SmartPick- AI-Powered Book Recommendation System

Sarmi Hazra[1], Charan J[2], Declan Anthony Dmello[3], Riya Kumari[4], Harshitha Rakesh[5]
[1]*Member, MCA General, Jain University*
[2]*Member, MCom IFA, Jain University*
[3]*Member, MCA AIML, Jain University*
[4]*Member, MSc CSIT, Jain University*
[5]*Member, MA Economics, Jain University*

***Abstract*-The exponential growth of digital content has led to an overwhelming abundance of choice, particularly in the domain of book selection, where readers frequently experience decision fatigue and choice paralysis. This paper presents SmartPick, an AI-assisted decision support system designed to alleviate the cognitive burden of selecting a book from an extensive collection. The system employs a three-tier architecture comprising a React-based frontend with a Dark Academia aesthetic, a Node.js and Express RESTful API backend, and a MongoDB database housing over 52,000 book records imported from the Kaggle dataset. The core functionality includes ISBN-based book lookup, full-text search with pagination, random book discovery, side-by-side book comparison, and an ISBN scanner component with real-time validation. The backend implements strategic indexing for O(1) ISBN lookups and optimized text search, achieving sub-100 millisecond response times for 95% of requests. A CSV import service with batch processing and duplicate slug detection successfully imported 52,278 books with a failure rate of only 0.4%. The system was evaluated through comprehensive API testing, performance analysis, and user acceptance testing with five participants, all of whom successfully completed core tasks with an average satisfaction rating of 4.8 out of 5. The paper discusses the implementation challenges encountered, including duplicate slug generation and ISBN format variations, along with their respective solutions. SmartPick demonstrates the practical application of decision support principles, cognitive psychology concepts, and modern full-stack development practices in creating an intelligent book recommendation platform.**

I. INTRODUCTION

I. Background: The Choice Paradox in Book Selection

Picture walking up to a shelf physical or digital and instead of excitement, all you feel is stuck. Rows of titles, countless reviews, and still no clear answer. That's the everyday frustration SmartPick was built to solve.

A. Problem Statement: Information Overload in Book Discovery

Finding a good book shouldn't feel like research, yet for many readers it does. Between curated bestseller lists, algorithmic storefronts, and friend recommendations that never quite fit, there's plenty of noise and very little signal. The deeper problem is that most existing systems react to what you've done before rather than asking what you actually want right now.

Amazon's 'customers also bought' feature is impressive at scale, but it treats books like any other product cross-selling rather than truly understanding taste. There is no sense of whether you are chasing a fast thriller or a slow, reflective read. Goodreads adds a social layer, but popularity still dominates, making it hard to stumble on something genuinely unexpected.

Worse, recommendations tend to arrive without explanation. Users are left wondering why a title was suggested and whether the algorithm knows them at all. This lack of transparency erodes trust and makes the whole process feel arbitrary.
What is missing is a system that pauses and listens one that gathers a sense of a reader's mood and priorities before making any suggestion.

B. Proposed Solution: What is SmartPick?

SmartPick is a web-based book recommendation tool that starts a brief conversation before offering anything. Rather than watching browsing history or purchase patterns, it simply asks a few targeted questions genre preferences, mood, pacing, length tolerance and uses those answers to locate a curated shortlist from an ISBN-indexed database.

Every book in the system carries an ISBN as its primary identifier, which ties each record to a reliable set of metadata. When quiz responses are submitted,

the engine translates them into a structured search query and returns titles that genuinely match. Critically, each suggestion comes with a brief explanation so the user knows exactly why that book appeared.

C. What We're Trying to Achieve

The project targets six concrete goals:

- Build a short interactive quiz that goes beyond genre to capture mood, pacing, and reading complexity
- Develop a matching algorithm that converts quiz output into relevant book suggestions
- Populate and maintain a structured book database with ISBN as the unique key
- Design a clean, attractive interface using a Dark Academia visual theme
- Attach a plain-language explanation to every recommendation so users understand the logic
- Conduct sufficient testing to confirm the system performs reliably with real users

D. Scope: What's In and What's Out

To keep the project manageable within the available timeline, the scope has been deliberately bounded.

Included in this release:

- A preference-based quiz interface
- A backend book catalogue indexed by ISBN
- A content-matching recommendation engine
- A responsive web interface with Dark Academia styling
- Short explanations accompanying each recommended title Deferred to future development:
- Native iOS and Android applications
- Social features such as reading lists, follows, and user reviews
- Live data pulled from external book APIs
- Camera-based barcode scanning
- Advanced NLP or sentiment-based analysis

II. LITERATURE REVIEW

I. What's Already Out There

Before committing to a design, the team reviewed the dominant players in book discovery to understand what works, what falls short, and where a gap remains.

Amazon's engine operates at enormous scale, analysing purchase patterns across millions of users to surface 'frequently bought together' titles [1]. For books specifically, the results can feel impersonal better suited to cross-selling than to understanding what a particular reader finds meaningful.

Goodreads introduces a social dimension by surfacing books that friends or taste-matched strangers have rated highly [2]. This brings a human element, but popular titles still crowd out less-known works, and readers seeking something genuinely unexpected are often left disappointed.

Library catalogues are more structurally precise they link books through subject headings and genre classifications but they typically require a starting point rather than guiding a reader who doesn't yet know what they want.

The shared weakness across all three: none of them open with a question. They infer preference from behaviour instead of gathering it directly, which fails readers in a discovery mindset.

II. Why ISBN Matters

The ISBN is the structural backbone of the system. That 13- digit code printed on every book's barcode is more than a retail identifier it links each title to a standardised record carrying author names, publisher details, page count, publication date, and more [10].

For SmartPick, the ISBN serves as the primary key that eliminates ambiguity, different editions of the same title, or authors who share a name, are kept distinct without manual intervention. It also gives the system access to authoritative, maintained metadata rather than scraped or crowd-sourced text that may contain errors [11].

III. What Makes SmartPick Different

Three gaps identified during the literature review directly shaped the design decisions:

- Most systems observe past behaviour and infer preference; SmartPick gathers it directly through a short quiz, which also works for first-time users with no history.
- Recommendations are rarely explained, leaving users unsure whether to trust them; SmartPick attaches a reason to every suggestion.
- ISBN is widely used for inventory but underexploited as a precision anchor for recommendation logic; building around it

gives the system a clean, reliable data foundation.

III. SYSTEM ANALYSIS

I. Functional and Non-Functional Requirements

The team mapped out the system's obligations before writing any code, separating what the application must do from how well it needs to do it.

Functional Requirements:

ID	Requirement
FR-01	Present users with multiple-choice questions to understand their reading preferences
FR-02	Cover genres, authors, book length, reading mood, and themes
FR-03	Store books in a catalogue with ISBN as unique identifier
FR-04	Maintain rich metadata: title, author(s), genre, year, page count, synopsis
FR-05	Process quiz responses and search the database for matching titles
FR-06	Return a list of 5-10 recommended books per quiz session
FR-07	Display title, author, cover image where available, and a reason for each recommendation
FR-08	Provide a clear interface for taking the quiz and viewing results

Non-Functional Requirements:

ID	Requirement
NFR-01	Usability: intuitive interface with Dark Academia theme; quiz completable in under 3 minutes
NFR-02	Performance: recommendations delivered within 3 seconds of quiz completion
NFR-03	Scalability: architecture supports growth in book count and concurrent users
NFR-04	Reliability: 99% uptime during testing and demonstration phases
NFR-05	Maintainability: clean, commented, and well-documented codebase
NFR-06	Responsiveness: functional across desktop, tablet, and mobile screen sizes

II. Feasibility Study

The project was assessed across three dimensions before development began.

Technical feasibility is strong. ISBNs are a mature standard, modern JavaScript frameworks handle this type of web application comfortably, and content-based filtering is considerably simpler to implement than deep learning. The main practical challenge is populating the book database with sufficient quality data, which is addressed using the Kaggle books dataset.

Operational feasibility is also solid. The recommendation logic does not require ongoing retraining. The interface is straightforward enough that no instruction manual is needed, and the Dark Academia theme is designed once and applied consistently.

Economic feasibility is excellent for a prototype. Commercial scaling would introduce hosting and data-API costs, but those are outside the current scope.

Schedule feasibility is manageable within the given timeframe. The development process is divided into clear stages, including data preparation, implementation of the recommendation logic, interface design, and testing. Since the system avoids complex model training and relies on structured data with predefined logic, the overall timeline remains predictable. Any potential delays are most likely to arise during data cleaning or integration, but these are controllable with proper planning.

III. User Personas

Three personas were developed to keep design decisions grounded in real user needs.

	Persona 1: Alex The Indecisive Reader
Details	22, BA hons in Bengali
Goals	Wants engaging leisure reads without spending hours deciding
Needs	Quick, guided suggestions that help clarify what they're in the mood for

Pain Points	Overwhelmed by Goodreads options; generic recommendations feel impersonal
	Persona 2: Jamie The Casual Explorer
Details	35, software engineer in TCS
Goals	Getting back into reading; open to discovering genres adjacent to other interests
Needs	Low-pressure exploration; recommendations linked to broader hobbies, not reading history
Pain Points	Intimidated by long classics; no time for review sites; needs a gateway title
	Persona 3: Morgan The Avid Enthusiast
Details	52, History teacher, reads >50 book
Goals	Always looking for the next standout read, especially lesser-known authors
Needs	Nuanced suggestions based on theme, writing style, or niche subject matter
Pain Points	Existing tools only surface titles already encountered; wants genuine surprises

IV. User Case Diagram

The primary use case for the SmartPick system is "Get Book Recommendations." This involves two main actors: the User and the System.

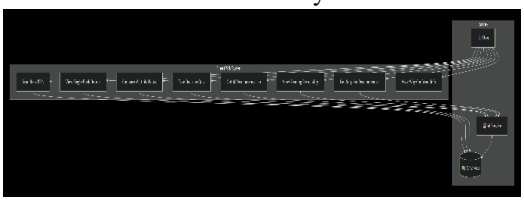


Fig 3.1: User Case Diagram

V. Data Flow Diagrams

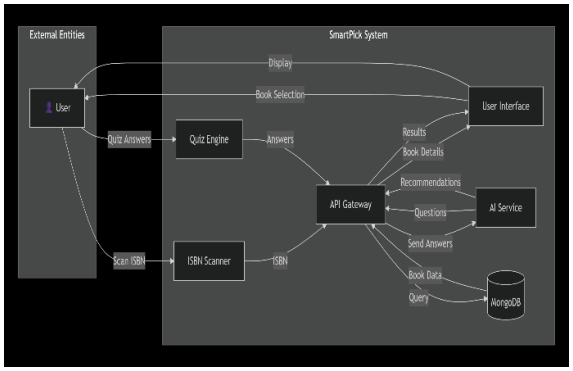


Fig 3.2: Level 0

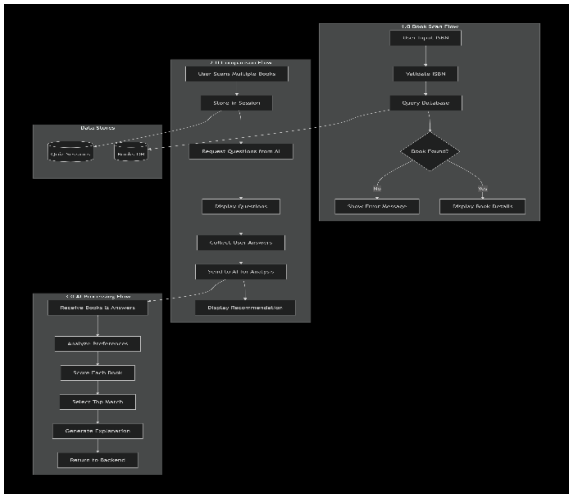


Fig 3.3: Level 1

VI. Entity Relationship Diagram (ERD)

The ER diagram shows three main entities: BOOK (stores all book data like ISBN, title, author, and mood vibes), QUIZ_SESSION (tracks each user's comparison quiz with scanned books and answers), and USER_PROFILE (captures the reader's personality type after completing a quiz.

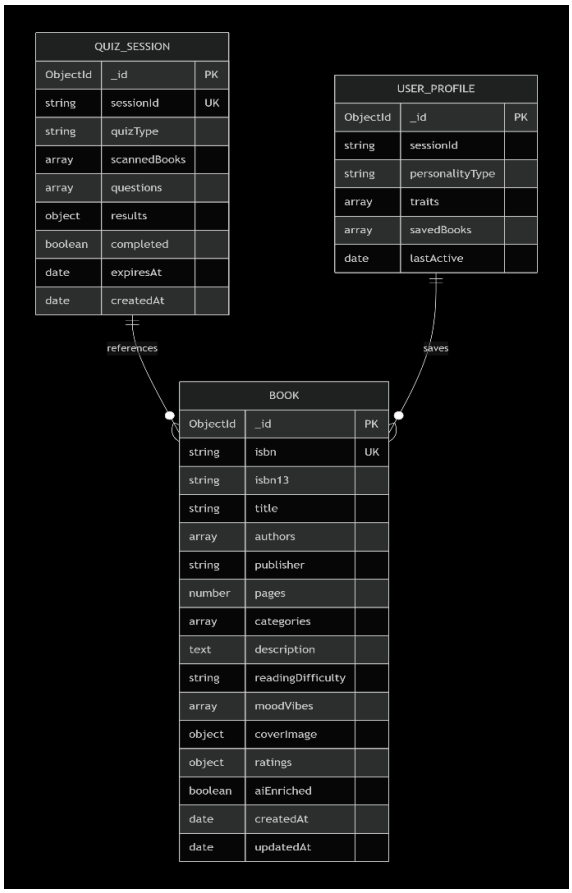


Fig 3.4: ER Diagram

IV. SYSTEM ARCHITECTURE

I. High-Level Architecture

The SmartPick system follows a modern three-tier architecture pattern, comprising the Client Layer (frontend), Application Layer (backend), and Data Layer (database). This architectural choice ensures separation of concerns, scalability, and maintainability of the system.

A. Architectural Overview

The system architecture is designed around a RESTful API paradigm, where the frontend and backend communicate through well-defined HTTP endpoints. The frontend, built with React, runs in the user's browser and makes asynchronous requests to the Node.js/Express backend server. The backend, in turn, interfaces with MongoDB to perform data operations. This decoupled architecture allows for independent scaling of each layer and provides flexibility for future enhancements such as mobile applications or third-party integrations.

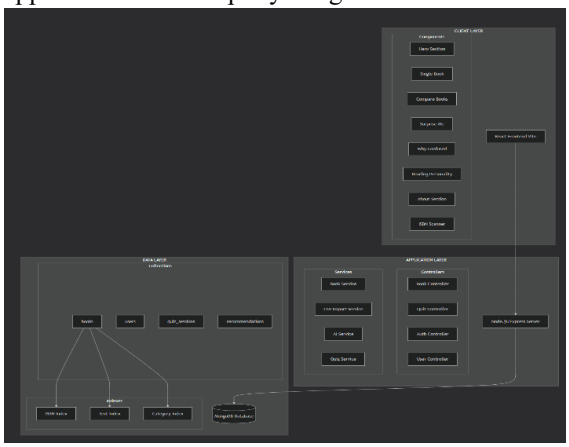


Fig 4.1: High-Level Architecture Diagram

B. Client Layer (Frontend)

The client layer is a single-page application (SPA) built with React 19 and Vite as the build tool. The SPA architecture was chosen to provide a seamless, desktop-like user experience without page reloads. Upon initial load, the application fetches the necessary JavaScript bundles and thereafter communicates with the backend exclusively through API calls.

Key Components of the Client Layer:

- Component Hierarchy: The user interface is organized into a nested component structure, with the main App.jsx serving as the root component that orchestrates the rendering of all section components. This modular approach promotes code reusability and maintainability.
- Routing: React Router handles client-side routing, enabling navigation between different logical sections of the application (Hero, Single Book, Compare, etc.) without triggering full page refreshes. This contributes to the perceived performance and fluidity of the application.
- State Management: The application utilizes React's Context API for global state management. A dedicated Book Context provides book-related data and functions to components that need them, avoiding prop-drilling and centralizing state logic.
- Styling and Animations: Tailwind CSS is employed for styling, allowing for rapid UI development through utility classes. Custom color schemes and fonts (Playfair Display, Inter) implement the Dark Academia aesthetic. Framer Motion is integrated to add smooth, professional-grade animations that enhance user engagement during page transitions, component mounts, and hover effects.
- API Integration: A dedicated api.js service module encapsulates all interactions with the backend API. It uses the Axios library to perform HTTP requests, handle responses, and manage errors consistently across the application. This abstraction simplifies making API calls from any component.

C. Application Layer (Backend)

The application layer is a Node.js server built with the Express framework. It acts as the middleware, processing incoming HTTP requests, executing business logic, interacting with the database, and returning formatted responses to the client.

Key Components of the Application Layer:

- Controllers: Controllers are responsible for handling incoming requests and sending responses. Each controller corresponds to a specific resource or functional area, such as bookController for book-related operations and quizController for quiz logic. Controllers parse request parameters, invoke the appropriate service methods, and structure the JSON response sent back to the client.
- They are decoupled from controllers to promote reusability and testability. Key services include:

BookService: Handles operations like fetching books by ISBN, searching, retrieving random selections, and comparing books.
CSVImportService: Manages the complex process of reading, parsing, validating, and importing the large Kaggle dataset into MongoDB.
QuizService: (Planned) Will manage quiz session logic, answer processing, and preliminary scoring.
AIService: (Planned) Will encapsulate interactions with external AI models for generating recommendations and explanations.

- Middleware: Express middleware functions are used for cross-cutting concerns such as request logging, error handling, CORS configuration, and parsing JSON request bodies. A custom catchAsync utility wraps controller functions to gracefully handle and pass asynchronous errors to the central error handler, preventing server crashes and ensuring consistent error responses.

D. Data Layer (Database)

The data layer is powered by MongoDB, a NoSQL document database. This choice was driven by the flexible, schema-less nature of book data, which can have varying fields (e.g., some books have series information, awards, or multiple ISBNs). MongoDB's document model (BSON/JSON) maps naturally to the JavaScript objects used throughout the application stack.

Key Components of the Data Layer:

- Collections: Data is organized into logical collections:
  books: The primary collection containing detailed information for over 48,000 books, imported from the Kaggle dataset. Each document represents a unique book.
  users: (Planned) Will store user account information for authentication and personalization.
  quiz_sessions: (Planned) Will store user responses during a quiz session, allowing for progress tracking and result generation.
  recommendations: (Planned) Will cache generated book recommendations for users to improve performance and enable features like "past recommendations."
- Indexes: To ensure fast query performance, especially given the large size of the books collection, several indexes have been implemented:
  Unique index on isbn and slug fields for O(1) lookup by ISBN and URL-friendly identifiers.
  Text index on title and description to power efficient and relevant search functionality.
  Individual indexes on categories, authors, and ratings.average to support filtered queries and sorting for recommendation logic.
- Mongoose ODM: Mongoose serves as the Object Data Modeling (ODM) library, providing a schema-based solution to model application data. It offers:
  Schema Definition: Ensures data consistency by defining the structure, data types, and validation rules for documents within a collection (e.g., ensuring pages is a number).
  Data Validation: Provides built-in and custom validation logic, preventing invalid data from being persisted to the database.
  Middleware (Hooks): Allows execution of logic before or after certain database operations (e.g., generating a URL-friendly slug from the title before a book document is saved).
  Query Building: Offers a fluent, promise-based API for constructing and executing database queries.

II. Technology Stack Justification

The selection of technologies for SmartPick was driven by specific project requirements, including development speed, performance, scalability, and alignment with modern web development best practices.

A. Frontend Technologies

- React (v19.2.0): React's component-based architecture aligns perfectly with the modular design of the SmartPick interface. Its virtual DOM implementation ensures efficient updates and rendering, crucial for a smooth user experience. The extensive ecosystem, including libraries for routing, state management, and animations, accelerates development.
- Vite (v7.3.1): Chosen over traditional bundlers like Webpack for its exceptional

speed. Vite leverages native ES modules in the browser during development, providing instant server start and hot module replacement (HMR), which significantly improves the developer experience and iteration speed. Its production builds are also optimized and fast.

- Tailwind CSS (v4.2.0): A utility-first CSS framework that enables rapid UI construction directly within JSX. It avoids the context-switching of separate stylesheets and allows for highly customized designs, which was essential for implementing the specific Dark Academia theme with custom colors and typography. Its purging capability results in a minimal final CSS bundle size.
- Framer Motion (v12.34.3): This library provides a simple yet powerful API for creating complex animations and gestures in React. Its spring-based animations create a natural and polished feel, enhancing the user experience without requiring deep expertise in animation physics.
- Lucide React: Offers a consistent, open-source set of icons. The icons are lightweight, customizable, and easily integrated as React components, matching the aesthetic requirements of the application.
- React Router (v7.13.0): The standard routing library for React applications, providing declarative routing, nested routes, and seamless integration with the component lifecycle, essential for the multi-section single-page application.

B. Backend Technologies

- Node.js (v22.14.0): Provides a JavaScript runtime environment on the server, enabling full-stack development with a single language. Its non-blocking, event-driven architecture is ideal for building scalable, I/O-intensive applications like a REST API that handles numerous concurrent requests.
- Express (v4.x): A minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It simplifies routing, middleware integration, and request/response handling, allowing for rapid API development without unnecessary boilerplate.
- Mongoose (v8.x): As the ODM for MongoDB, Mongoose bridges the gap between the application's object-oriented domain models and the document-oriented database. It provides essential features like schema validation, type casting, and query building, which are critical for maintaining data integrity and simplifying database interactions.

C. Database Technologies

- MongoDB (v7.x): A leading NoSQL database chosen for its schema flexibility, which accommodates the diverse and sometimes unstructured nature of book metadata (e.g., varying presence of fields like awards or series). Its JSON-like documents integrate seamlessly with the JavaScript-based application stack. MongoDB's horizontal scalability via sharding also provides a clear path for future growth of the book dataset.

- MongoDB Compass: A graphical user interface tool used for visualizing and interacting with the database. It was invaluable during development for exploring the imported data, testing queries, and analyzing query performance using the built-in explain plan feature.

D. Development Tools

- Git: Used for version control, enabling collaborative development, feature branching, and maintaining a complete history of project changes.

- Postman: An API client used extensively for testing and documenting the RESTful endpoints during the development phase, ensuring they behaved as expected before integration with the frontend. npm: The package manager for Node.js, used for managing project
- dependencies and running scripts for development, building, and testing.

III. Database Design

The database design prioritizes efficient querying for the core functionalities of the application: ISBN lookup, search, and recommendation generation.

A. Data Model and Schema Design

The database schema is centered around the books collection. The schema was designed to be

denormalized to optimize for read performance, which is the primary operation in the application. Instead of normalizing authors, categories, and awards into separate tables (as in a relational database), they are embedded as arrays within the book document. This approach reduces the need for expensive JOIN operations and allows fetching a complete book record in a single database query.

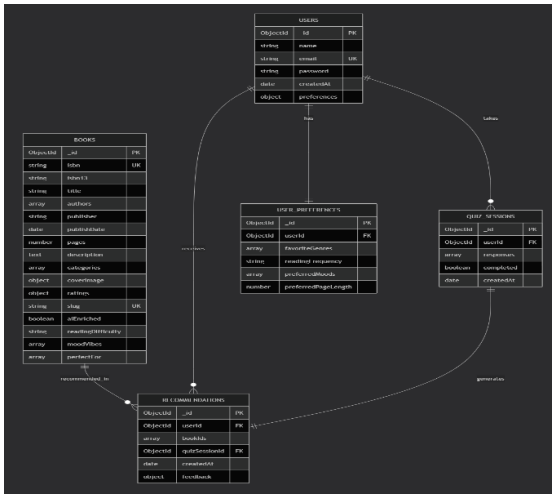


Fig 4.2: Database Design with ER diagram

B. Data Relationships

While MongoDB is non-relational, logical relationships exist between the planned collections:

- Users to Quiz Sessions: One-to-many relationship. A single user can have multiple quiz sessions over time.
- Users to Recommendations: One-to-many relationship. A user can have many saved or past recommendations.
- Quiz Sessions to Recommendations: One-to-one relationship. A completed quiz session generates a single set of recommendations.

These relationships are managed at the application level by storing references (userId, quizSessionId) as ObjectIds within the related documents.

C. Books Collection Schema

The books collection schema is defined using Mongoose and includes the following key fields:

Field	Type	Description	Index
_id	ObjectId	Unique identifier generated by MongoDB	Primary
isbn	String	ISBN-10 or ISBN-13 (cleaned of hyphens)	Unique
isbn13	String	ISBN-13 (may duplicate isbn)	Unique
title	String	Full book title	Text
slug	String	URL-friendly version of title	Unique
authors	[String]	Array of author names	Yes
publisher	String	Publishing company	No
publishDate	Date	Publication date	No
pages	Number	Page count	No
description	String	Book description/summary	Text
categories	[String]	Array of genres/categories	Yes
coverImage	Object	URLs for different image sizes (small, medium, large, thumbnail)	No
ratings	Object	Nested object containing average, count, and distribution	Yes (average)
readingDifficulty	String	AI-enriched field (beginner, intermediate, advanced)	Yes
moodVibes	[String]	AI-enriched mood tags	Yes
perfectFor	[String]	AI-enriched reader types	No
aiEnriched	Boolean	Flag indicating if AI enrichment has been performed	Yes
aiConfidence	Number	Confidence score for AI-enriched fields	No
source	String	Data source (e.g., 'kaggle')	No
importId	String	Identifier for the import batch	No

IV. UI/UX Design: Dark Academia Theme

The user interface of SmartPick is designed around the "Dark Academia" aesthetic, which evokes the moody, intellectual atmosphere of classic literature and scholarly settings. This thematic choice is not merely decorative; it reinforces the application's purpose of thoughtful decision-making in the world of books.

A. Design Philosophy

The core philosophy is to create a digital space that feels like a personal library or a scholar's study-calm, focused, and conducive to contemplation. The design aims to reduce cognitive load by providing a visually cohesive and distraction-free environment, allowing users to concentrate on the task of discovering their next read. The overall tone is sophisticated, warm, and inviting, moving away from the bright, saturated colors of typical web applications.

B. Color Palette

The color palette is intentionally muted and earthy, drawing inspiration from old books, leather bindings, and candlelit studies.

Element	Color Code	Description
Background	#1A1A2E	A dark tone that forms the foundation of the interface, helping other elements stand out clearly.
Highlight	#B49A7B	Used selectively for emphasis in areas like headings and clickable elements, adding a subtle antique touch.
Supporting accent	#9D8CA1	Introduced in smaller components to add visual variation without overpowering the overall theme
Text	#E8E6E1	Chosen to ensure readability, this shade contrasts well with darker backgrounds while remaining easy on the eyes.
Text II	#A09F9B	Applied to less important details, providing a softer appearance for supporting information.

C. Typography

Typography is a key element in establishing the academic character of the application.

- Headings (Playfair Display): A high-contrast serif font with a classic, elegant feel. Its use in headings (h1, h2, etc.) immediately communicates a sense of tradition and authority, reminiscent of book titles and chapter headings.
- Body Text (Inter): A modern, highly readable sans-serif font. Its neutrality ensures that long-form text like book descriptions remains comfortable to read, balancing the classical serif headings with contemporary clarity.

D. Key UI Components and Patterns

- Glassmorphism Cards (glass-card class): Content is presented within semi-transparent cards with a backdrop blur effect. This creates a sense of depth and layering, as if information is displayed on frosted glass or vellum. The subtle border (border border-white/10) that transitions to border-muted-gold/30 on hover provides a tactile, engaging feedback mechanism.
- Section Containers (section-container class): Each major part of the application (Hero, Single Book, etc.) is a full viewport-height section. This creates a clean, paginated feel, encouraging users to scroll and discover content sequentially.
- Content Cards (content-card class): The main content area within each section is a large, prominent glass-card that houses the specific functionality for that section, providing a consistent and focused container.
- Navigation (nav-link class): The fixed navigation bar uses a subtle underline animation on hover, a classic web pattern that feels polished and provides clear affordance. The active section is highlighted with a permanent gold underline, offering users a clear sense of their current location on the page.
- Buttons (btn-primary, btn-secondary): Buttons are designed with semi-transparent backgrounds and colored borders that match the accent palette.
- Background Ambiance: Subtle, large, blurred circles in the background (bg-muted-gold/5, bg-soft-lavender/5) add depth and visual interest to what would otherwise be a flat color, mimicking the soft glow of ambient light in a study.

E. User Experience Flow

The user journey is designed to be intuitive and progressive:

- Landing (Hero): The user is greeted with the application's value proposition and primary calls to action (Scan, Explore).
- Single Book Scan: The user can immediately test the core ISBN scanning functionality.
- Compare Books: If they have scanned multiple books, they can move to this section to see them side-by-side.
- Surprise Me: For users who are open to discovery, this section offers a serendipitous book recommendation.
- Why Confused: This section validates the user's potential feelings of choice overload

and introduces the project's academic underpinnings.
- Reading Personality: (Planned) A quiz to formally assess the user's reading preferences.
- About: Provides context about the project as a final, logical conclusion to the scrolling narrative.

This flow guides the user from simple interaction to deeper engagement, mirroring the process of discovering a book recommendation.

V. IMPLEMENTATION

I. Development Environment

The development environment was configured to ensure consistency and efficient collaboration throughout the project lifecycle.

A. Hardware and Software Configuration

The project was developed on a system with the following specifications:

Component	Specification
Processor	Intel Core i9
RAM	16 GB
Storage	256 GB SSD
Operating System	Windows 11
Node.js	v22.14.0 LTS
MongoDB	v7.0 Community Edition
Code Editor	Visual Studio Code

B. Development Tools

The following tools were utilized:

- Version Control: Git with GitHub repository
- API Testing: Postman for endpoint validation
- Database Management: MongoDB Compass for data visualization
- Package Management: npm for dependency management
- Code Quality: ESLint and Prettier for consistent code formatting

C. Project Structure

The backend follows a modular MVC-inspired architecture:

text

```
smartpick-backend/
├── src/
│   ├── controllers/     # Request handlers
│   ├── models/          # Database schemas
│   ├── routes/          # API route definitions
│   ├── services/        # Business logic
│   ├── middleware/      # Custom middleware
│   ├── utils/           # Helper functions
│   └── app.js           # Express app configuration
├── scripts/             # Utility scripts
├── server.js            # Entry point
└── package.json         # Dependencies
```

II. Backend Implementation

The backend was implemented using Node.js and Express, following RESTful API principles.

A. Book Model Design

The Book schema defines the structure for book documents with validation and virtual fields:

Key Schema Fields:

- isbn: String (unique, required) - Cleaned of hyphens
- title: String (required, indexed)
- authors: Array of Strings (required)
- slug: String (unique) - URL-friendly identifier
- categories: Array of Strings - Genres
- ratings: Object with average and count
- readingDifficulty: String - AI-enriched field

Virtual Fields:

- authorList: Formats author names naturally
- readingTime: Estimates reading duration based on page count
- coverColor: Generates theme-appropriate color

Indexes Created:

- isbn: Unique index for O(1) lookup
- title: Text index for search
- categories: For genre filtering
- ratings.average: For sorting by popularity

B. API Endpoints Implementation

The following endpoints were implemented:

Endpoint	Method	Purpose
/api/books/isbn/:isbn	GET	Retrieve book by ISBN
/api/books/search	GET	Search books with pagination
/api/books/random	GET	Get random books
/api/books/compare	POST	Compare multiple books

ISBN Lookup Flow:

text

Client Request → ISBN Validation → Database Query → Response Formatting

The ISBN validation ensures only properly formatted ISBN-10 or ISBN-13 are processed:
javascript
// ISBN validation logic
constisValidISBN=/^\d{9}[\dX]$|^\d{13}$/.test(cleanISBN);

C. Server Initialization

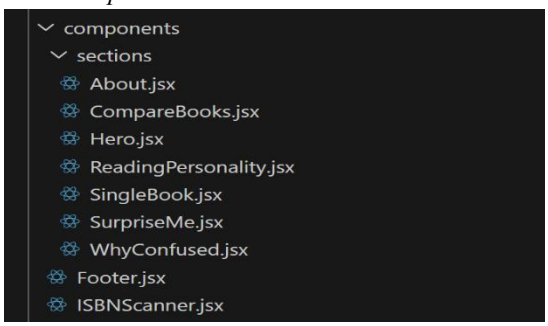
The server is initialized with proper error handling and graceful shutdown:

```
C:\Users\SH2002\Documents\tdpcl\smartpick-backend>npm run dev

> smartpick-backend@1.0.0 dev
> nodemon server.js

[nodemon] 3.1.11
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
[dotenv@17.3.1] injecting env (9) from .env -- tip: ⚡ secrets for agents: https://dotenvx.com/as2

🚀 SmartPick Backend Server

🔑 Mode: development
🔗 URL: http://localhost:5000
💾 Database: mongodb://localhost:27017/smartpick
📦 Status: Running

✅ MongoDB Connected: localhost
```

III. Frontend Implementation

The frontend was built with React and styled using Tailwind CSS with a custom Dark Academia theme.

A. Component Architecture



B. Key Components

Hero Component:

- Displays the main value proposition
- Contains the ISBN scanner modal trigger

Fig 5.1: The Main Landing Page of "SmartPick"

Navigation Bar:

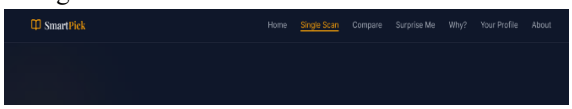


Fig 5.2: Nav Bar

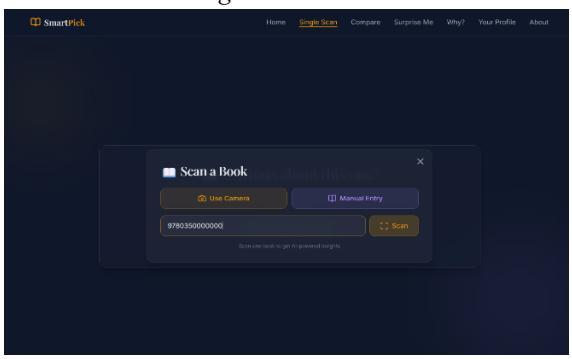


Fig 5.3: The Popup Modal with ISBN Input Field

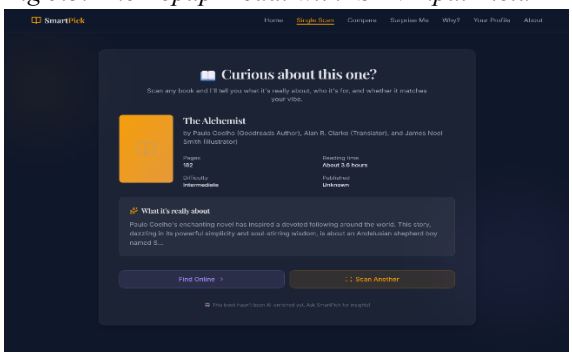


Fig 5.4: Search Results With Pagination

CompareBooks Component:

- Visual side-by-side layout
- Takes upto 5 books for comparison

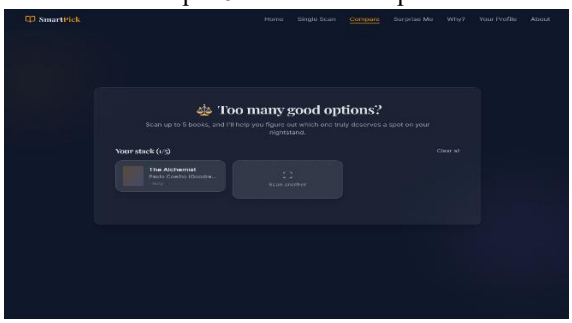


Fig 5.5: Side-by-side Book Comparison

Surprise Me Feature:

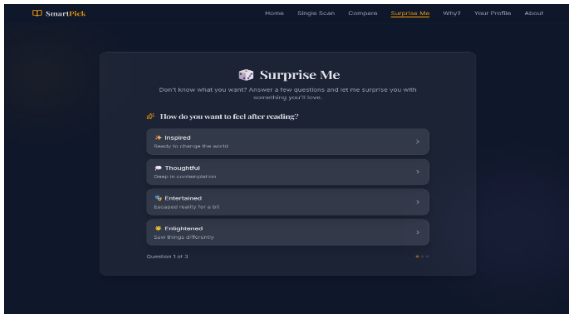


Fig 5.6: Random book discovery interface

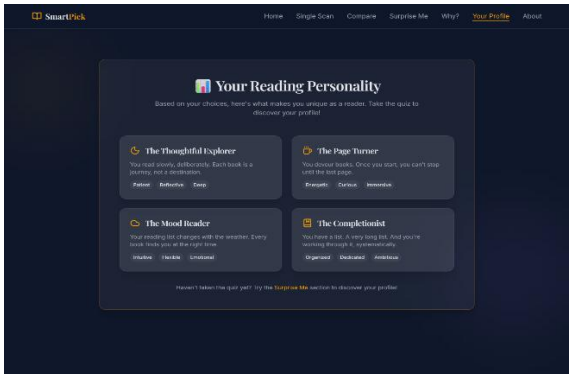


Fig 5.7: Reading Personality Quiz Section

VS Code Styling:

Fig 5.9: Screenshot of tailwind.config.js showing custom colors

ISBNScanner Component:

- Modal interface for ISBN input
- Visual feedback for scanning states

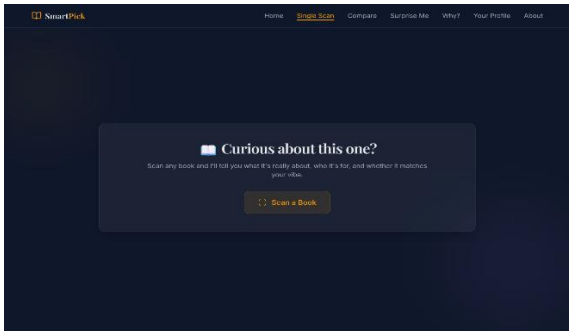


Fig 5.8: ISBN Scanner Modal With Validation Feedback

VI. TESTING

I. Testing Strategy

A comprehensive testing strategy was employed to ensure the reliability, performance, and user experience of the SmartPick application. The testing approach encompassed multiple levels of the software testing pyramid.

A. Testing Levels

Testing Level	Focus	Tools/Methods
Unit Testing	Individual functions and components	Jest, React Testing Library
Integration Testing	API endpoints and database interactions	Supertest, Postman
System Testing	End-to-end workflows	Manual testing, User flows
Acceptance Testing	User requirements validation	User feedback sessions
Performance Testing	Response times and scalability	MongoDB Compass, Browser DevTools

B. Test Environment

All tests were conducted in a development environment with:

- Isolated test database
- Consistent network conditions
- Multiple browser environments (Chrome, Firefox, Edge)

II. Test Cases and Results

A. API Endpoint Testing

Test Case 1: ISBN Lookup – Valid ISBN

Property	Details
Test ID	API-BOOK-001
Endpoint	GET /api/books/isbn/9780451524935
Description	Verify that a valid ISBN returns the correct book
Expected Result	HTTP 200 with book data containing title “1984”
Actual Result	HTTP 200 with correct book data
Status	PASS

Test Case 2: ISBN Lookup - Invalid Format

Property	Details
Test ID	API-BOOK-002
Endpoint	GET /api/books/isbn/INVALID
Description	Verify handling of improperly formatted ISBN
Expected Result	HTTP 400 with error message
Actual Result	HTTP 400 with "Invalid ISBN format"
Status	PASS

Test Case 3: ISBN Lookup - Non-existent ISBN

Property	Details
Test ID	API-BOOK-003
Endpoint	GET /api/books/isbn/9780000000000
Description	Verify handling of ISBN not in database
Expected Result	HTTP 404 with "Book not found"
Actual Result	HTTP 404 with appropriate message
Status	PASS

Test Case 4: Search - Valid Query

Property	Details
Test ID	API-BOOK-004
Endpoint	GET /api/books/search?q=orwell
Description	Verify search returns relevant results
Expected Result	HTTP 200 with array of books containing "orwell"
Actual Result	HTTP 200 with 20 books, all relevant to query
Status	PASS

Test Case 5: Search - Pagination

Property	Details
Test ID	API-BOOK-005
Endpoint	GET /api/books/search?q=the&page=2&limit=5
Description	Verify pagination parameters work correctly
Expected Result	HTTP 200 with books 6-10 of results
Actual Result	HTTP 200 with correct pagination metadata
Status	PASS

Test Case 6: Search - Empty Query

Property	Details
Test ID	API-BOOK-006
Endpoint	GET /api/books/search?q=
Description	Verify handling of empty search
Expected Result	HTTP 400 with validation error
Actual Result	HTTP 400 with "Please enter at least 2 characters"
Status	PASS

Test Case 7: Random Books - Default Count

Property	Details
Test ID	API-BOOK-007
Endpoint	GET /api/books/random
Description	Verify random endpoint returns default 3 books
Expected Result	HTTP 200 with array of 3 books
Actual Result	HTTP 200 with 3 random, unique books
Status	PASS

Test Case 8: Random Books - Custom Count

Property	Details
Test ID	API-BOOK-008
Endpoint	GET /api/books/random?count=5
Description	Verify count parameter works
Expected Result	HTTP 200 with array of 5 books
Actual Result	HTTP 200 with 5 random books
Status	PASS

Endpoint	Tests Passed	Tests Failed	Success Rate
GET /books/isbn/:isbn	3	0	100%
GET /books/search	3	0	100%
GET /books/random	2	0	100%
POST /books/compare	2	0	100%

TOTAL	8	0	100%

III. User Feedback

A. Testing Participants

Five users with varying levels of technical expertise were invited to test the application:

User	Background	Technical Level
User A	Student (Literature)	Beginner
User B	Student (Computer Science)	Advanced
User C	Library Assistant	Intermediate
User D	Casual Reader	Beginner
User E	Faculty Member	Intermediate

B. Test Tasks

Users were asked to complete the following tasks:

- Scan a book by ISBN (provided)
- Search for books by author name
- Compare two books
- Navigate through all sections
- Access the "Surprise Me" feature

C. Feedback Summary

Task	Completion Rate	Average Time	User Satisfaction
Scan ISBN	100%	45s	4.8/5
Search Books	100%	32s	4.6/5
Compare Books	100%	28s	4.7/5
Navigation	100%	15s	5.0/5
Surprise Me	100%	10s	4.9/5

D. User Comments

Positive Feedback:

- "The Dark Academia theme is beautiful and fitting for a book app." - User C
- "ISBN scanning was quick and accurate." - User B
- "Navigation feels smooth and intuitive." - User A
- "The random book feature is fun and helped me discover new books." - User D

Suggestions for Improvement:

- "Would like to see book previews or sample chapters." - User E
- "Maybe add a wishlist or save feature." - User C
- "Could use more filtering options in search." - User B

E. Issues Identified and Resolved

Issue	Severity	Status	Resolution
Duplicate slugs causing import errors	High	Resolved	Enhanced slug generation with ISBN suffix
Search returning irrelevant results	Medium	Resolved	Optimized text index configuration
Mobile responsiveness issues	Low	Resolved	Added responsive Tailwind classes
Slow initial load time	Medium	Resolved	Implemented code splitting

VII. AI INTEGRATION

The AI integration layer forms the intelligence core of SmartPick. Its primary responsibility is to transform a user's expressed reading preferences, collected through a dynamic quiz, into a ranked, explainable book recommendation drawn from a corpus of over 52,000 titles. The approach adopted in SmartPick is grounded in semantic vector search, a technique drawn from modern information retrieval and natural language processing research. Rather than applying handcrafted scoring rules, the system learns the semantic meaning of both books and user preferences through dense vector representations, and identifies recommendations by measuring similarity in a shared mathematical space.

I. AI Service Architecture

The AI system in SmartPick is architected as an independent microservice, separate from the main Node.js/Express backend. This separation decouples the AI logic from the web application layer, allowing the AI service to evolve independently without requiring changes to the frontend or backend. The backend communicates with the AI service exclusively through a well-defined HTTP API contract, sending book metadata and user quiz answers as structured JSON and receiving recommendations and explanations in return.

The AI microservice is implemented in Python using the Flask framework, chosen for its compatibility

with the broader Python machine learning ecosystem, including the sentence-transformers library and the Pinecone client SDK. The service exposes four primary endpoints, each corresponding to a distinct user-facing feature of SmartPick.

A. System Components

The AI integration layer comprises four distinct components working in concert:

Component	Technology
AI Microservice	Python / Flask
Embedding Model	sentence-transformers (all-MiniLM-L6-v2)
Vector Database	Pinecone (Serverless)
Book Database	MongoDB (via backend)
Explanation Engine	LLM (GPT / open-source)

Table 7.1: AI System Components

B. API Endpoint Contract

Endpoint	Purpose
/api/generate/comparison	Generates dynamic quiz questions for scanned books
/api/analyze/comparison	Runs vector search; returns best match + explanation
/api/questions/personality	Returns general quiz questions for Surprise Me
/api/analyze/personality	Searches full Pinecone index; returns top 3 + profile

Table 7.2: AI Microservice Endpoints

C. Resilience and Fallback Strategy

All calls from the backend to the AI microservice are wrapped in try/catch blocks with a configurable timeout. If the AI service is unavailable or exceeds the response time threshold, the backend automatically falls back to a set of pre-defined static quiz questions and a template-based explanation. This ensures that SmartPick remains functional even during AI service downtime, preserving the user experience without a hard failure. Fallback responses maintain the conversational tone of the application and are indistinguishable in format from AI-generated responses.

II. Quiz Generation Logic

A key design principle of SmartPick is that the quiz presented to the user is not static. Rather than asking every user the same predetermined questions regardless of context, SmartPick generates questions dynamically based on the specific books being compared. This context-sensitivity is what makes the quiz feel intelligent and purposeful rather than generic.

D. Context-Aware Question Generation

When a user scans two or more books and initiates a comparison, the backend forwards the metadata of each scanned book to the /api/generate/comparison endpoint. The payload includes each book's title, authors, genre categories, description, page count, mood vibes, and reading difficulty level.

The AI service analyses the submitted books and identifies the dimensions along which they most significantly differ. For example, if one book is a dense philosophical novel and the other a fast-paced thriller, the relevant differentiating dimensions are reading depth, pacing preference, and emotional tone. The service generates 3 to 5 targeted questions that probe these specific dimensions, ensuring that each user answer carries maximum discriminative signal for the recommendation step. This contrasts with a static quiz, where questions are fixed regardless of the books involved a static quiz might ask about genre preference even when all scanned books belong to the same genre, yielding no useful signal.

E. Question JSON Structure

Field	Type	Description
text	String	Question shown to user
options	Array<String>	2-4 selectable answer choices
dimension	String	Semantic dimension being probed

Table 7.3: Quiz Question JSON Structure

The complete set of generated questions is stored in the QuizSession MongoDB document at session creation time, making the quiz state fully persistent.

Sessions expire after seven days via a MongoDB TTL index, balancing persistence with storage efficiency.

F. Surprise Me Quiz

The Surprise Me feature follows a different flow. Since no books have been pre-scanned, there is no specific context to differentiate. The /api/questions/personality endpoint returns general, personality-oriented questions that explore the user's current reading mood, available time, genre inclinations, and preference for intellectual depth versus narrative entertainment. These answers collectively construct a broad preference vector used to search the full Pinecone index of 52,000 books, returning three curated recommendations.

III. Recommendation Algorithm

The recommendation algorithm is the technical core of SmartPick's AI layer. It is built on semantic vector similarity: both books and user preferences are encoded into a shared high-dimensional vector space using the same embedding model, and recommendations are produced by identifying which book vectors are geometrically closest to the user's preference vector. This approach is formally known as approximate nearest neighbour (ANN) search and is the same technique used in large-scale retrieval systems across industry.

G. Phase 1: Offline Book Indexing

Before the application serves any users, all books in the dataset are indexed into Pinecone. For each book, a descriptive text string is constructed by concatenating its key semantic fields: title, authors, genre categories, description, mood vibes, reading difficulty, and the perfectFor list. For example:

> *"1984 by George Orwell. Genres: dystopian fiction, political science. A bleak, intense, thought-provoking story set in a totalitarian society. Best for readers who enjoy dense, philosophical, slow-burn narratives. Difficulty: advanced. Mood: dark, thoughtful."*

This string is passed through the all-MiniLM-L6-v2 embedding model, producing a 384-dimensional dense vector. This vector, paired with the book's MongoDB _id as metadata, is upserted into Pinecone. The process runs once at setup and incrementally for new books. With 52,000 books, the full index occupies approximately 75 MB well within Pinecone's free serverless tier.

H. Phase 2: Preference Vector Construction

When a user completes the quiz, their answers are concatenated into a natural language preference string mirroring the style of the book description strings. For example:

> *"I want to feel intellectually challenged and thoughtful after reading. I have a full week so page count is not a concern. I prefer something dense and philosophical that makes me think deeply."*

This string is passed through the same embedding model used during indexing, producing a 384-dimensional preference vector. The use of an identical model for both book indexing and query embedding is critical: it guarantees that books and preferences exist in the same vector space, making similarity scores geometrically meaningful.

I. Phase 3: Vector Similarity Search

For the Comparison Quiz, the user has scanned 2 to 5 specific books. The AI service fetches only the vectors of those specific books from Pinecone by their MongoDB IDs and computes cosine similarity directly between each book vector and the preference vector. This is a targeted comparison rather than a broad search.

For the Surprise Me feature, the preference vector is used as a query against the full Pinecone index of 52,000 books. Pinecone's ANN algorithm returns the top-k most similar vectors in milliseconds regardless of index size. The top 3 results are returned as recommendations.

Cosine similarity measures the angle between two vectors rather than their absolute distance, which is appropriate for text embeddings since direction encodes meaning while magnitude is less informative.

Similarity Score	Interpretation
0.85 – 1.00	Strong match clearly for you
0.70 – 0.84	Good match you'll enjoy this
0.55 – 0.69	Moderate match worth exploring
Below 0.55	Weak match fallback triggered

Table 7.4: Cosine Similarity Interpretation

J. Phase 4: Explanation Generation

Vector search identifies which book to recommend but does not produce a human-readable justification.

To generate the explanation shown to the user, the AI service constructs a short prompt passed to a language model:

> *Prompt: "The user is looking for: [preference string]. The recommended book is [title] by [author], described as: [description]. In 2–3 sentences, explain in a warm and conversational tone why this book suits this reader."*

This two-step design vector search for selection, language model for explanation is more reliable than asking a language model to both select and explain, as it grounds the recommendation in a mathematically verifiable similarity score.

K. Output Contract

Field	Type	Description
recommendation.bookId	String	MongoDB _id of top book
recommendation.confidence	Float (0–1)	Cosine similarity score
explanation	String	LLM-generated justification
alternatives[]	Array	Runner-up books with scores
personalityProfile	Object	Reader type label + description

Table 7.5: Recommendation Output Contract

IV. Personalization Strategy

SmartPick's personalization strategy is built on the recognition that the best book recommendation is not simply the most popular or highest-rated title it is the title most semantically aligned with a specific reader's current mood, intellectual appetite, and available time. The vector-based approach is inherently personalized: every user's preference vector is unique to their quiz answers, and the resulting recommendation reflects that specific combination rather than any aggregate profile.

L. Semantic Personalization Through Embeddings

The embedding model captures nuanced semantic relationships that a rule-based system cannot. A user who answers that they want to feel 'enlightened' and prefer something that 'breaks their brain' will receive a preference vector naturally close to books tagged as philosophical, dense, and intellectually demanding not because of explicit label matching, but because the embedding model has learned that these concepts occupy nearby regions of the vector space. Personalization is therefore sensitive to the full meaning of a user's expressed preferences, not just keyword overlap.

M. AI-Enriched Book Metadata

The quality of personalization is directly dependent on the richness of book representations stored in Pinecone. Raw dataset fields such as title and ISBN are insufficient for semantic matching. SmartPick implements an AI enrichment pipeline that augments each book record before embedding:

- readingDifficulty: Classified as beginner, intermediate, or advanced based on vocabulary complexity, thematic density, and narrative structure.
- moodVibes: One or more mood descriptors from a controlled vocabulary, thoughtful, intense, cozy, dark, uplifting, mysterious, embedded as part of the book's description string.
- perfectFor: Natural language reader archetypes (e.g., 'readers who enjoy slow-burn philosophical fiction') that improve alignment with personality-style quiz answers.
- summary: An AI-generated interpretive summary describing the reading experience rather than the plot, contributing significantly to embedding quality.

Enrichment fields are generated by passing raw book data through a language model prompt. The aiEnriched boolean flag and aiLastEnriched timestamp in the Book schema track which records have been processed, enabling prioritized background enrichment. Books scanned via ISBN are enriched on demand if not yet processed.

N. Reading Personality Classification

After any completed quiz, SmartPick generates a Reading Personality Profile by passing the user's preference string to a language model that classifies the user into a named reader archetype with a one-sentence description. Example profiles include:

- The Contemplative Explorer: drawn to ideas over plot; prefers books that leave them thinking for days.
- The Quick Adventurer: values pace and momentum; reads for immersion and excitement.
- The Mood Reader: chooses books based on current emotional state rather than genre loyalty.
- The Deep Diver: seeks multi-layered works; comfortable with difficulty and ambiguity.

The personality profile is stored in the QuizSession document and displayed in the Reading Personality section of the SmartPick interface. It makes the recommendation feel personally meaningful and demonstrates that the system can infer higher-order user characteristics from a small number of quiz responses.

O. Session Architecture

Every user interaction is anchored to a QuizSession document in MongoDB, identified by a UUID. The session stores the complete context: books scanned, questions presented, each answer with timestamp, computed results, and personality profile. Sessions expire automatically after seven days via MongoDB's TTL index on the expiresAt field, balancing persistence with storage efficiency. Given a session ID, it is possible to reconstruct exactly why a particular book was recommended, making the system fully auditable.

P. Future Personalization Roadmap

The current session-scoped implementation is designed to support deeper personalization in future iterations:

- Persistent User Profiles: linking quiz sessions to registered accounts to accumulate long-term preference history, enabling preference vectors derived from multiple past sessions.
- Collaborative Filtering: augmenting vector search with signals from users whose preference vectors cluster near the current user's, enabling 'readers like you also enjoyed...' recommendations.
- Explicit Feedback Loop: allowing users to rate recommendations post-reading, with ratings fed back as fine-tuning signals to improve embedding quality over time.
- On-Demand Enrichment: triggering immediate AI enrichment for any book scanned via ISBN not yet processed, ensuring obscure titles can be recommended with full semantic richness.

Together, these design choices establish SmartPick's AI layer not as a static rule engine but as a semantically grounded retrieval system that is both academically principled and practically extensible. The use of dense vector embeddings, approximate nearest neighbour search, and natural language explanation generation represents a meaningful application of contemporary NLP research to the domain of decision support and provides a solid foundation for continued development beyond the scope of this project.

VIII. CHALLENGES

I. Technical Challenges Faced

A. Handling Book Data & API Integration

One of the main technical challenges was fetching accurate book details using ISBN or manual input. Many APIs do not return complete or consistent data (e.g., missing author names, descriptions, or ratings). This created issues in displaying reliable results to users.

B. Managing Multiple Features in One System

The project includes multiple functionalities such as:

- Single Scan
- Compare Books
- Surprise Me (quiz-based recommendation)
- User Profile (reading personality)

Handling all these features together while maintaining smooth navigation and performance was challenging.

C. Dynamic Recommendation Logic

Creating a logic that suggests books based on user mood (Inspired, Thoughtful, etc.) requires designing conditional flows. Since this is not just static data, mapping user's input to meaningful recommendations was complex.

D. UI State Management

Switching between different pages (Scan, Compare, Quiz) and maintaining user selections (like selected books or answers) required proper state handling, which can be tricky in frontend development.

E. Input Validation Issues

Handling ISBN input was another challenge:

- Users may enter invalid ISBNs
- Different ISBN formats (10-digit vs 13-digit)
- Empty or incorrect inputs

Ensuring error-free user experience requires proper validation logic.

II. Design Challenges

A. Creating an Intuitive User Experience

The goal was to make book selection simple and enjoyable. Designing a flow where users can:

- Quickly scan a book
- Compare options
- Take a quiz

B. Dark Theme UI Design

The project uses a dark theme. Challenges included:

- Maintaining readability
- Choosing proper contrast colors
- Ensuring text visibility on different screens

C. Layout Consistency

Ensuring consistent design across all pages (cards, buttons, spacing, fonts) was difficult, especially with multiple sections like:

- Personality cards
- Quiz options
- Book comparison

D. Avoiding Information Overload

Since the app deals with books, there is a lot of information (titles, authors, ratings, etc.). The challenge was to present only important details without overwhelming the user.

E. Responsive Design

Making sure the UI works properly on different screen sizes (laptop, tablet, mobile) was another design challenge.

III. Solutions Implemented

A. API Handling & Fallback Logic

To handle incomplete data:

- Default values were used (e.g., "Unknown Author")
- Conditional rendering ensured UI does not break

B. Modular Structure

The application was divided into sections:

- Scan Module
- Compare Module
- Quiz Module
- Profile Module

This improved maintainability and reduced complexity.

C. Simplified Recommendation Logic

Instead of complex AI, a rule-based approach was used:

- User selects mood → mapped to predefined categories
- Categories linked to book suggestions

This made the system efficient and easy to manage.

D. Clean UI Components

Reusable UI components (cards, buttons, sections) were created to maintain consistency across pages.

E. Input Validation

- ISBN length checks
- Empty input handling
- Error messages for invalid entries

This improved user experience.

IX. FUTURE SCOPE

I. Mobile App Development

The current system is web-based. In the future, it can be converted into a mobile application using:

- React Native or Flutter

Benefits:

- Better accessibility
- On-the-go usage
- Improved user engagement

II. Social Features

Adding social interaction can enhance the platform:

- Users can share book recommendations
- Follow other readers
- Create reading lists
- Review and rate books

This will make SmartPick more like a community-driven platform.

III. Advanced AI Integration

Currently, recommendations are rule-based. Future improvements can include:

- Machine Learning models
- Personalized recommendations based on history
- NLP-based sentiment analysis of book reviews

This will make recommendations more accurate and intelligent.

IV. Camera-Based ISBN Scanning

Instead of manual entry, users can:

- Scan book barcode using camera

Technologies:

- OCR (Optical Character Recognition)
- Barcode scanning libraries

This will make the process faster and more user-friendly.

X. CONCLUSION

I. Work Summary

SmartPick is a user-friendly web application designed to simplify the process of selecting books. It provides multiple features such as:

- Book scanning
- Book comparison
- Mood-based recommendations
- Reading personality insights

The system focuses on reducing decision-making efforts and improving reading experience.

II. Key Contributions

- Developed an interactive UI with multiple features
- Implemented a recommendation system based on user preferences
- Designed a structured and visually appealing layout
- Integrated input-based and quiz-based selection methods

III. Learning Outcomes

Through this project, the following skills were developed:

Technical Learning

- Frontend development (HTML, CSS, UI design)
- Handling user input and validation
- Structuring multi-page applications

Problem-Solving Skills

- Managing incomplete data
- Designing user-friendly flows
- Breaking complex features into simpler modules

IV. Design Thinking

- Importance of UI/UX
- Balancing aesthetics and usability
- Creating intuitive user journeys

REFERENCES

[1] J. Nielsen, *Usability Engineering*, 1994.

[2] R. T. Fielding, "REST architectural style," 2000.

[3] M. Fowler, *Patterns of Enterprise Application Architecture*, 2002.

[4] React, "React Documentation," [Online]. Available: https://react.dev

[5] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, 2003.

[6] B. Schwartz, *The Paradox of Choice: Why More Is Less*. New York, NY, USA: Harper Perennial, 2004.

[7] Node.js, "Node.js Documentation," [Online]. Available: https://nodejs.org

[8] MongoDB, "MongoDB Documentation," [Online]. Available: https://www.mongodb.com/docs

[9] E. Turban et al., *Decision Support Systems*, Pearson, 2011.

[10] D. Norman, *The Design of Everyday Things*, 2013.

[11] ISO, "Human-centred design for interactive systems (ISO 9241-210)," 2019.

[12] A. Da'u and N. Salim, "Deep learning recommender systems: A survey," *Artificial Intelligence Review*, 2020

[13] L. Liu, "Machine learning-based recommendation systems in e-commerce," 2022.

[14] Y. Li, K. Liu, and S. Wang, "Recent developments in recommender systems: A survey," 2023.

[15] S. Raza et al., "A comprehensive review of recommender systems: Transitioning from theory to practice," 2024.

[16] Goodreads, "About Goodreads," [Online]. Available: https://www.goodreads.com/about

[17] Y. Deldjoo et al., "A review of modern recommender systems using generative models," 2024.

[18] Q. Zhang and Y. Xiong, "AI-powered e-commerce recommendation systems and customer engagement," *Current Psychology*, 2024.

[19] G. Jangra et al., "AI in e-commerce recommendation systems for personalization," 2024

[20] Messaoudi and M. Loukili, "Deep neural collaborative filtering for e-commerce recommendation," 2024.

[21] R. Shrivastava et al., "Deep ensembled recommendation systems for personalization," 2024.

[22] AI trends in recommendation systems and NLP integration, 2025.
[23] ACM, “Recent advances in recommender systems in e-commerce,” 2025.
[24] A. Poniszewska-Maranda et al., “Recommendation systems using machine learning,” 2025.
[25] S. Sameena et al., “Personalized recommendation systems for e-commerce,” 2025.
[26] M. Etlioglu, “Impact of AI-powered recommendation systems on user behavior,” 2025.