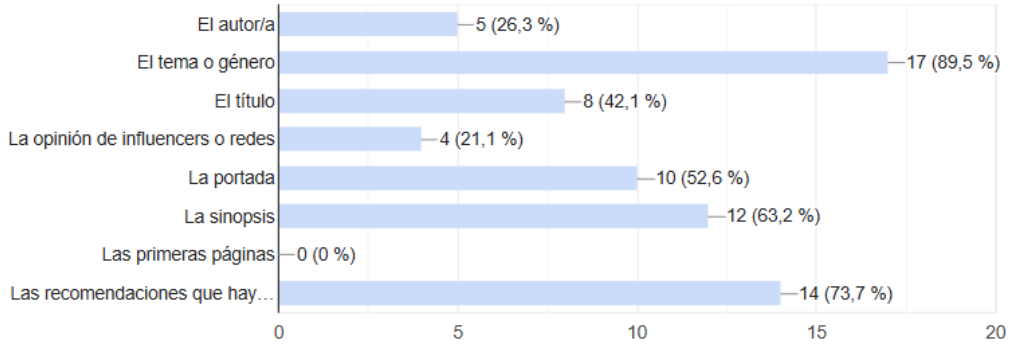


Figure 4.9: Users criteria for book selection

Book-related applications used by readers

As shown in Figure 4.10, participants were asked if they use any applications related to books (e.g., platforms that allow them to keep track of books they have read, receive recommendations from other users or based on content, etc.). The most selected option was Goodreads, with 10 votes. Interestingly, the second most common response, with 9 votes, was not using any application at all.

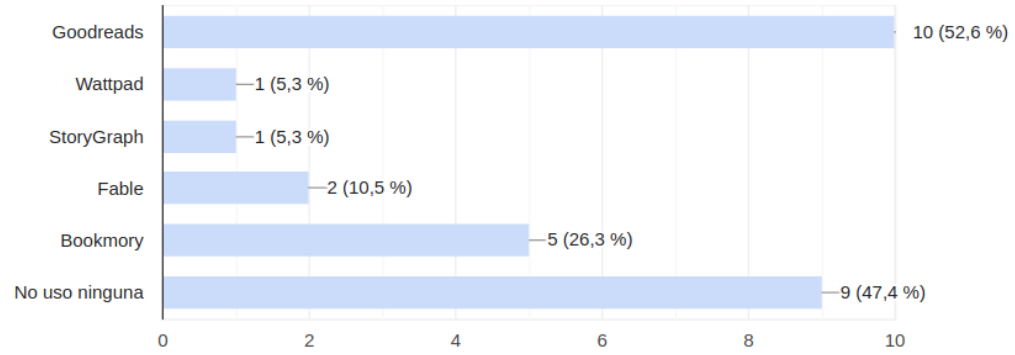


Figure 4.10: Book-related applications used by readers

Relevance of reviews in book selection

As shown in Figure 4.9, recommendations play an important role in the users selection process of books. For this reason, participants were asked whether they are influenced by reviews or critiques before choosing a book. A total of 31.6% reported that they are influenced by reviews, 36.8% stated that it depends, 15.8% indicated that they are only influenced when they are uncertain about a particular book, and the remaining 15.8% claimed not to be influenced at all. These results, presented in Figure 4.11, confirm that reviews indeed play a fundamental role in the decision-making process when selecting a book.

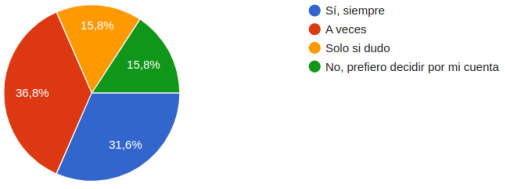


Figure 4.11: Relevance of reviews in book selection

4.2.2 Evaluation results

Once all the necessary data about the users' literary background has been collected, we can continue to evaluate their feedback on Coverless.

User registration experience

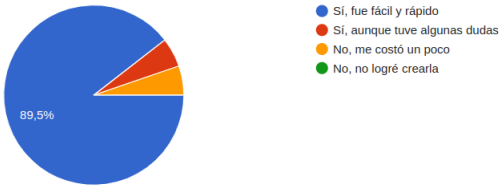


Figure 4.12: User registration experience

To evaluate the user registration process in the application, participants were asked a series of questions. First, they were asked if they were able to create an account without any problems. As shown in Figure 4.12, 89.5% of users stated that the process was quick and simple. Another 5.3% mentioned experiencing some doubts but managed to complete the registration successfully, while the remaining 5.3% found the process slightly challenging.

After, users were asked to rate the registration process on a scale from 1 to 5, based on their overall experience. The average rating obtained was an impressive 4.84 out of 5.

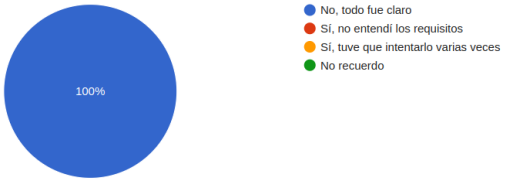


Figure 4.13: User ratings on password selection difficulty

Finally, participants were asked whether they faced any difficulties when choosing a password. The password requirements were: a minimum of eight characters, at least one uppercase letter, one lowercase letter, one number and one special character. All users (100%) confirmed that they had no issues with this step and found the instructions clear and easy to follow (Figure 4.13). The register page is located in Section A.1 of the Appendix.

User login experience

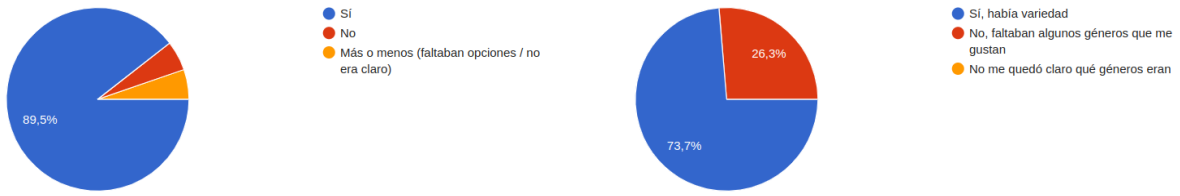
Figure 4.14: User login experience

To evaluate the login process, users were asked if they were able to successfully log in on their first attempt. A total of 89.5% responded affirmatively, while 10.5% indicated that they were able to log in but needed to correct an error before succeeding. See Figure 4.14.

In addition, participants were asked to rate the login process on a scale from 1 to 5. The final average rating was 4.74. The login page is in Section A.2 of the Appendix.

Preference-based recommender system

To evaluate the first recommender system, where users are asked to indicate their preferred reading language and one or more genres of interest, (the first page of the recommender is in Section A.3 of the Appendix) a series of questions were presented.



(a) User feedback on language selection experience

(b) User feedback on genre selection experience

Figure 4.15: User feedback on language and genre selection

The first question asked if users were able to successfully select their preferred language. A total of 89.5% responded affirmatively, 5.3% reported difficulties, and the remaining 5.3% indicated that there were not enough language options available. See Figure 4.15(a) and Section A.4 of the Appendix.

The second question evaluated if the list of genres available for selection was enough. Here, 73.7% of the users responded positively, while 26.3% felt that some genres were missing. See Figure 4.15(b) and Section A.5 of Appendix.

The final question regarding this recommender system asked users if the recommendations they received were correct based on their selections. While 42.1% responded positively, 57.9% indicated that some recommendations were accurate, while others were not. See Figure 4.16(a) and Section A.6 of the Appendix.

In addition, users were asked if they discovered any books that might interest them through this recommender. A total of 52.6% answered yes, 42.1% responded that they would need to read the book first, and 5.3% answered no. See Figure 4.16(b).

(a) Quality of recommendations based on language and genre preferences

(b) Discovery of interesting books through the recommender

Figure 4.16: User feedback on the recommendation system

To conclude the evaluation of this feature, participants were asked to rate the recommender system on a scale from 1 to 5. The final average rating was 4.21.

Book recommender for new releases

The new book recommendation system appears on the same screen as the personalized recommender, as shown in Section A.7 of the Appendix. To collect user feedback, participants were asked about their impressions of these recommendations. A total of 52.6% found the suggestions very interesting, 42.1% considered them interesting but not relevant to their personal preferences, and 5.3% found them of little relevance. See Figure 4.17.

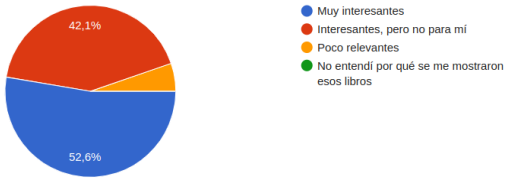


Figure 4.17: User opinions on the new releases recommender

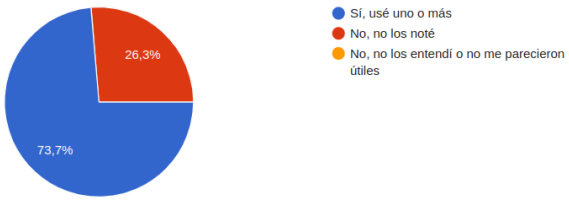
This recommender focuses on displaying the most recently published books, regardless of the user's preferred genres or languages. For this reason, it is understandable that some users may not find the suggestions completely relevant to their tastes. To conclude the evaluation of this recommender, participants were asked to rate it on a scale from 1 to 5. The final average rating was 3.89.

Book section

To evaluate this part of the application, users were first asked if they found it easy to navigate this screen. As shown in Figure 4.18, 100% of them responded positively, stating that it is clear and neat.

Figure 4.18: User experience navigating the book section

Users were asked about the sidebar filters (genre, language, and publisher). A total of 73.7% used them, while 26.3% did not notice them (Figure 4.19(a)).



(a) Sidebar filters usage (Genre, language and publisher)

(b) Sorting options usage (Title, year and price)

Figure 4.19: User feedback on the book section filters and sorting options

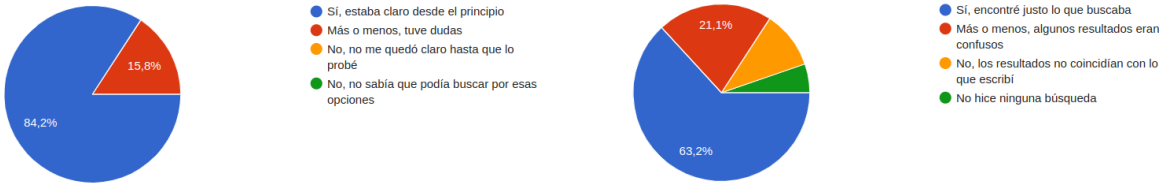
They were also asked about the sorting filters (by title, price or year). 68.4% said they worked correctly, 15.8% said they did not notice them, and the remaining 15.8% said they were not interested in sorting the books (Figure 4.19(b)). These filters were rated on a scale from 1 to 5, resulting in an average score of 4.26.

Finally, users were asked if they had any difficulty finding the books they were interested in. 68.4% responded that they found various interesting books, 15.8% said they found some but had to search in depth, and the remaining 15.8% said they did not use this section much. See Figure 4.20.

Figure 4.20: Ability to find relevant books in the book section

Search bar evaluation

To properly evaluate the Coverless search bar, users were asked a series of questions (the design of the search bar can be seen in Section A.8 of the Appendix).



(a) User familiarity with the search bar operation

(b) Relevance of the search results

Figure 4.21: User feedback on the search functionality

The first question asked if users clearly understood that it was possible to search by author, title or ISBN. A total of 84.2% responded affirmatively, indicating that it was very clear, while 15.8% reported having some doubts (as shown in Figure 4.21(a)).

The second question asked if users could easily find the books they were looking for using the search bar. 63.2% responded that they found exactly what they were looking for, 21.1% stated that some results were confusing, 10.5% said the results did not match their input, and the remaining 5.3% indicated that they did not carry out any search at all (see Figure 4.21(b) for the results).

To conclude the evaluation of the search bar, users were asked to rate its precision on a scale from 1 to 5. The final score was 4.37.

Book information

To evaluate the book details screen (which can be seen in Section A.9 of the Appendix), users were asked the following question: *Did you find the book information clear and sufficient?* A total of 94.7% of users responded affirmatively, while the remaining 5.3% indicated that some information was missing (see Figure 4.22).

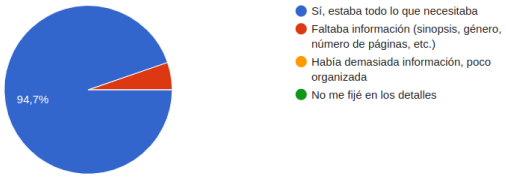
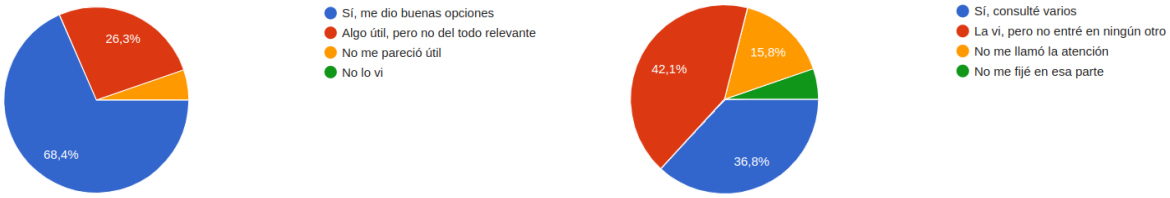


Figure 4.22: Clarity and sufficiency of book information

Users were also asked about the two other most important elements on this page: the similar books recommender and the list of books by the same author.

To evaluate the first element, users were asked if they found the similar books recommender useful. 68.4% responded that it was useful and provided good suggestions, 26.3% said it was useful but not entirely relevant, and the remaining 5.3% indicated that they did not find it useful. See Figure 4.23(a).

In addition, users were asked if they found the list of other books by the same author interesting. 36.8% answered positively, 42.1% said they noticed the list but did not click on any other book, 15.8% reported that it did not catch their attention, and the remaining 5.3% stated that they did not notice that section (see Figure 4.23(b)).



(a) Utility of the similar book recommender

(b) Utility of the same-author book list

Figure 4.23: User feedback on similar and same-author book suggestions

Finally, users were asked to rate the overall book details page on a scale from 1 to 5. The final score for this page was 4.47.

Favorite list

To evaluate the final element of Coverless, users were asked the following questions: *Could you easily access your list of favorite books?* To this question, 94.7% of users responded that it was easy to find, while 5.3% reported that it took them a little longer to find it (see Figure 4.24).

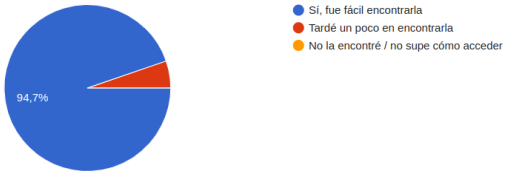
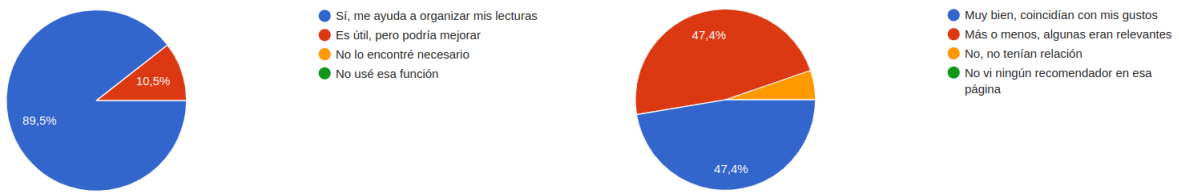


Figure 4.24: Access to user's favorite list

The next question was: *Did you find it useful to manage your favorites list (adding or removing books)?* 89.5% of users indicated that it helped them organize their reading, while the remaining 10.5% found it useful but suggested it could be improved (see Figure 4.25(a) and Section A.10 of the Appendix).

Finally, users were asked: *How well do the recommendations based on your favorites match your preferences?* 47.4% answered that the recommendations matched very well, another 47.4% stated that some recommendations were relevant and 5.3% responded that they were not related (see Figure 4.25(b)). To conclude, users were asked to rate the favorites page on a scale from 1 to 5, resulting in a final score of 4.58.



(a) Utility of favorite list management

(b) Recommendations based on favorite books graph

Figure 4.25: User feedback on favorite book features

App Section	Score (out of 5)
Registration process	4.88
Login process	4.47
Language and genres recommender system	4.21
New releases	3.89
Utility of filters	4.26
Search bar	4.37
Book information page	4.47
Favorite page	4.58

Table 4.1: User ratings for all the parts of the application

4.2.3 Usability assessment with the SUS

The System Usability Scale (SUS) produces a single combined score representing the overall usability of a system. Individual item scores, which range from 0 to 4, are combined using a specific calculation method: for odd-numbered items, the contribution is the scale position minus one; for even-numbered items, it is five minus the scale position. The total is then multiplied by 2.5 to give a final SUS score ranging from 0 to 100 [52].

Each response from the 19 users was processed according to the standard SUS methodology to obtain the final score. The resulting values were used to calculate the average and standard deviation, offering a summary of the system's usability. The combined results from all participants show an **average SUS score of 93.75** with a standard deviation of approximately 5.30. This high average score indicates that users found the system highly usable and intuitive, with very consistent feedback reflected by the relatively low variation in scores.

Based on the individual ratings provided for each section (summarized in Table 4.1) and the overall SUS result, we can conclude that the tests were successful and that users responded positively to the application. In addition to rating the app, users also provided suggestions for improvement. Some of these, such as adding more language options and a dedicated page for search results, are already implemented in the current version of Coverless. Others, while not yet integrated, have been added to the list of future improvements due to their valuable insights.

5 Conclusions and Future Work

This chapter presents the conclusions from the work carried out, along with possible future steps for the continuation or improvement of this project.

5.1 Conclusions

The aim of this work is to support readers in selecting their next read by offering books they may not have faced otherwise. In addition, it also helps publishers increase the visibility of their books.

To achieve this, **Coverless** was created, a personalized book recommendation application. Different recommendation systems were implemented in the application to provide variety and diversity in the books presented to users. The first recommender suggests books **based on the language and genres specified by the user**, allowing users to receive recommendations adapted to their preferences. Secondly, the **book-based recommender** enables users to view books that are very similar to a selected title. Lastly, there is a recommender based on each **user's favorite books**. Furthermore, the application includes a "New Releases" list, displaying recently published books, along with multiple filters designed to make the search for the perfect book faster and more convenient.

All of these systems, along with the entire platform, including the database, the web scraper and crawler used to collect the data, the recommendation algorithms, and the web application itself, were implemented entirely from scratch.

During the Coverless testing phase, users were able to evaluate the precision of the recommenders, the extensive database and the user-friendly interface. Users could access the application from any device, including computers, tablets or mobile phones, simply by visiting the URL [1]: `https://coverless-app.onrender.com/`.

It is important to note that this database was ethically obtained from *Casa del Libro*, a publicly accessible website. As this project is designed for research and educational purposes, this approach

[1] Note that, since we use a free hosting option, Render spins down the instance after some time without activity, so it may require some minutes to be up again when visiting this URL.

was considered appropriate. For a professional system, it would be reasonable to consider establishing a partnership with *Casa del Libro* to legally use their database.

5.2 Future Work

Certain limitations were faced during the development of this Bachelor's thesis, as well as various future steps that could improve the whole system.

Expanding the database: One of the main limitations was the limited storage space available. The database had to be hosted on an external server in order to allow real users to test the application, which resulted in significant storage limitations. Increasing the volume of data would improve the performance and accuracy of the recommendation system.

Automating data collection: To keep the system up to date, automating the collection of newly released books would be ideal. This would enable users to access the most current information available without requiring manual updates.

Email-based login and authentication: Integrating an API that allows users to authenticate using only their email address would improve the user experience and streamline access to the platform.

Implementing a chatbot: A chatbot could be integrated into the system as a future improvement. This would allow users to request recommendations conversationally. By simply providing a reference book, a generative AI model could search the database for titles with similar descriptions. This would make the recommendation process faster, more accurate and personalized, while also offering a dynamic and interactive experience, including book related questions or specific queries about the recommended books.

Implementing a read books list: This feature would help users keeping track of the books they have read. In addition, a recommendation system similar to the one for the favorite books list could be implemented, allowing users to receive suggestions for books similar to those they have read.

Adding more filters for better organization in the application: The favorite books list could be sorted by genres to better categorize the books. This would also help make the recommendations provided in this section even more accurate.

Adding and evaluating recommendation algorithms: Expanding the online user study and incorporating more algorithms would allow for a more thorough evaluation of their performance. This process will help refine the system based on real user feedback, closing the recommender system development circle.

Bibliography

[1] S. Pascua Vicente, "España: El Ministerio de Cultura publica los datos de edición de libros en el año 2024." Instituto Autor, Apr. 2025. Accedido el 14 de junio de 2025.

[2] Grupo Planeta, "Libros." https://planeta.es/es/libros, 2025. Last accessed: May 2025.

[3] Agapea, "Novedades en libros. junio de 2025." https://www.agapea.com/Novedades-de-junio-de-2025-lnd14145.htm, 2025. Last accessed: May 2025.

[4] NVIDIA Corporation, "What is a recommendation system?." https://www.nvidia.com/en-us/glossary/recommendation-system/, 2024. Last accessed: February 2025.

[5] F. Isinkaye, Y. Folajimi, and B. Ojokoh, "Recommendation systems: Principles, methods and evaluation," *Egyptian Informatics Journal*, vol. 16, no. 3, pp. 261–273, 2015.

[6] SmartPanel, "La inteligencia artificial en los sistemas de recomendación." https://www.smartpanel.com/como-funciona-la-inteligencia-artificial-en-los-sistemas-d Dec. 2023. Last accessed: May 2025.

[7] O. Kaššák, M. Kompan, and M. Bieliková, "Personalized hybrid recommendation for group of users: Top-n multimedia recommender," *Information Processing & Management*, vol. 51, no. 6, pp. 791–806, 2015.

[8] A. Bellogín and P. Sánchez, "Collaborative filtering based on subsequence matching: A new approach," *Information Sciences*, vol. 405, pp. 118–134, 2017.

[9] F. Ricci, L. Rokach, and B. Shapira, "Recommender systems: Techniques, applications, and challenges," in *Recommender Systems Handbook* (F. Ricci, L. Rokach, and B. Shapira, eds.), pp. 1–35, Springer US, 2022.

[10] P. Sánchez and A. Bellogín, "Building user profiles based on sequences for content and collaborative filtering." https://doi.org/10.1016/j.ipm.2018.10.003, 2019. Last accessed: April 2025.

[11] M.-P. T. Do, D. V. Nguyen, and L. Nguyen, "Model-based approach for collaborative filtering," in *Proceedings of the 6th International Conference on Information Technology for Education (IT@EDU2010)*, (Ho Chi Minh City, Vietnam), 2010.

[12] H. Al-bashiri, M. A. Abdulgabber, A. Romli, and F. Hujainah, "Collaborative filtering recommender system: Overview and challenges," *Journal of Computational and Theoretical Nanoscience*, vol. 23, no. 9, pp. 9045–9049, 2017.

[13] Xomnia, "The power of personalization: A guide to recommendation systems." https://www.xomnia.com/post/recommender-engines-a-different-approach-with-network-sci 2023. Last accessed: February 2025.

[14] A. Goodreads, "About goodreads." https://www.goodreads.com/about/us, s. f. Last accessed: May 2025.

[15] A. G. P. R. G. N. . Interviews, "Announcing goodreads personalized recommendations." https://www.goodreads.com/blog/show/303-announcing-goodreads-personalized-recommendations, s. f.-b. Last accessed: December 2024.

[16] Planeta, "Casa del libro: A community of readers in spain." https://planeta.es/en/books#:~:text=12%20publishing%20imprints.-,Casa%20del%20Libro,community%20of%20readers%20in%20Spain., 2025. Last accessed: December 2024.

[17] L. más vendidos y leídos 2023, "Libros más vendidos y leídos 2023." https://www.casadellibro.com/libros-mas-vendidos, s. f. Last accessed: December 2024.

[18] L. L. más Recomendados del año, "Los libros más recomendados del año." https://www.casadellibro.com/libros-recomendados, s. f. Last accessed: December 2024.

[19] L. recomendados por autores, "Libros recomendados por autores." https://www.casadellibro.com/recomendaciones-de-escritores-y-escritoras, s. f. Last accessed: December 2024.

[20] A. Michaelides, *La paciente silenciosa*. Best Seller, Madrid, España: DEBOLSILLO, 2022. Edición en formato bolsillo.

[21] JealousLeopard, "Goodreads books dataset." https://www.kaggle.com/datasets/jealousleopard/goodreadsbooks, 2025. Last accessed: November 2024.

[22] SaurabhBagchi, "Books dataset." https://www.kaggle.com/datasets/saurabhbagchi/books-dataset, 2025. Last accessed: November 2024.

[23] somnambwl, "Book-crossing dataset." https://www.kaggle.com/datasets/somnambwl/bookcrossing-dataset, 2025. Last accessed: November 2024.

[24] Goodreads Help, "Does goodreads support the use of apis?." https://help.goodreads.com/s/article/Does-Goodreads-support-the-use-of-APIs, 2020. Accessed: 2025-04-26.

[25] Axarnet, "Qué es un web crawler y cómo influye en el seo." https://axarnet.es/blog/web-crawler, s. f. Last accessed: December 2024.

[26] Cloudflare.com, "¿qué es el data scraping?." https://www.cloudflare.com/es-es/learning/bots/what-is-data-scraping/, s. f. Last accessed: January 2025.

[27] Selenium, "Selenium." https://pypi.org/project/selenium/, s. f. Last accessed: February 2025.

[28] Beautifulsoup4, "Beautifulsoup4." https://pypi.org/project/beautifulsoup4/, s. f. Last accessed: February 2025.

[29] Requests, "Requests." https://pypi.org/project/requests/, s. f. Last accessed: February 2025.

[30] H. Kiran, "Web crawling vs. web scraping [4 key differences]." https://techjury.net/blog/

web-crawling-vs-web-scraping/, octubre 2023. Last accessed: February 2025.

[31] A. Ken, ""backend: ¿qué es y para qué sirve?"." https://www.gluo.mx/blog/backend-que-es-y-para-que-sirve, s. f. Last accessed: February 2025.

[32] Arsys, "Frontend: ¿qué es y para qué se utiliza en desarrollo web?." https://www.arsys.es/blog/frontend-que-es-y-para-que-se-utiliza-en-desarrollo-web, agosto 2024. Last accessed: February 2025.

[33] MDN Contributors, "Django introduction." https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Django/Introduction. Last accessed: 2025-06-18.

[34] T. Olanrewaju, "How django's mvt architecture works: A deep dive into models, views, and templates." https://www.freecodecamp.org/news/how-django-mvt-architecture-works/. Last accessed: 2025-06-18.

[35] GeeksforGeeks, "Django project mvt structure." https://www.geeksforgeeks.org/django-project-mvt-structure/, agosto 2021. Last accessed: March 2025.

[36] I. Amazon Web Services, "¿qué es django? - explicación del software django." https://aws.amazon.com/es/what-is/django/#:~:text=Django%20es%20un%20software%20que,y%20la%20administraci%C3%B3n%20de%20cookies, s. f. Last accessed: March 2025.

[37] CodeMaple, "Understanding django mvt architecture and view functions – django full course for beginners (lesson)." https://medium.com/@CodeMaple/understanding-django-mvt-architecture-and-view-functions-django-full-cour 2023. Accessed: 2025-05-31.

[38] Pallets Projects, "Flask." https://pypi.org/project/Flask/. Last accessed: 2025-06-18.

[39] Express.js Contributors, "Express: Fast, unopinionated, minimalist web framework for node.js." https://expressjs.com/, 2025. Last accessed: 2025-06-18.

[40] MaestrosWeb, "Desarrollando frameworks: Comparativa django vs expressjs." https://maestrosweb.puntanetwork.com/full-stack-development/desarrollando-frameworks-comparativa-django-expressjs/, 2025. Last accessed: March 2025.

[41] N. Team, "State management." https://nuxt.com/docs/getting-started/state-management, 2025. Last accessed: 2025-06-18.

[42] V. Contributors, "Server-side rendering (ssr)." https://vuejs.org/guide/scaling-up/ssr.html, 2025. Last accessed: 2025-06-17.

[43] N. Rifki, "What you should know about code-splitting with nuxt.js." https://www.telerik.com/blogs/what-you-should-know-code-splitting-nuxtjs, 2020. Last accessed: 2025-06-17.

[44] Vue.js, "Vue.js guide: Introduction." https://vuejs.org/guide/introduction.html,

2025. Last accessed: March 2025.

[45] Next.js Templates, "Next.js vs nuxt: A comprehensive comparison." https://nextjstemplates.com/blog/nextjs-vs-nuxt, 2025. Last accessed: March 2025.

[46] I. Neon Technologies, "Neon – serverless postgres, built for the cloud." https://neon.com/, 2024. Last accessed: May 2025.

[47] Render, "Render – the modern cloud for developers." https://render.com/, 2024. Last accessed: May 2025.

[48] Supabase, "Supabase – the open source firebase alternative." https://supabase.com/, 2024. Last accessed: May 2025.

[49] Pinia Contributors, "Nuxt integration." https://pinia.vuejs.org/ssr/nuxt.html, 2025. Last accessed: January 2025.

[50] Nuxt.js Contributors, "Tailwind css module for nuxt." https://tailwindcss.nuxtjs.org/, 2025. Last accessed: January 2025.

[51] Penguin Random House Grupo Editorial, "Nuestros sellos." https://www.penguinrandomhousegrupoeditorial.com/sellos-y-negocios/nuestros-sellos/, 2025. Last accessed: May 2025.

[52] J. Brooke, "Sus: A "quick and dirty" usability scale," in *Usability Evaluation in Industry* (P. W. Jordan, B. Thomas, B. A. Weerdmeester, and I. L. McClelland, eds.), pp. 189–194, London: Taylor & Francis, 1996. Originally developed at Digital Equipment Corporation.

APPENDICES

A

Coverless Interface

A.1 Register

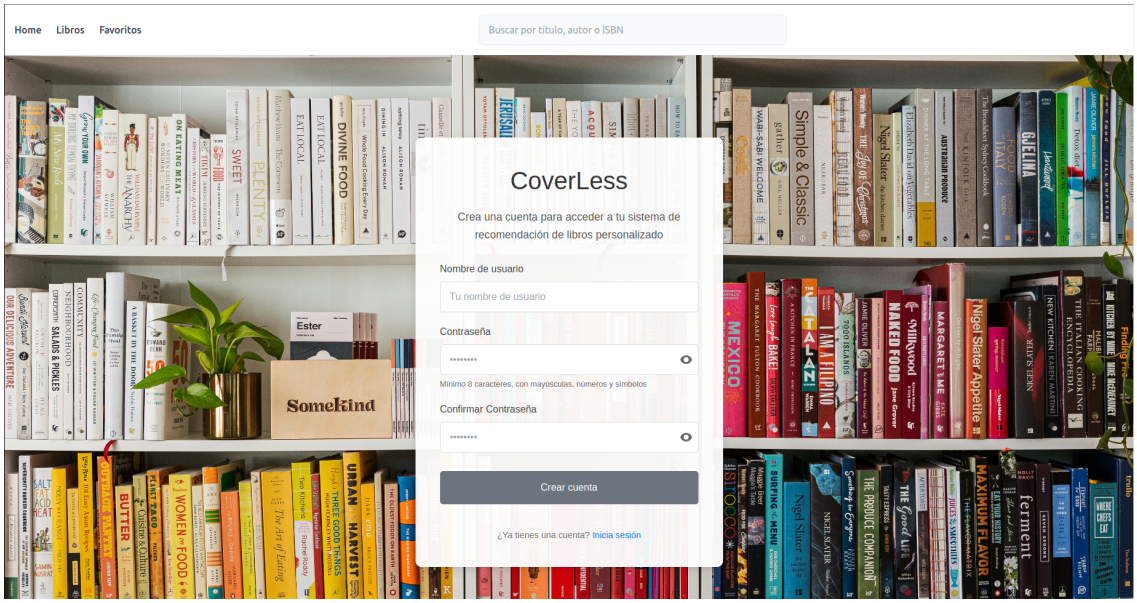


Figure A.1: Register Coverless page

A.2 Login

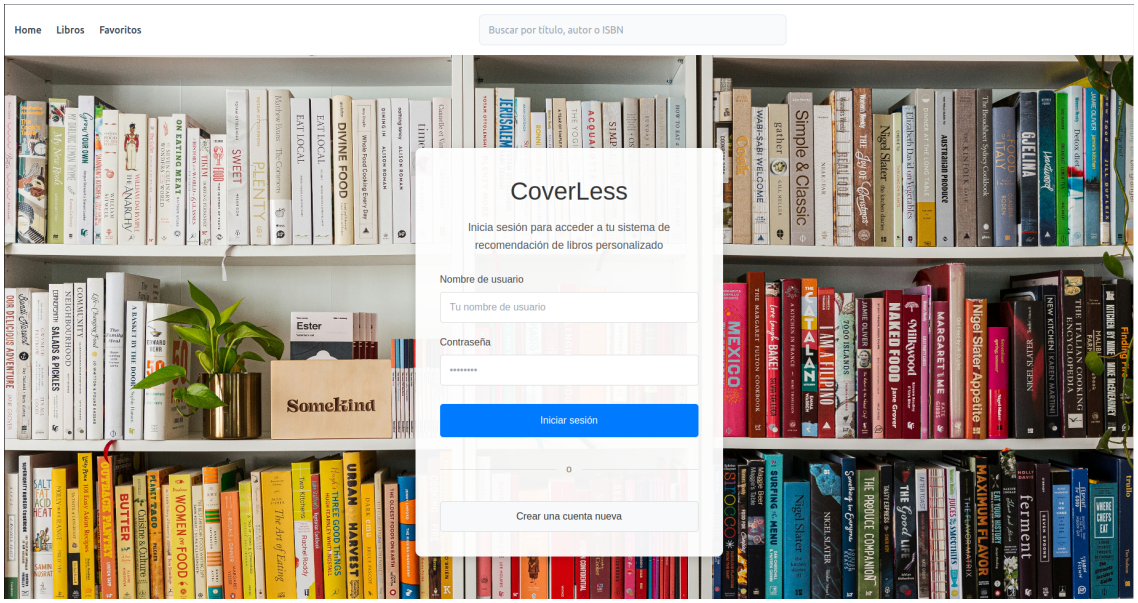


Figure A.2: Login Coverless page

A.3 Home

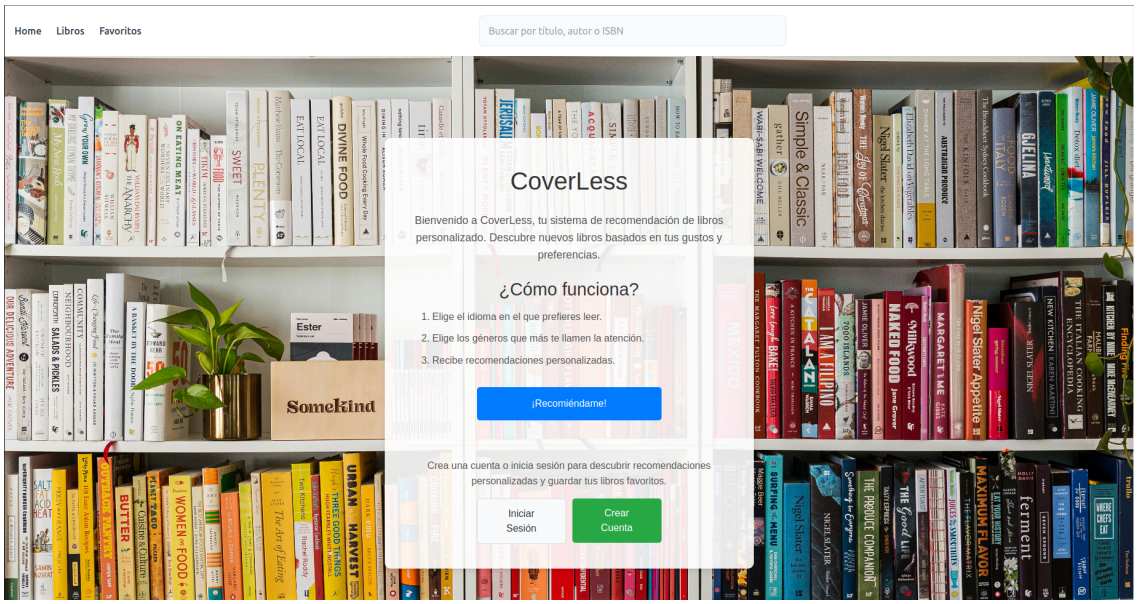


Figure A.3: Home Coverless page

A.4 Language selection

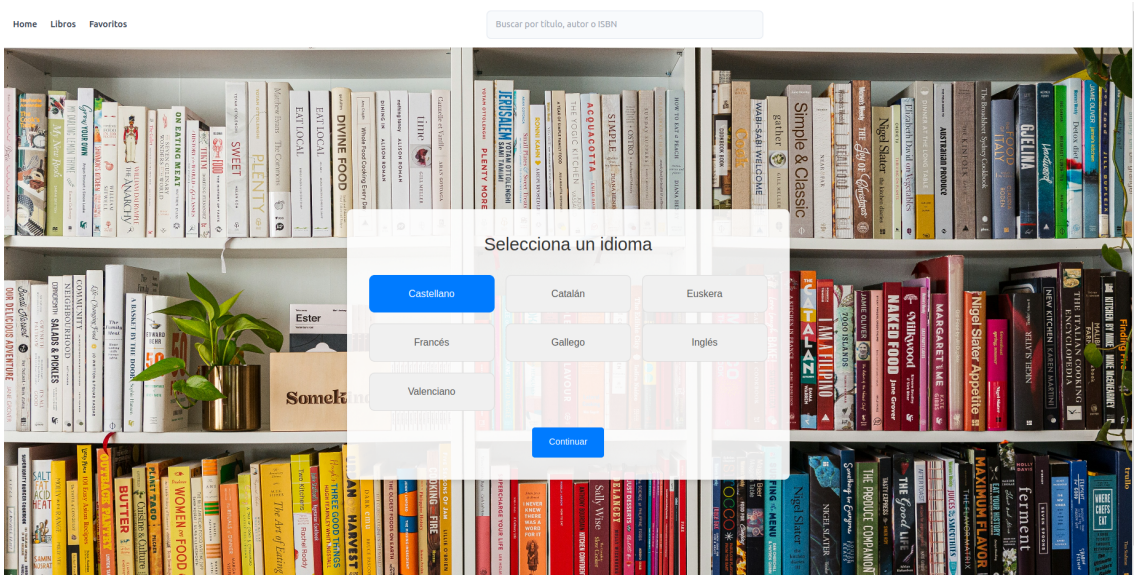


Figure A.4: Book language selection page

A.5 Genres selection

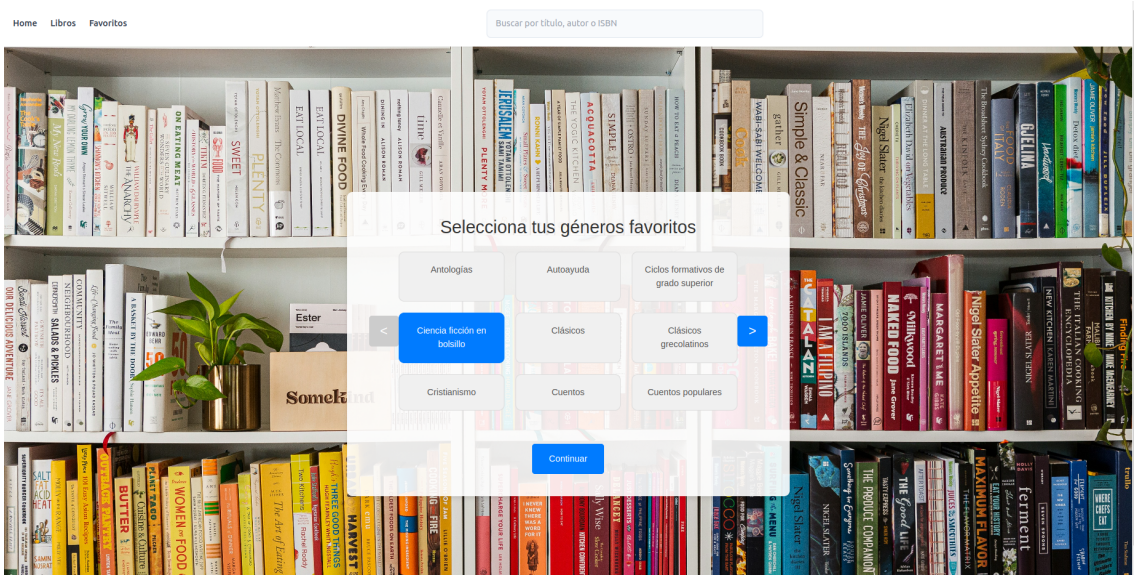


Figure A.5: Book genre selection page

A.6 Language and genre-based recommendations

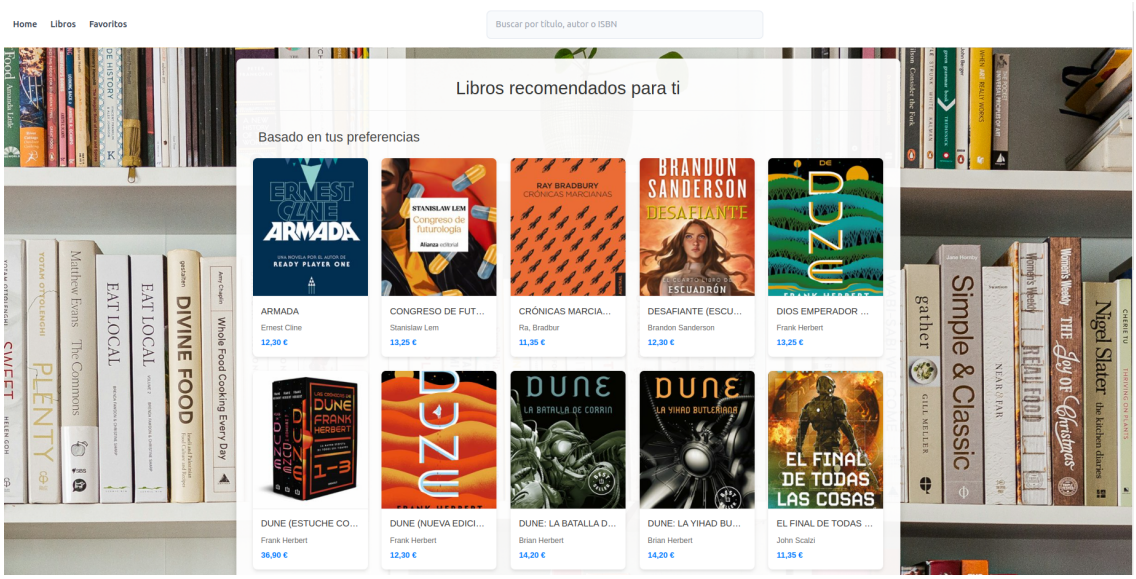


Figure A.6: Language and genre-based recommendations

A.7 New releases

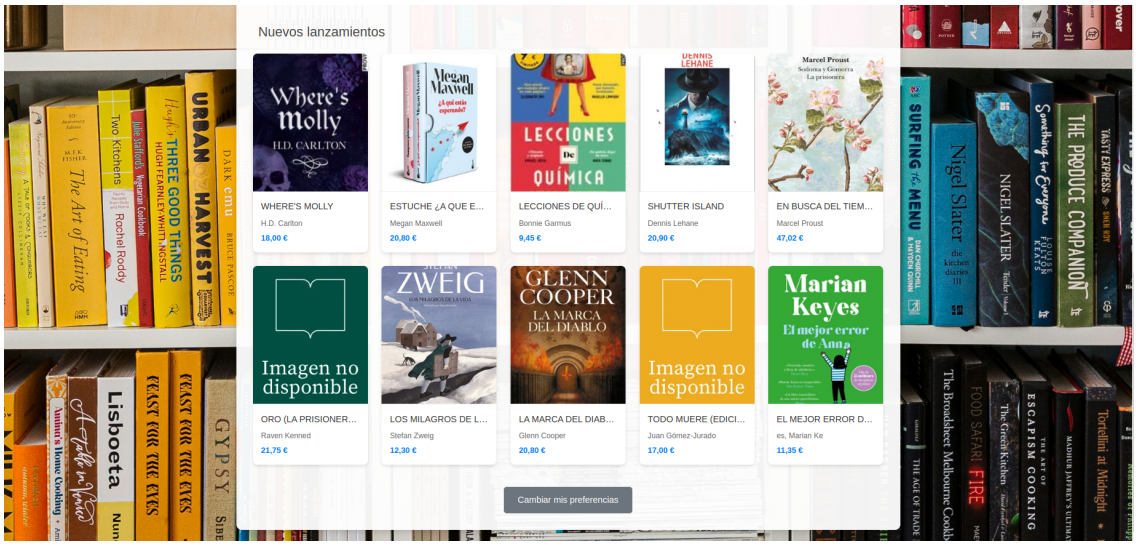


Figure A.7: New releases page

A.8 Search bar

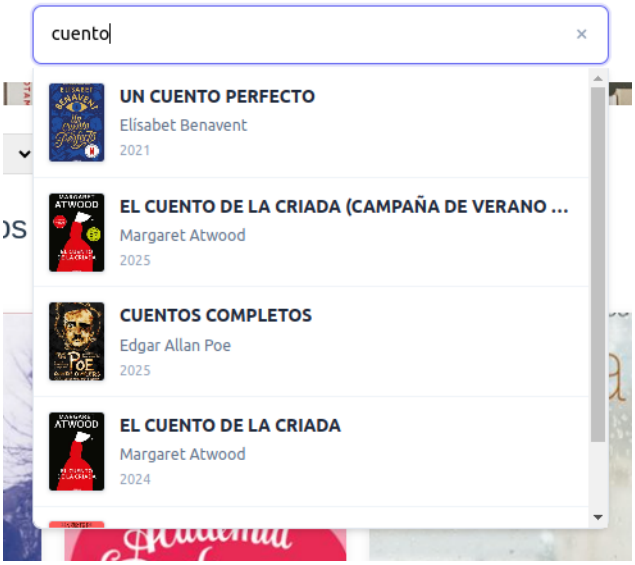


Figure A.8: Search bar

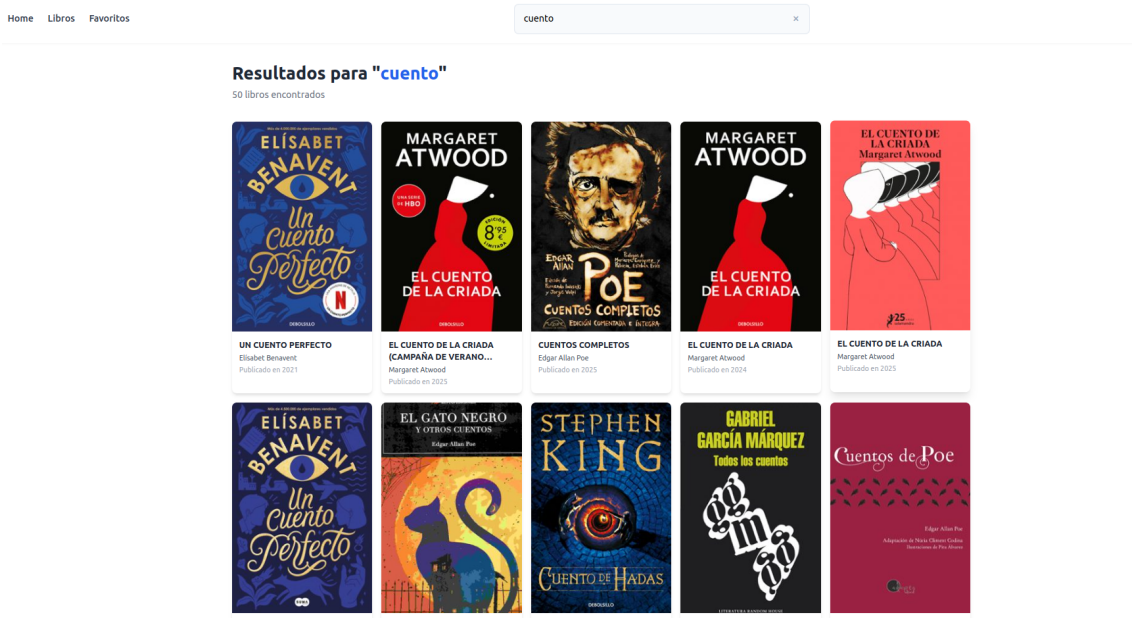


Figure A.9: Search page

A.9 Book detail page

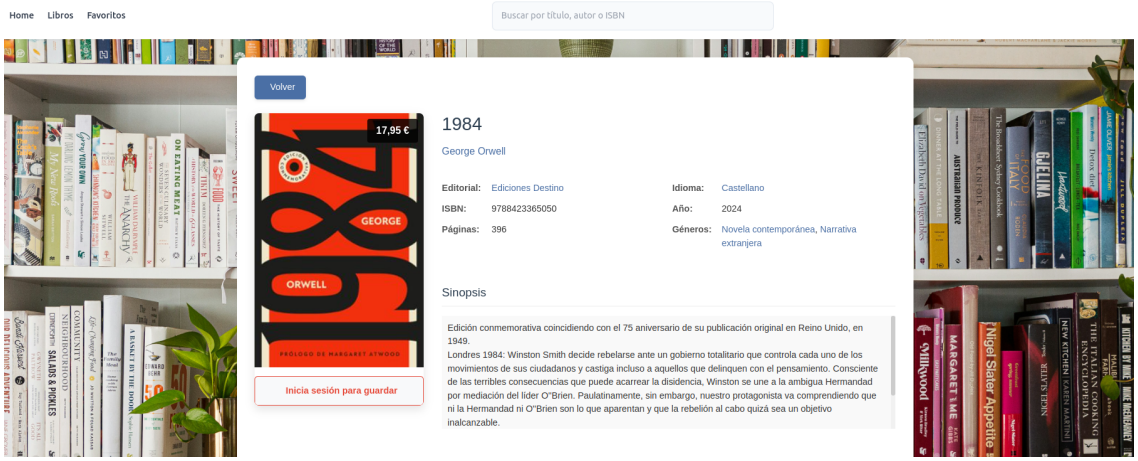


Figure A.10: Book detail page

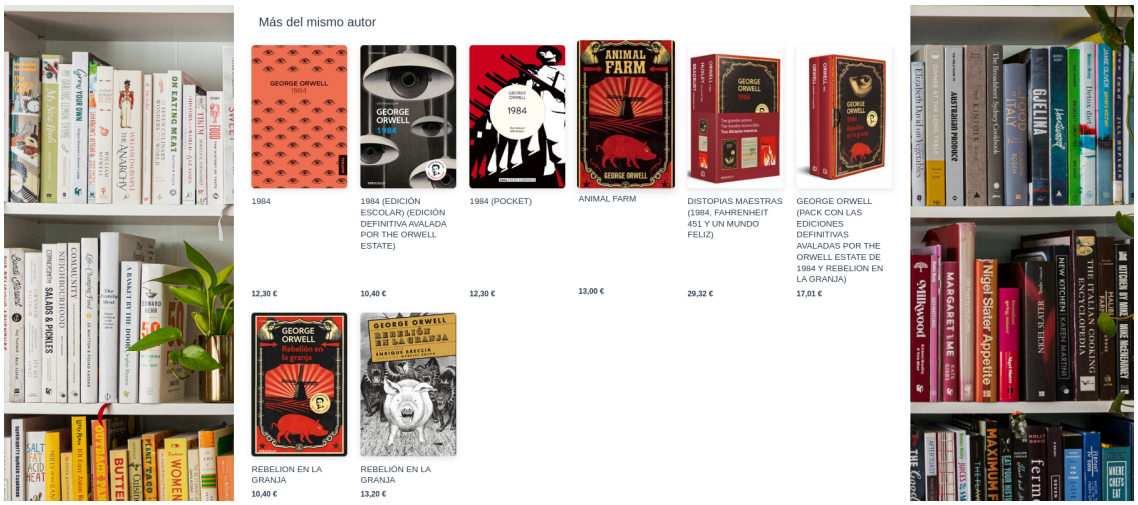


Figure A.11: More from the same author

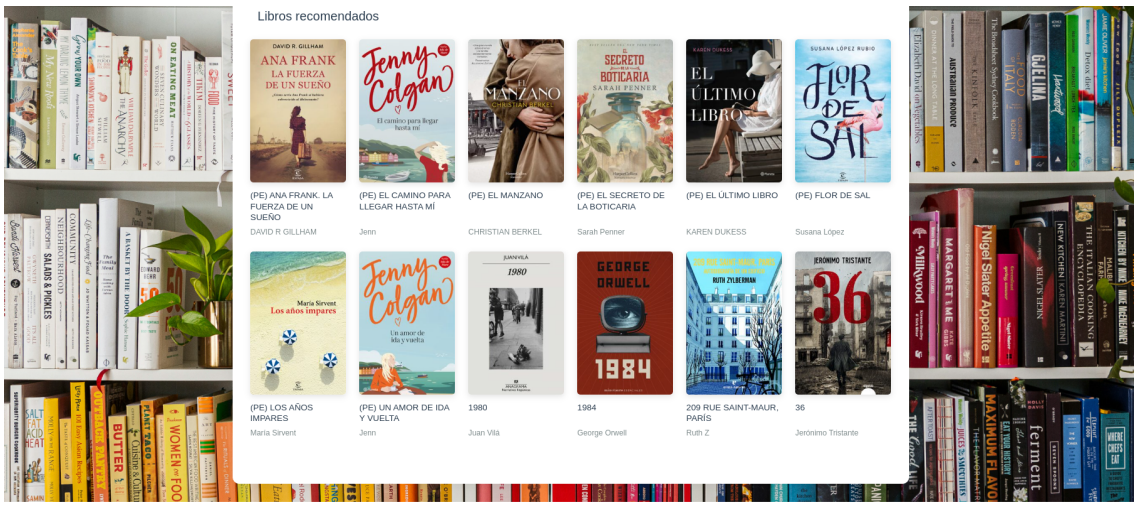


Figure A.12: Recommender books

A.10 Favorite books page

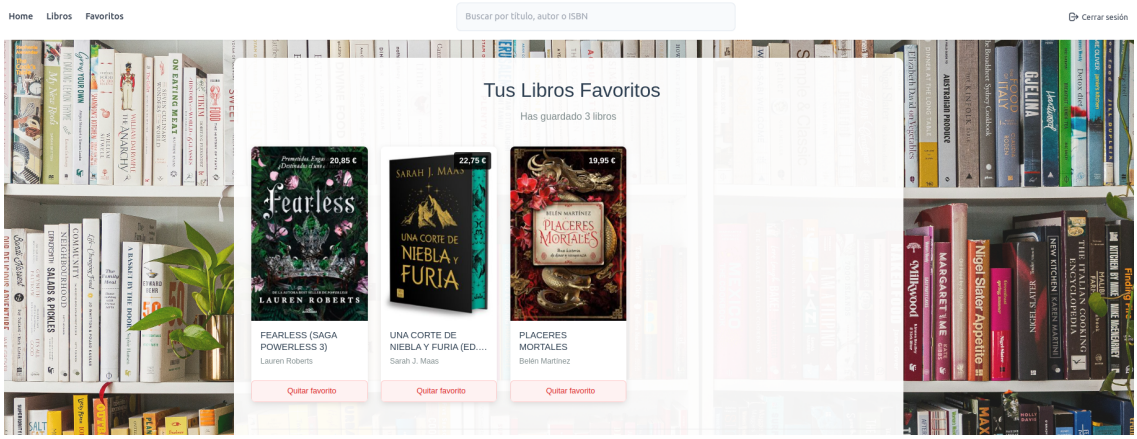


Figure A.13: Favorite books page

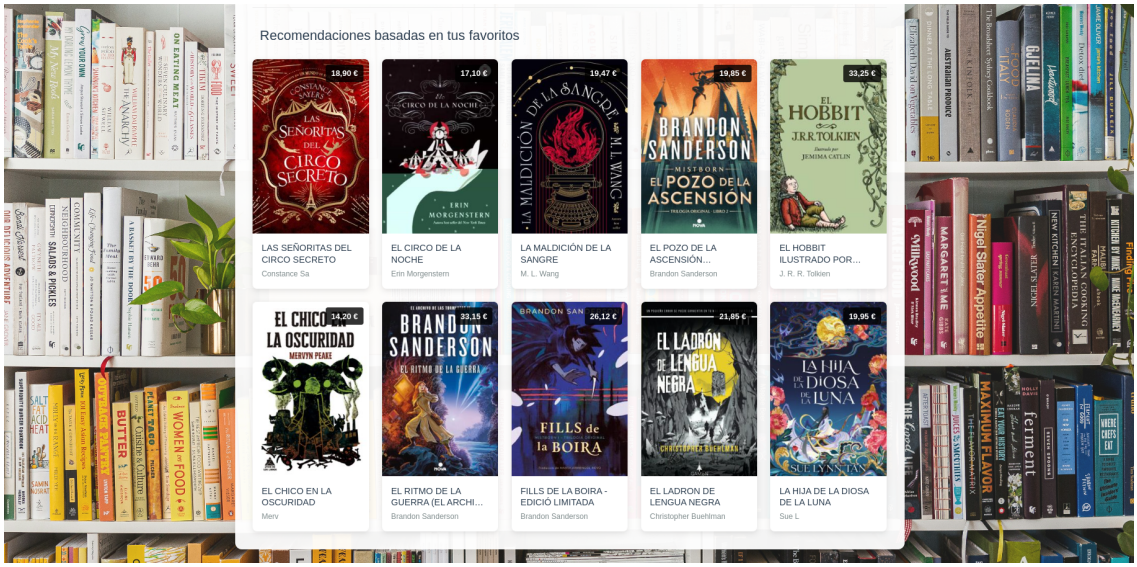


Figure A.14: Recommender based on the favorite books


UÁM
Universidad Autónoma
de Madrid