Lipisha Gupta (RVCE24MIT005)
Varada S Naik (RVCE24MIT011)

GRANTHA – BOOK RESOURCE AGGREGATOR

Objective

- **Comprehensive Book Search** – Enable users to search for books based on title, author, or genre, ensuring quick and accurate results.
- **Book Aggregation & Resource Enrichment** – Retrieve and display additional resources related to searched books, including PDFs, YouTube videos, and other relevant materials.
- **Spelling Variation Handling** – Implement intelligent search handling for spelling variations, especially for Indian native language words, to improve search accuracy.
- **Search Suggestion Feature** – Provide real-time suggestions based on database records, including author names, genres, and book titles, enhancing the user experience.
- **Enhanced Book Accessibility** – Help users discover and access a wide range of books by aggregating multiple sources into a single platform.

Technologies & Tools Used

Frontend:

- **ReactJS** – Provides a dynamic and responsive user interface for smooth navigation and interaction.
- **Tailwind CSS** (or other UI framework) – Enhances UI styling with modern and minimalistic design.

Backend:

- **Spring Boot** – Ensures efficient, scalable, and high-performance backend processing.
- **Spring Security** – Manages authentication and user access securely.
- **RESTful APIs** – Facilitates seamless communication between the frontend and backend.

Database:

- **PostgreSQL** – Stores and manages book-related data efficiently.

Third-Party API Integrations:

- **Google Books API** – Fetches book details, availability, and metadata.
- **Open Library API** – Provides additional book-related information.
- **YouTube API** – Retrieves related videos, such as reviews, summaries, and audiobooks.
- **Podcast API** – (If applicable) Aggregates book-related podcasts.

Other Tools & Technologies:

- **Spring Data JPA** – Simplifies database interactions.
- **Docker** (if used) – Containerizes the application for deployment.
- **Postman** – Used for API testing and development.
- **Git & GitHub/GitLab** – Version control and collaborative development.

Agile Methodology

Sprint 1: Backend Development – PDF Service

- Designed and implemented the **PDF availability service** to fetch book-related PDFs from external sources.
- Integrated the backend with APIs like **Google Books API and Open Library API** to retrieve book metadata.
- Stored and managed book-related data using **PostgreSQL**.

Sprint 2: Backend Development – Additional Services

- Extended the backend to support fetching **YouTube videos** related to books (e.g., reviews, summaries, audiobooks).
- Developed APIs to provide enriched search results by integrating **YouTube API and Podcast API**.
- Implemented **search suggestion features** based on book titles, authors, and genres.

Sprint 3: Testing and Refinement

- Conducted unit and integration testing using **JUnit and Postman** to validate backend services.
- Ensured API responses were optimized for performance and accuracy.
- Identified and fixed bugs before frontend development began.

Sprint 4: Frontend Development

- Developed the **ReactJS-based frontend** with a user-friendly UI and intuitive search functionality.
- Implemented **real-time search suggestions** for enhanced user experience.
- Styled the UI using **Tailwind CSS** (or other UI frameworks).

Sprint 5: Integration & Deployment

- Integrated the **frontend with the backend** via RESTful APIs.
- Performed **end-to-end testing** to ensure seamless communication between the frontend and backend.
- Deployed the system and ensured smooth functioning across different devices.

Sprint 6: Spelling Variation Handling Feature

- Implemented intelligent **spelling variation handling** for Indian native language words to improve search accuracy.
- Optimized database queries and search algorithms to refine search results.
- Conducted final testing to ensure all features worked efficiently.

Implementation of Word Variations Feature

Fuzzy Matching

To enhance the search functionality in **Grantha - The Book Resource Aggregator**, **Fuzzy Matching** was implemented to handle **spelling variations, typos, and alternate word forms**, particularly for **Indian native language words**. This ensures that users can find books even if they misspell a title or enter a slightly different version of a book's name.

How Fuzzy Matching Works

1. **Levenshtein Distance for Approximate Matching**
   - The **Levenshtein Distance algorithm** is used to measure how different two words are based on the number of changes (insertions, deletions, substitutions) required to transform one word into another.
   - If the difference (distance) between the user's input and a stored book title is within a certain limit, it is considered a match.

2. **Applying Fuzzy Matching in the Search Process**
    - When a user enters a query, the backend checks for **exact matches** first.
    - If an exact match is not found, the system looks for **similar words** using the fuzzy matching algorithm.
    - The results are **ranked** based on their similarity score, ensuring that the closest matches appear at the top.
3. **PostgreSQL Trigram Matching for Efficient Database Searches**
    - Since checking **every word** for similarity can be slow for large datasets, **PostgreSQL's pg_trgm extension** is used to improve performance.
    - This enables the database to compare words based on small character sequences (trigrams) and determine their similarity scores.
    - Queries are optimized to return books where the similarity score **exceeds a predefined threshold**.
4. **Integration with Spring Boot**
    - In the backend, the **Spring Boot service layer** processes search queries by applying **fuzzy matching algorithms** before fetching results.
    - The search logic is integrated into **repository queries**, ensuring that only **relevant book titles** are retrieved from the database.
    - The final results are returned to the frontend with **sorted rankings**, ensuring users see the **best matches first**.

Prototype Analysis

The development of **Grantha - The Book Resource Aggregator** followed a structured prototyping approach to ensure a **functional, user-friendly, and scalable** system. Below is the detailed **Prototype Analysis** covering different aspects of the project.

1. Low-Fidelity Prototype

Before starting development, **low-fidelity prototypes** were created to visualize the platform's core functionality.

- **Homepage layout** – A search bar, featured book categories, and recommendations.
- **Search results page** – Displaying books with additional resources (PDFs, videos).
- **Book details page** – Showing book metadata, related videos, and external access links.
- **User interaction flow** – How users search, filter, and navigate book results.

2. High-Fidelity Prototype (UI/UX Design & Functional Testing)

After refining the wireframes, high-fidelity prototypes were built using UI design tools to **simulate the actual user experience**.

- Used **Figma/Adobe XD** to design interactive UI mockups.

- Implemented **color schemes, typography, and component layouts** for a polished look.
- Simulated **real search interactions** to test the effectiveness of the design.

3. Functional Prtotype

The first working prototype focused on implementing the **core functionalities**:

- **Backend Development (Spring Boot & PostgreSQL)** – Initial implementation of book search, API integrations (Google Books, YouTube, Open Library).
- **Frontend Development (ReactJS)** – Built a simple UI for testing search functionality.
- **API Testing** – Verified the integration of external book sources and resource retrieval.
- **Search Optimization** – Implemented **Fuzzy Matching** to handle spelling variations.

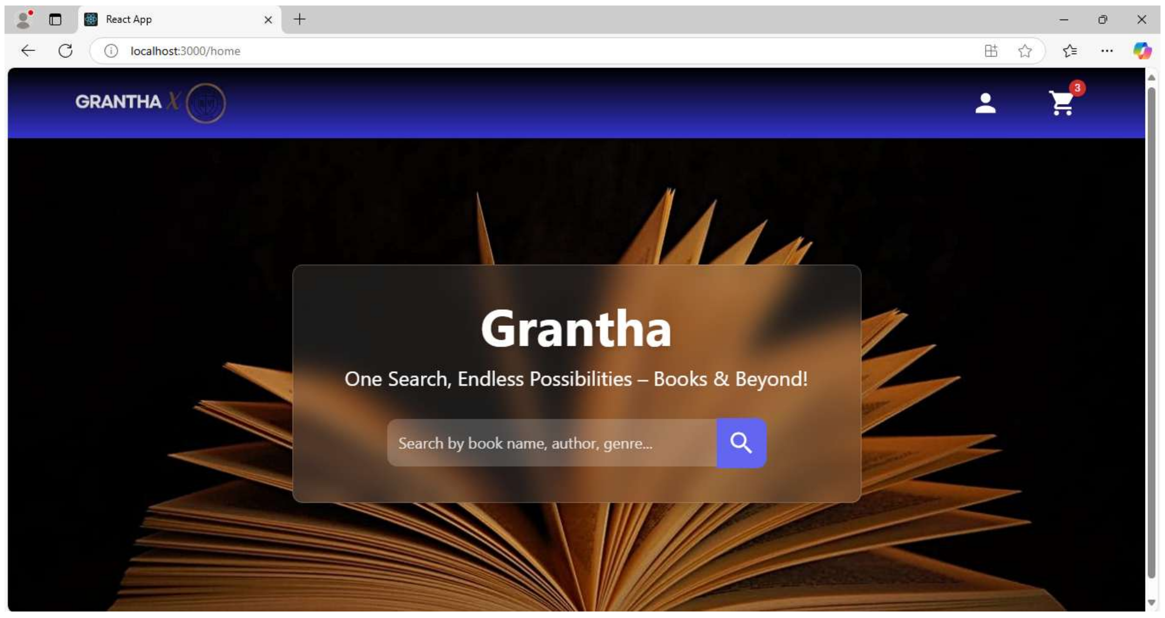
4. Final Prototype

After multiple iterations and improvements, the final prototype was tested with all features integrated:

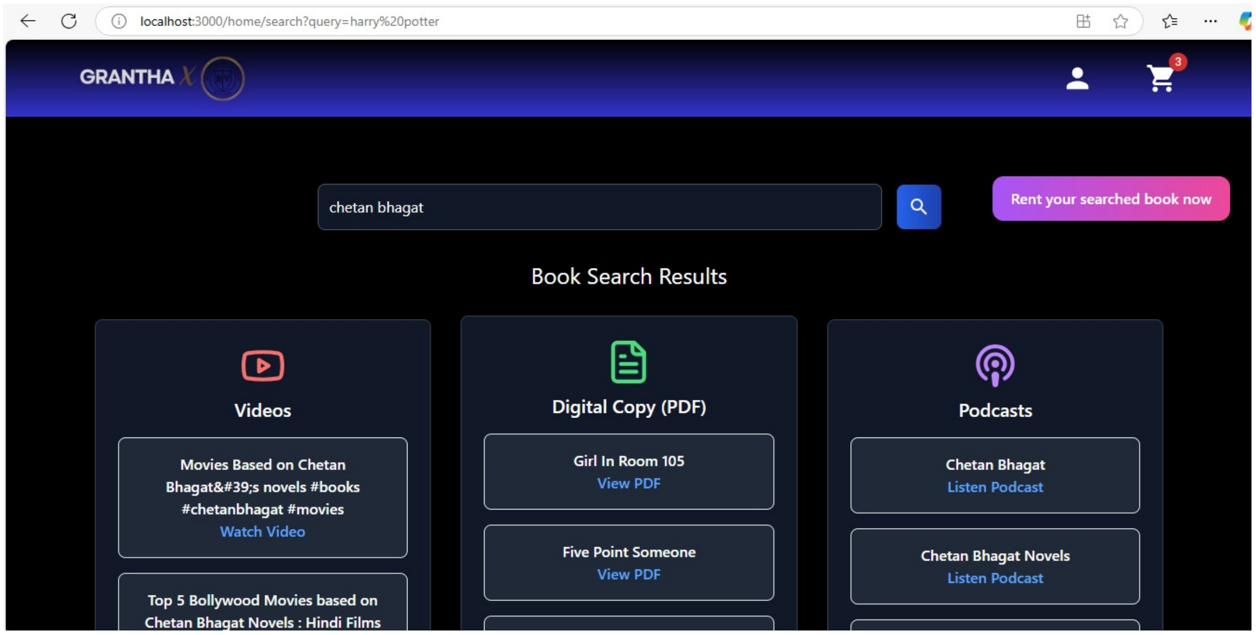
- **Frontend & Backend integration** for real-time book search.
- **Search suggestion & fuzzy matching** enabled for typo handling.
- **Resource enrichment** with PDFs, YouTube videos, and metadata retrieval.
- **Performance Testing** to optimize database queries and API response times.
- **UI Enhancements** for better navigation, accessibility, and responsiveness.

Final Prototype

Home Page


React App
localhost:3000/home
GRANTHA
Grantha
One Search, Endless Possibilities – Books & Beyond!
Search by book name, author, genre...

Search by Author



localhost:3000/home/search?query=harry%20potter

chetan bhagat

Book Search Results

Videos

Movies Based on Chetan Bhagat&#39;s novels #books #chetanbhagat #movies
Watch Video

Top 5 Bollywood Movies based on Chetan Bhagat Novels : Hindi Films

Digital Copy (PDF)

Girl In Room 105
View PDF

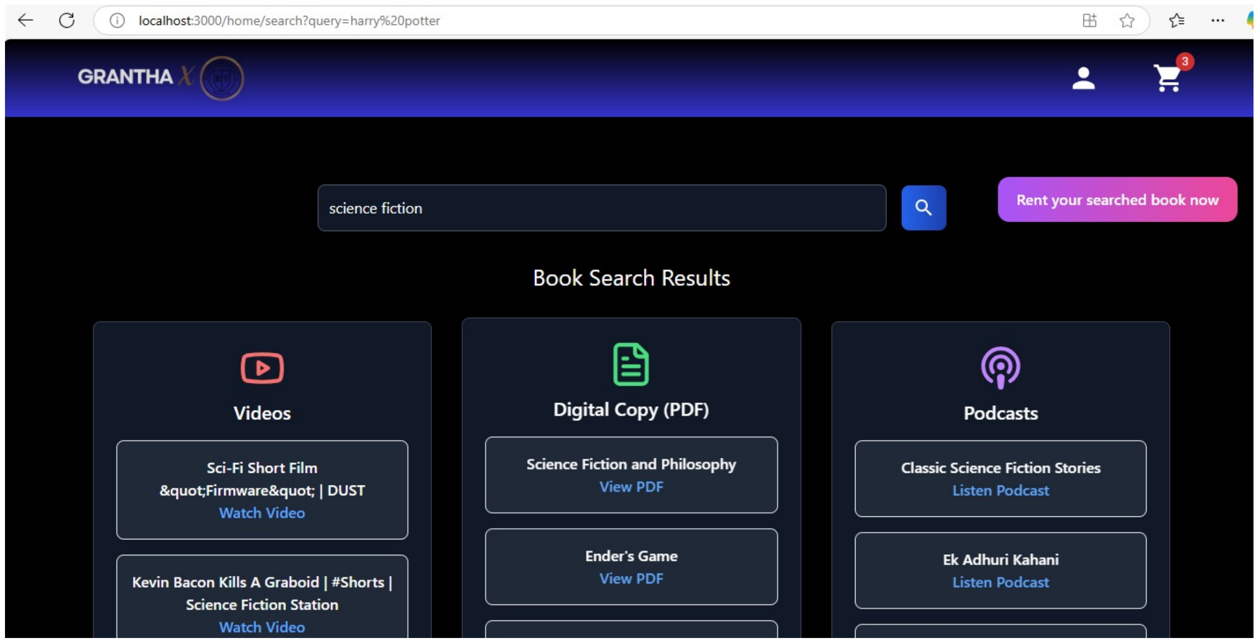
Five Point Someone
View PDF

Podcasts

Chetan Bhagat
Listen Podcast

Chetan Bhagat Novels
Listen Podcast

Search by Genre



localhost:3000/home/search?query=harry%20potter

science fiction

Book Search Results

Videos

Sci-Fi Short Film &quot;Firmware&quot; | DUST
Watch Video

Kevin Bacon Kills A Graboid | #Shorts | Science Fiction Station
Watch Video

Digital Copy (PDF)

Science Fiction and Philosophy
View PDF

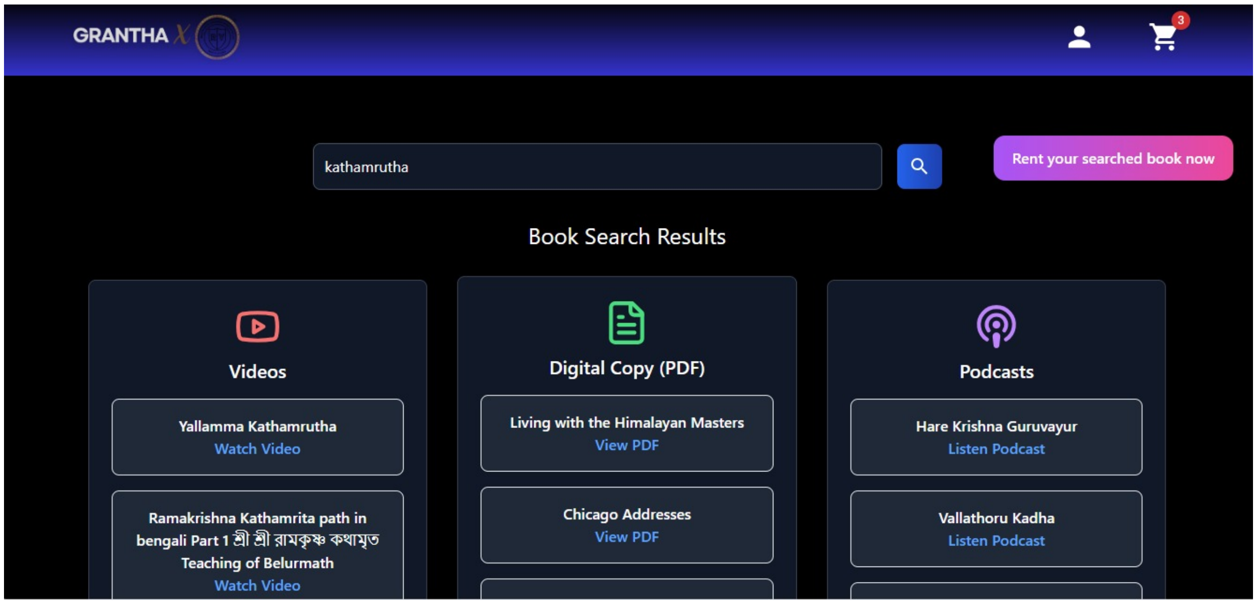
Ender's Game
View PDF

Podcasts

Classic Science Fiction Stories
Listen Podcast

Ek Adhuri Kahani
Listen Podcast

Word variation Feature



kathamrutha

Book Search Results

Videos

Yallamma Kathamrutha
Watch Video

Ramakrishna Kathamrita path in bengali Part 1 শ্রী শ্রী রামকৃষ্ণ কথামৃত Teaching of Belurmath
Watch Video

Digital Copy (PDF)

Living with the Himalayan Masters
View PDF

Chicago Addresses
View PDF

Podcasts

Hare Krishna Guruvayur
Listen Podcast

Vallathoru Kadha
Listen Podcast

Results and Inference

- Comprehensive Book Aggregation – Successfully retrieves books, PDFs, YouTube videos, and metadata from Google Books API, Open Library, and YouTube API, offering enriched book-related resources.
- Enhanced Search Accuracy – Fuzzy Matching in Spring Boot handles typos and spelling variations, especially for Indian native language words, ensuring relevant search results.
- Optimized Performance & Scalability – PostgreSQL Trigram Matching improves search efficiency, while API-driven architecture allows real-time data retrieval and future expansion.
- User-Friendly Interface & Engagement – ReactJS frontend provides a responsive UI, while real-time search suggestions improve usability and user experience
- Efficient System Handling & Future Growth – Handles large datasets smoothly, supports multimedia learning integration, and allows for future AI-driven recommendations.

Conclusion

Grantha - The Book Resource Aggregator successfully enhances book discovery by integrating Google Books API, Open Library API, and YouTube API, providing users with books, PDFs, and related multimedia resources. The implementation of Fuzzy Matching in Spring Boot ensures accurate search results, even with spelling variations or typos, making the platform more user-friendly.

The system is highly scalable, with an optimized PostgreSQL backend, efficient API integration, and a responsive ReactJS frontend for seamless navigation. Real-time search suggestions further improve the user experience by making searches more intuitive.

With its robust performance, scalability, and enriched book resources, Grantha lays the foundation for future enhancements, such as AI-driven recommendations and additional book sources, ensuring a more comprehensive and engaging reading experience.