Insight Beam – Book Review Web App

Student: Yumna Waris

University: University of Karachi
Department: UBIT - Department of Computer Science
Course: Introduction to Internet Programming (3rd Year, 6th Semester)
Supervisor: Mr. Zaeem Tariq
Date of Submission: December, 2025

Abstract

Insight Beam is a learning-oriented, full-stack web application that explores how modern web systems are designed, built, and connected end to end.
The app enables users to sign up, log in, browse books, like or unlike titles, save them to a personal collection, and submit contact messages.

The frontend is built with **Next.js and React**, styled with **Material UI**, and manages session state using a lightweight context-based approach.
The backend uses **Node.js with Express**, **JWT** for authentication, and **PostgreSQL** accessed through **Knex**; image uploads are handled via **Multer**.
Docker Compose provisions PostgreSQL and pgAdmin for a consistent local database environment.

Through this project, I practiced **REST API design**, **database schema modeling**, **authentication flows**, **protected route handling**, **error handling**, and **UI composition**.
While not production-ready, *Insight Beam* provided hands-on experience that clarified how client–server interactions, state management, and data persistence come together—laying a strong foundation for tackling more complex, scalable systems.

Introduction

Context

I chose this project to combine my interest in books with a desire to learn how a real-world web app connects a modern frontend to a secure, data-backed backend.

Purpose

The goal was **learning**—not competing with commercial platforms—by building a working app that exercises authentication, CRUD, and client–server communication.

Objectives

- Understand client–server communication via RESTful APIs
- Learn database integration using PostgreSQL and Knex
- Practice containerization using Docker Compose
- Implement authentication (JWT) and core CRUD operations

System Analysis

Problem Statement

Full-featured platforms like Goodreads are complex.
This project distills key features to understand the technical foundations of authentication, browsing, engagement (likes/saves), and database-backed APIs.

Functional Requirements

- User Accounts – Register, log in, update profile, change password
- Books – Create, update (owner-only), list all, fetch one
- Engagement – Like/unlike books; save/unsave to personal collection
- Contact – Submit contact messages

(Reviews and personalized recommendations are planned enhancements.)

Non-Functional Requirements

- **Usability:** Clean, responsive UI with Material UI
- **Performance:** Lightweight API responses and minimal round-trips
- **Deployability:** Dockerized database environment for consistency

---

System Design

Architecture

```
Frontend (Next.js + React)
        ⇅
REST API (Express)
        ⇅
PostgreSQL (Knex)
```

- **Auth:** JWT stored client-side; Bearer token for protected endpoints
- **Media:** Image uploads via Multer; binary data stored in database

Database Design

Table	Key Columns
users	id, name, email, password, created_at
profile	id, user_id, data, created_at, updated_at
books	id, title, author, description, publisher, isbn, creator, purchase_link, image_link, created_at
user_like	user_id, book_id, created_at
user_collection	user_id, book_id, created_at
contacts	id, name, email, message, created_at
reviews	id, user_id, book_id, review_message, created_at *(endpoints pending)*

Technologies Used

Layer	Tools / Frameworks
Frontend	Next.js (App Router), React, Material UI
Backend	Node.js, Express, Multer, JWT
Database	PostgreSQL with Knex Query Builder
Containerization	Docker Compose (for Postgres & pgAdmin)
Version Control	Git + GitHub

Implementation

Frontend

- Built Next.js pages for Login, Register, Explore, My Likes, My Collections, and Contact.
- Managed state via `AuthContext`, persisted to `localStorage`.
- API client injects JWT into headers and handles JSON errors.

- Material UI components ensure responsive and accessible design.

Backend

- Express routers for users, books, likes, collections, contacts, and images.
- Middleware (`auth`, `optional_auth`) verifies and attaches JWT claims.
- Aggregated book metadata (`like_count`, `save_count`, user flags).
- Image uploads handled via Multer; served from Express routes.

Deployment / Environment

- **Docker Compose** provisions Postgres and pgAdmin.
- Environment variables configure DB connection, server port, and JWT secret.

Challenges

- Coordinating authentication across client routes and protected API calls
- Maintaining consistent error responses and controller logic
- Handling CORS and base URL differences between dev and prod
- Managing schema evolution and migrations

Testing

Approach

Manual testing via Postman (for APIs) and Browser DevTools (for UI flows).

Sample Cases

Test	Expected Result
Register & Log In	Token issued and stored client-side
List Books	Books display with live like/save metadata
Like/Unlike	Counter and UI state update instantly
Save/Unsave	Personal collection reflects changes
Contact Form	Validates inputs and submits successfully

Results

Working Features

- Account creation and JWT-based login
- Browse books with live engagement metadata
- Like/unlike and save/unsave flows
- Contact form submissions
- Image upload and profile image storage

DevOps

- Database environment reproducible via Docker Compose and pgAdmin

---

Learning Outcomes / Reflections

- Understood how frontend events map to Express handlers and DB operations
- Gained insight into REST API structure and authorization mechanisms
- Learned many-to-many data modeling (via join tables)
- Practiced state management with React Context
- Balanced simplicity and security (password hashing, validation)
- Built confidence to tackle larger Next.js and TypeScript projects

---

Future Enhancements

- Personalized recommendations (genre & ratings-based ML)
- Admin dashboard for user and book management
- Full review CRUD and aggregation
- CI/CD pipeline with GitHub Actions
- Automated testing (Jest, React Testing Library)
- Security improvements (bcrypt, rate limiting)

Conclusion

Insight Beam served as a **practical, end-to-end introduction** to full-stack development. It connected a modern React/Next.js frontend to an Express/PostgreSQL backend with JWT authentication and REST APIs.
The project clarified key concepts—API structure, state management, data modeling, and environment setup—while highlighting next steps to productionize features. This foundation positions me to build more complex, reliable, and scalable systems.

References

- React Documentation
- Next.js Documentation
- Material UI Documentation
- Node.js Documentation
- Express Guide
- Knex.js Documentation
- PostgreSQL Documentation
- JWT Introduction
- Multer (File Uploads)
- Docker Documentation
- pgAdmin Documentation
