# **Software Requirement Specification (SRS)**

## **Project: Semantic Book Recommender with LLMs**

**Version:** 1.0

**Date:** September 28, 2025

**Author:** [Your Name/Company Name]

**Client:** NextRead Corp.

[Page 1]

## **Table of Contents**

1. Introduction ............................................................................................ 4

1.1. Purpose ............................................................................................ 4

1.2. Scope .............................................................................................. 5

1.3. Definitions, Acronyms, and Abbreviations ................................ 6

2. Overall Description ................................................................................. 8

2.1. Product Perspective ...................................................................... 8

2.2. Product Functions ......................................................................... 9

2.3. User Characteristics ................................................................... 10

2.4. Constraints ................................................................................... 11

2.5. Assumptions and Dependencies .............................................. 12

3. Specific Requirements ........................................................................ 13

3.1. Functional Requirements ........................................................... 13

3.1.1. FR-1: Semantic Querying System ................................. 13

3.1.2. FR-2: Recommendation Display ................................... 14

3.1.3. FR-3: Genre-Based Filtering ....................................... 15

3.1.4. FR-4: Emotion-Based Filtering .................................... 16

3.1.5. FR-5: User Interface Components .............................. 17

3.2. Non-Functional Requirements ................................................... 18

3.2.1. NFR-1: Performance ...................................................... 18

3.2.2. NFR-2: Usability ............................................................ 19

3.2.3. NFR-3: Reliability .......................................................... 20

3.2.4. NFR-4: Scalability .......................................................... 21

3.2.5. NFR-5: Maintainability .................................................. 21

3.3. External Interface Requirements ............................................. 22

3.3.1. User Interfaces ............................................................... 22

3.3.2. Hardware Interfaces ..................................................... 23

3.3.3. Software Interfaces ...................................................... 23

3.3.4. Communications Interfaces ........................................ 24

[Page 2]

3.4. Use Cases .................................................................................... 25  
 3.4.1. Use Case 1: Basic Semantic Search .......................... 25  
 3.4.2. Use Case 2: Filtered Search ..................................... 26  
3.5. Sequence Diagrams ................................................................... 27  
 3.5.1. Basic Search Sequence .............................................. 27  
 3.5.2. Filtered Search Sequence .......................................... 28

4. Appendices .......................................................................................... 29

4.1. Appendix A: System Architecture Diagram ............................. 29

4.2. Appendix B: Source Code and Data Files ............................ 30

[Page 3]

## **1. Introduction**

### **1.1. Purpose**

This Software Requirement Specification (SRS) document provides a detailed description of the requirements for the **Semantic Book Recommender with LLMs** project. Its purpose is to serve as a comprehensive guide and single source of truth for all stakeholders involved in the development, testing, and deployment of this system.

This document is intended for:

* **Developers and Engineers:** To understand the system's architecture, functional and non-functional requirements, and external interfaces, enabling them to build the software accurately.
* **Quality Assurance (QA) Team:** To create test plans, test cases, and validation scripts to ensure the software meets the specified requirements.
* **Project Managers:** To manage the project scope, track progress, and ensure that the final product aligns with the client's objectives.
* **Client Stakeholders (NextRead Corp.):** To review and formally approve the specified requirements, ensuring they accurately reflect their business needs and vision for the product.

This SRS defines what the system will do, how it will behave, and the constraints under which it will operate. It establishes the foundation for the project's design and implementation phases.

[Page 4]

### **1.2. Scope**

The scope of this project is the development of a web-based, proof-of-concept book recommendation application, referred to as the **Semantic Book Recommender**. This system will function as the Minimum Viable Product (MVP) for NextRead Corp.

The primary functionalities of the software include:

1. **Semantic Search:** Allowing end-users to input natural language queries to search for books based on thematic and conceptual similarity rather than simple keyword matching.
2. **Content-Based Filtering:** Providing users with the ability to refine search results based on two key attributes:
   * **Genre:** A high-level classification of books into "Fiction" and "Non-fiction," determined by a zero-shot learning model.
   * **Emotional Tone:** A mood-based filter that allows users to find books matching sentiments like "Happy," "Suspenseful," or "Sad," derived from an emotion analysis model.
3. **Interactive User Interface:** A single-page web application that provides an intuitive interface for entering queries, applying filters, and viewing recommendations in a visual gallery format.

Out of Scope:

The following features are explicitly considered out of scope for this version of the project:

* User accounts, profiles, and personalized history.
* User-submitted ratings, reviews, or feedback mechanisms.
* Real-time data ingestion for new books; the system will operate on a static, pre-processed dataset.
* E-commerce functionalities (e.g., purchasing books).
* Deployment to a production-grade, public-facing cloud infrastructure. The initial deployment will be for local execution and demonstration purposes.

[Page 5]

### **1.3. Definitions, Acronyms, and Abbreviations**

| **Term/Acronym** | **Definition** |
| --- | --- |
| **SRS** | Software Requirement Specification. This document. |
| **CRS** | Client Requirement Specification. The document detailing the client's high-level needs. |
| **LLM** | Large Language Model. A type of AI model trained on vast text data to understand and generate language. |
| **NLP** | Natural Language Processing. A field of AI focused on enabling computers to understand and process human language. |
| **Embedding** | A numerical, vector representation of text that captures its semantic meaning. |
| **Semantic Search** | A search technique based on the meaning and context of a query, powered by embeddings. |
| **Vector Database** | A database optimized for storing and querying high-dimensional vectors, like text embeddings. |
| **Chroma DB** | The specific open-source vector database used in this project. |
| **Zero-Shot Learning** | An LLM's ability to perform a task (e.g., classification) without having been explicitly trained on examples for that task. |
| **Sentiment Analysis** | The computational process of identifying and categorizing emotions or opinions in text. |
| **Gradio** | The Python library used to build the web-based user interface for this application. |
| **API** | Application Programming Interface. A contract that allows software components to communicate. In this context, it primarily refers to the OpenAI API for embeddings. |
| **MVP** | Minimum Viable Product. The initial version of the product with just enough features to be usable by early customers. |
| **UI** | User Interface. The graphical layout through which a user interacts with the system. |
| **FR** | Functional Requirement. A statement of what the system must do. |
| **NFR** | Non-Functional Requirement. A statement of how the system must perform (e.g., quality attributes like speed or usability). |
| **ISBN** | International Standard Book Number. A unique numeric identifier for books. |

[Page 6]

[Page 7]

## **2. Overall Description**

### **2.1. Product Perspective**

The Semantic Book Recommender is a new, self-contained product that will serve as the core technology demonstration for NextRead Corp. It is not an extension of any existing system.

* **System Interfaces:** The application is a standalone web application. It interfaces with several external software components:
  + **User's Web Browser:** The primary interface for end-users.
  + **OpenAI API:** An external, third-party service required to generate text embeddings for user queries in real-time.
  + **Hugging Face Models:** The pre-trained models for sentiment analysis and zero-shot classification are downloaded from the Hugging Face model hub and run locally.
* **Product Context:** This system is the MVP for NextRead Corp. and is intended to be the foundation for a future, more comprehensive literary discovery platform. Its architecture is designed with modularity in mind to facilitate future expansion and integration with other services (e.g., e-commerce platforms, user account systems).

[Page 8]

### **2.2. Product Functions**

The Semantic Book Recommender provides the following key functions to the user:

1. **Book Discovery via Semantic Search:** The primary function is to allow users to input a free-text, natural language query. The system processes this query to understand its semantic meaning and returns a list of books from its database whose descriptions are conceptually similar.
2. **Recommendation Filtering by Genre:** Users can refine the list of semantically relevant books by applying a high-level genre filter. The available filters are "Fiction" and "Non-fiction." This allows users to narrow their results based on the type of content they are seeking.
3. **Recommendation Re-ranking by Mood:** Users can further tailor the results to match their desired emotional experience. By selecting a mood (e.g., "Happy," "Suspenseful"), the system re-ranks the filtered list of books to prioritize those whose descriptions exhibit the strongest emotional signal for the chosen tone.
4. **Visual Browsing of Results:** The system presents the final list of recommended books in a user-friendly gallery format. This allows users to quickly scan book covers and read brief summaries to evaluate the recommendations.

These functions combine to create a powerful and intuitive discovery workflow that goes beyond traditional keyword search.

[Page 9]

### **2.3. User Characteristics**

The intended users of the Semantic Book Recommender fall into a single primary category, though with varying levels of technical proficiency.

| **User Category** | **Description** | **Technical Proficiency** |
| --- | --- | --- |
| **Reader** | The target audience includes avid readers, casual readers, students, and anyone looking for a new book to read. | **Low to High.** The system is designed for a general audience and assumes no special technical knowledge. The interface must be simple enough for a novice computer user to operate effectively. |

**Key Characteristics and Needs:**

* **Goal-Oriented:** Users have a primary goal: to find a book that matches their current interests or mood. They value efficiency and relevance.
* **Varied Search Strategies:** Some users may have a very specific idea of what they want (e.g., a specific sub-genre), while others may have a vague, thematic, or mood-based idea. The system must cater to both.
* **Visually Driven:** The visual appeal of a book cover is often a significant factor in a reader's choice. The UI must prioritize the visual presentation of recommendations.
* **Expectation of Speed:** Users expect a modern web application to return results quickly. Delays will lead to a negative perception of the system's quality.

[Page 10]

### **2.4. Constraints**

The development of the Semantic Book Recommender is subject to the following constraints:

1. **Technological Constraints:**
   * The backend must be implemented in **Python 3.11**.
   * The web user interface must be built using the **Gradio** library.
   * The vector database for the MVP must be a local instance of **Chroma DB**.
   * The project must use the specific libraries and versions listed in the requirements.txt file.
2. **Data Constraints:**
   * The system will be developed using the provided static dataset of approximately 5,200 cleaned book records. There will be no live data feed for this version.
   * The quality of the recommendations is inherently limited by the quality and content of the descriptions in the source dataset.
3. **API and External Service Constraints:**
   * The system requires a valid **OpenAI API key** to function. This introduces an external dependency.
   * The performance and availability of the recommendation system are partially dependent on the performance and availability of the OpenAI API.
   * API usage may incur costs, which must be managed. Rate limits on the API could also impact performance under heavy load.
4. **Deployment Constraints:**
   * The MVP is intended for local deployment and demonstration. The design does not need to account for complexities of a large-scale, distributed cloud deployment at this stage.

[Page 11]

### **2.5. Assumptions and Dependencies**

**Assumptions:**

* It is assumed that the descriptions provided in the dataset are sufficiently descriptive to generate meaningful semantic embeddings and emotion scores.
* It is assumed that the pre-trained LLMs used for classification and sentiment analysis are accurate enough for the purposes of this MVP without requiring further fine-tuning.
* It is assumed that users will have a modern web browser and a stable internet connection to access the application and for the OpenAI API calls to succeed.
* It is assumed that the client (NextRead Corp.) will provide a valid and funded OpenAI API key for development and demonstration.

**Dependencies:**

* **Python 3.11 Environment:** The application is dependent on a specific version of the Python runtime.
* **External Python Libraries:** The project is dependent on the libraries listed in requirements.txt. The absence or version mismatch of any of these libraries will prevent the application from running.
* **OpenAI API Service:** The application has a critical, real-time dependency on the OpenAI API for embedding user queries. If this service is unavailable, the semantic search functionality will fail.
* **Static Data Files:** The application depends on the presence of books\_with\_emotions.csv and tagged\_description.txt in its root directory to load the necessary book data and initialize the vector database.

[Page 12]

## **3. Specific Requirements**

### **3.1. Functional Requirements**

This section provides a detailed breakdown of the system's functional requirements.

#### **3.1.1. FR-1: Semantic Querying System**

ID: FR-1

Description: The system shall process a user's natural language query to find and rank books based on semantic similarity.

**Inputs:**

* A string of text provided by the user.

**Processing:**

* FR-1.1: The system shall accept a UTF-8 encoded text string from the user input field.
* FR-1.2: The system shall make a request to the configured OpenAI Embeddings API endpoint, passing the user's query string.
* FR-1.3: The system shall receive a high-dimensional vector embedding from the API in response.
* FR-1.4: The system shall use the LangChain Chroma integration to perform a similarity search in the pre-loaded vector database, using the query embedding.
* FR-1.5: The similarity search shall use a standard metric (e.g., Cosine Similarity or L2 Distance) to compare vectors.
* FR-1.6: The system shall retrieve the top k (default k=50) results from the vector database. Each result will contain the tagged\_description which includes the book's isbn13.
* FR-1.7: The system shall parse the isbn13 from each result.

**Outputs:**

* A ranked list of isbn13 identifiers, ordered from most to least semantically similar to the query.

[Page 13]

#### **3.1.2. FR-2: Recommendation Display**

ID: FR-2

Description: The system shall present the final list of recommended books to the user in a graphical gallery format.

**Inputs:**

* A pandas DataFrame containing the records of the books to be displayed, filtered and ranked according to the specified logic.

**Processing:**

* FR-2.1: The system shall iterate through each row of the input DataFrame.
* FR-2.2: For each book, the system shall construct an image URL. If the thumbnail field is valid, it will be used. If it is null or invalid, a predefined local placeholder image URL (cover-not-found.jpg) will be used. The system will also append &fife=w800 to valid Google Books URLs to request a larger image.
* FR-2.3: For each book, the system shall construct a caption string.
  + The caption must begin with the book's title.
  + This is followed by " by " and the formatted authors string. The author string must be grammatically correct for single and multiple authors (e.g., "Author A", "Author A and Author B", "Author A, Author B, and Author C").
  + This is followed by a colon and a truncated description. The description must be limited to the first 30 words, followed by an ellipsis (...).
* FR-2.4: The system shall compile a list of tuples, where each tuple contains (image\_url, caption).

**Outputs:**

* A list of tuples formatted for the Gradio Gallery component.

[Page 14]

#### **3.1.3. FR-3: Genre-Based Filtering**

ID: FR-3

Description: The system shall filter the list of semantically relevant books based on the user's selected genre.

**Inputs:**

* A pandas DataFrame containing the initial list of books from the semantic search.
* A string representing the selected category ('All', 'Fiction', or 'Non-fiction').

**Processing:**

* FR-3.1: If the selected category is "All", the system shall not perform any filtering and will pass the input DataFrame through without modification.
* FR-3.2: If the selected category is "Fiction" or "Non-fiction", the system shall filter the input DataFrame, retaining only the rows where the value in the simple\_categories column matches the selected category string.

**Outputs:**

* A pandas DataFrame containing a subset of the input books that match the genre filter.

[Page 15]

#### **3.1.4. FR-4: Emotion-Based Filtering**

ID: FR-4

Description: The system shall re-rank the list of recommended books based on the user's selected emotional tone.

**Inputs:**

* A pandas DataFrame containing the list of books (already processed by semantic search and genre filtering).
* A string representing the selected tone ('All', 'Happy', 'Surprising', 'Angry', 'Suspenseful', 'Sad').

**Processing:**

* FR-4.1: If the selected tone is "All", the system shall not perform any re-ranking and will pass the input DataFrame through without modification.
* FR-4.2: If a specific tone is selected, the system shall map the tone string to its corresponding column name in the DataFrame:
  + 'Happy' -> joy
  + 'Surprising' -> surprise
  + 'Angry' -> anger
  + 'Suspenseful' -> fear
  + 'Sad' -> sadness
* FR-4.3: The system shall sort the input DataFrame in descending order based on the values in the mapped emotion column.

**Outputs:**

* A pandas DataFrame with the same books as the input, but re-ordered according to the selected emotional tone.

[Page 16]

#### **3.1.5. FR-5: User Interface Components**

ID: FR-5

Description: The system shall provide specific, interactive UI components for user input and output.

**Inputs:**

* User interactions (typing, clicking, selecting from dropdowns).

**Processing:**

* FR-5.1: The system shall instantiate a gradio.Textbox component with a predefined label ("Please enter a description of a book:") and placeholder text.
* FR-5.2: The system shall instantiate two gradio.Dropdown components.
  + One for categories, populated with the unique values from the simple\_categories column, plus "All".
  + One for tones, populated with a predefined list of emotions, plus "All".
  + Both dropdowns must default to "All".
* FR-5.3: The system shall instantiate a gradio.Button with the text "Find recommendations".
* FR-5.4: The system shall instantiate a gradio.Gallery component to display the results, configured with 8 columns.
* FR-5.5: The system shall link the click event of the button to the main recommend\_books function, mapping the UI components to the function's inputs and outputs.

**Outputs:**

* A rendered, interactive single-page web application.

[Page 17]

### **3.2. Non-Functional Requirements**

This section details the quality attributes and constraints of the system.

#### **3.2.1. NFR-1: Performance**

ID: NFR-1

Description: The system's response time must be adequate for a real-time interactive application.

**Requirements:**

* **NFR-1.1 (Query Response Time):** The average latency from a user submitting a query to the results being displayed shall be less than **3 seconds**. This is contingent on the external OpenAI API latency being within normal parameters (under 1 second).
* **NFR-1.2 (Page Load Time):** The initial load time of the Gradio web interface shall be less than **5 seconds**.
* **NFR-1.3 (Throughput):** The system, running on a standard local machine (e.g., 4-core CPU, 8GB RAM), shall support at least **5 concurrent users** without response time exceeding 5 seconds.

[Page 18]

#### **3.2.2. NFR-2: Usability**

ID: NFR-2

Description: The user interface must be simple, intuitive, and accessible to a general audience.

**Requirements:**

* **NFR-2.1 (Learnability):** The system must be usable without any training or documentation. The function of each UI component should be immediately obvious from its label and design.
* **NFR-2.2 (Efficiency):** A user must be able to complete a full search-and-filter workflow (enter query, select category, select tone, click button) in **fewer than 5 clicks/interactions**.
* **NFR-2.3 (Accessibility):** The UI text and components should have sufficient contrast and be legible. While full WCAG compliance is not in scope for the MVP, basic readability principles must be followed.

[Page 19]

#### **3.2.3. NFR-3: Reliability**

ID: NFR-3

Description: The system must be stable and handle errors gracefully.

**Requirements:**

* **NFR-3.1 (Availability):** The application shall target a 99% uptime for demonstration purposes. This is primarily dependent on the stability of the machine it is running on and the uptime of the OpenAI API.
* **NFR-3.2 (Error Handling):** The application must not crash due to invalid user input or external API failures.
  + If the OpenAI API call fails (e.g., invalid key, network error), the backend shall catch the exception and return an empty list of recommendations to the UI.
  + Malformed or empty user queries shall not cause backend errors and should result in an empty recommendation list.
* **NFR-3.3 (Data Consistency):** The system must ensure that every isbn13 returned from the vector search corresponds to a valid entry in the main book data CSV.

[Page 20]

#### **3.2.4. NFR-4: Scalability**

ID: NFR-4

Description: The architecture must be designed to allow for future expansion.

**Requirements:**

* **NFR-4.1 (Vertical Scalability):** The application logic should be efficient enough that its performance scales reasonably with increased hardware resources (CPU, RAM).
* **NFR-4.2 (Data Scalability):** The use of a dedicated vector database (Chroma DB) ensures that the architecture can conceptually scale to handle a much larger dataset (e.g., >100,000 books) by migrating the database to a more powerful, dedicated server in the future.

#### **3.2.5. NFR-5: Maintainability**

ID: NFR-5

Description: The codebase must be clean, documented, and easy to maintain.

**Requirements:**

* **NFR-5.1 (Coding Standards):** All Python code shall adhere to the PEP 8 style guide.
* **NFR-5.2 (Documentation):** All functions shall include docstrings explaining their purpose, arguments, and return values.
* **NFR-5.3 (Configuration Management):** Sensitive information, such as the OpenAI API key, must not be hardcoded. It must be managed via environment variables using a .env file.

[Page 21]

### **3.3. External Interface Requirements**

#### **3.3.1. User Interfaces**

The only user interface is the web-based UI provided by Gradio.

* **Layout:** A single-page application with a header, a row of input controls, and a gallery for output.
* **Style:** The UI will use the built-in gr.themes.Glass() for a modern look and feel.
* **Input Controls:**
  + gr.Textbox: For free-text query input.
  + gr.Dropdown: For single-selection of category and tone.
  + gr.Button: To trigger the search action.
* **Output Control:**
  + gr.Gallery: To display a grid of images with captions.

[Page 22]

#### **3.3.2. Hardware Interfaces**

There are no direct hardware interfaces for this software. The application runs on standard computing hardware (e.g., a laptop or server) and does not interface with any specialized or custom hardware. Its hardware requirements are determined by the resource needs of the Python environment, the pandas library, and the loaded machine learning models.

#### **3.3.3. Software Interfaces**

The system interfaces with several key software components:

1. **OpenAI Embeddings API**
   * **Interface:** RESTful API.
   * **Protocol:** HTTPS.
   * **Data Format:** JSON.
   * **Interaction:** The application sends a POST request containing text to be embedded and receives a JSON object containing the vector embedding. This is managed via the langchain-openai library. Authentication is handled via an API key.
2. **Chroma DB**
   * **Interface:** Python library API (chromadb).
   * **Protocol:** In-process function calls (for local instance).
   * **Interaction:** The application uses the LangChain wrapper to create the database from documents (Chroma.from\_documents) and to perform searches (db\_books.similarity\_search).
3. **Hugging Face Transformers Library**
   * **Interface:** Python library API (transformers).
   * **Interaction:** The application uses the pipeline function to load pre-trained models for sentiment analysis and zero-shot classification. Models are downloaded from the Hugging Face Hub and cached locally.
4. **Pandas Library**
   * **Interface:** Python library API (pandas).
   * **Interaction:** The application loads, filters, and sorts book metadata using pandas DataFrame objects.

[Page 23]

#### **3.3.4. Communications Interfaces**

* **HTTP/HTTPS:** The application communicates with the OpenAI API over HTTPS. The Gradio web server uses HTTP to communicate with the user's web browser on the local network.
* **WebSockets:** The Gradio framework uses WebSockets for real-time communication between the frontend (browser) and the backend (Python script) to handle the interactive updates.

[Page 24]

### **3.4. Use Cases**

#### **3.4.1. Use Case 1: Basic Semantic Search**

| **Field** | **Description** |
| --- | --- |
| **ID** | UC-01 |
| **Name** | User performs a basic semantic search |
| **Actor** | Reader |
| **Preconditions** | User has opened the application in their browser. |
| **Trigger** | User wants to find a book based on a theme. |
| **Main Success Scenario** | 1. User types "a story of survival against the odds" into the query box.  2. User clicks "Find recommendations".  3. System displays a gallery of relevant books (e.g., "The Martian", "Life of Pi"). |
| **Exceptions** | - If the OpenAI API is unavailable, the system displays an empty gallery.  - If the query is nonsensical, the system displays a gallery of the most likely (but probably irrelevant) matches or an empty gallery. |

[Page 25]

#### **3.4.2. Use Case 2: Filtered Search**

| **Field** | **Description** |
| --- | --- |
| **ID** | UC-02 |
| **Name** | User performs a search refined by genre and mood |
| **Actor** | Reader |
| **Preconditions** | User has opened the application in their browser. |
| **Trigger** | User wants to find a specific type of book that matches their current mood. |
| **Main Success Scenario** | 1. User types "a detective mystery" into the query box.  2. User selects "Fiction" from the category dropdown.  3. User selects "Suspenseful" from the tone dropdown.  4. User clicks "Find recommendations".  5. System displays a gallery of fiction thrillers, ranked with the most suspenseful (highest 'fear' score) at the top. |
| **Exceptions** | - If no books match the combined semantic and filter criteria, the system displays an empty gallery. |

[Page 26]

### **3.5. Sequence Diagrams**

#### **3.5.1. Basic Search Sequence**

This diagram shows the sequence of interactions for a basic semantic search.

Actor: Reader UI (Gradio) Backend (Python) OpenAI API Vector DB (Chroma)  
 | | | | |  
1. enter\_query("...") -->| | | |  
2. click\_button() ------>| | | |  
 |-- 3. recommend("...")-->| | |  
 | |-- 4. embed\_query("...") -->| |  
 | | |-- 5. POST /embeddings ->|  
 | | |<-- 6. return\_embedding-|  
 | |<-- 7. return\_embedding----| |  
 | |-- 8. search(embedding) -->|---------------------->|  
 | | |<---- 9. return\_isbns ---|  
 | |<-- 10. return\_isbns ------| |  
 | |-- 11. format\_results() ->| |  
 | |<-- 12. return\_gallery\_data| |  
 |<-- 13. update\_gallery()-------------------------| |  
 | | | | |

Flow Description:

1-3. The Reader interacts with the UI, which calls the backend function.

4-7. The backend gets an embedding for the query from the OpenAI API.

8-10. The backend uses the embedding to search the Vector DB and gets back a list of ISBNs.

11-13. The backend formats the data for the UI, which then updates to show the results to the Reader.

[Page 27]

#### **3.5.2. Filtered Search Sequence**

This diagram shows the sequence for a search that includes category and tone filters.

Actor: Reader UI (Gradio) Backend (Python) Book Data (Pandas)  
 | | | |  
1. enter\_query("...") -->| | |  
2. select\_filters() --->| | |  
3. click\_button() ------>| | |  
 |-- 4. recommend("...", filters)-->| |  
 | | (Semantic search happens first - See 3.5.1) |  
 | |-- 5. apply\_filters(isbns, filters) -->| |  
 | | |-- 6. filter\_and\_sort() ->|  
 | | |<-- 7. return\_dataframe -|  
 | |<-- 8. return\_dataframe---| |  
 | |-- 9. format\_results() -->| |  
 | |<-- 10. return\_gallery\_data| |  
 |<-- 11. update\_gallery()------------------------| |  
 | | | |