Development Report: Openverse Media Application (MERN Stack)

1.0 Introduction: Critical Analysis of Software Engineering Principles

Openverse Media Application is a web application that allows one to search for open license media based on the openverse API. The app relies on the MERN (MongoDB, Express.js, React.js, Node.js) stack to build the project, leveraging current web development frameworks and software engineering principles to enhance scalability, maintainability, and security. MERN stack is open source and enable the development of high performant web base application which makes it a suitable option for this project.

1.1 Contextualizing the Application in Modern Software Development

Web application development has come a long way with the advent of cloud computing, microservices, and API-based architecture. Contemporary web development calls for high-performance, scalable, and user-friendly applications that can easily integrate with third-party services. Openverse Media Application follows the trend by embracing the microservices-influenced architecture in the guise of RESTful APIs, JWT authentication, and Docker-based containerization.

1.2 Advancements in Software Engineering and Embraced Development Strategy

Current software development focuses on modularity, component-oriented design, and state management system implementation to enable the delivery of a better user experience. Sommerville (2020) argues that by employing a disciplined software development process, reusability and maintainability of code are enhanced. The MERN stack is supported by its flexibility and effectiveness in offering full-stack JavaScript development. The one-language JavaScript paradigm facilitates natural data sharing between frontend and backend without context-switching overhead (Tilkov and Vinoski, 2010).

One of the most important methodology choices was Redux Toolkit for handling state. Redux has been broadly used for managing global state, especially where there are complex data flows

needed (*Redux Essentials, Part 1: Redux Overview and Concepts | Redux*, no date). With Redux Toolkit usage, state mutations become simpler for better performance and avoiding unnecessary re-renders.

1.3 Rationale for Technological and Methodological Choices

MERN Stack for Full-Stack Development

- MongoDB: A NoSQL database that provides flexibility in handling media data, making retrieval and storage of data easy (Stonebraker, 2010).
- Express.js: A lightweight framework that simplifies API development, Increase performance and maintainability (Shen, Van Krimpen and Spruit, 2019).
- React.js: Meta Platforms, Inc. (n.d.) *React: A JavaScript library for building user interfaces*. Available at: https://reactjs.org.
- Node.js: Ensures efficient event-driven, non-blocking I/O operations, making it suitable for real-time applications (Tilkov and Vinoski, 2010).

1.4 RESTful API Design & Microservices-Inspired Architecture

The project follows a RESTful API design with a well-defined separation of concerns between frontend and backend. This is in line with contemporary software engineering standards that dictate modularity and scalability (Fielding, 2000). While not a microservices-based architecture in the true sense, the modular architecture guarantees future scalability with eventual evolution into a microservices architecture if needed.

1.5 JWT-Based Authentication for Security

Security is a critical requirement of contemporary web applications. JSON Web Token (JWT) authentication facilitates a stateless, scalable security paradigm that reduces database queries while allowing secure API access. This choice is in line with security best practices according to (Jones, Bradley and Sakimura, 2015).

1.6 Docker for Containerization

Docker is used to provide a uniform development and deployment environment. Containerization enhances dependency management, deployment automation, and scalability (MerkelDirk, 2014). With more adoption of DevOps practices, Docker makes the Openverse Media Application responsive to cloud deployments.

By adopting modern software engineering principles, the Openverse Media Application demonstrates best practices in full-stack development, security, and performance optimization. The adoption of MERN stack, Redux Toolkit, RESTful APIs, JWT authentication, and Docker is in line with current research and industry practice.

2.0 Project Planning

Systematic project planning is important for efficient execution, risk mitigation, and timely delivery of the openverse Media Application. This section outlines the architectural design, user interaction diagrams, project timeline, and risk assessment.

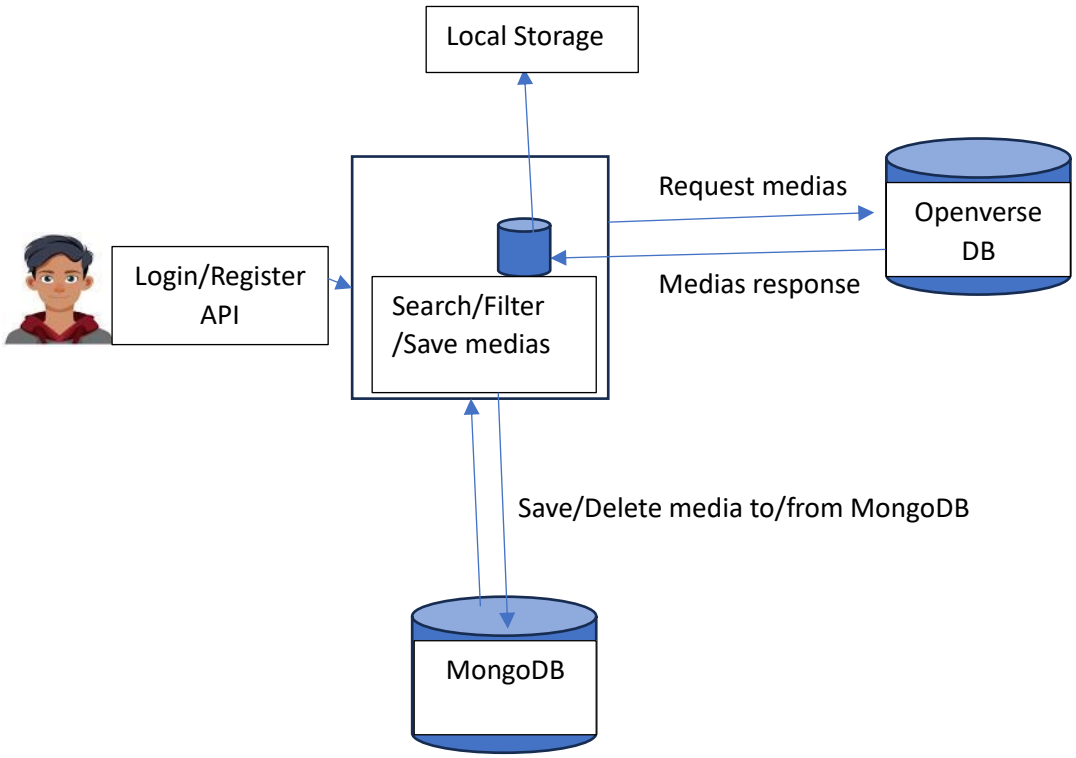
2.1 Architectural Diagram

Below is the high-level system architecture:

- Frontend (React.js, Redux Toolkit): Handles UI presentation and state management.
- Backend (Node.js, Express.js): Processes API requests and business logic.
- Database (MongoDB): Stores media search history and user authentication details.
- External API (Openverse API): Provides open-license media search functionality.
- Authentication (JWT): Ensures secure access control.
- Docker & Deployment: Manages the containerized backend and frontend services.

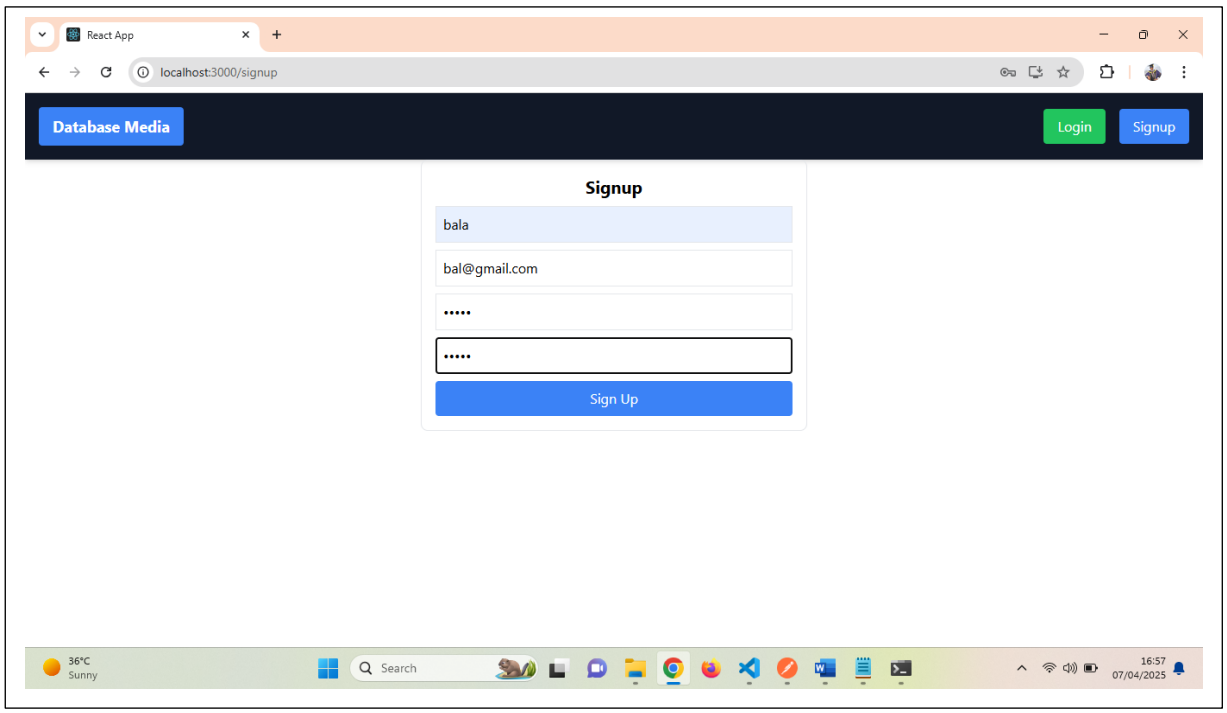
2.2. Sequence/Interaction Diagram

A sequence diagram illustrating user interactions:



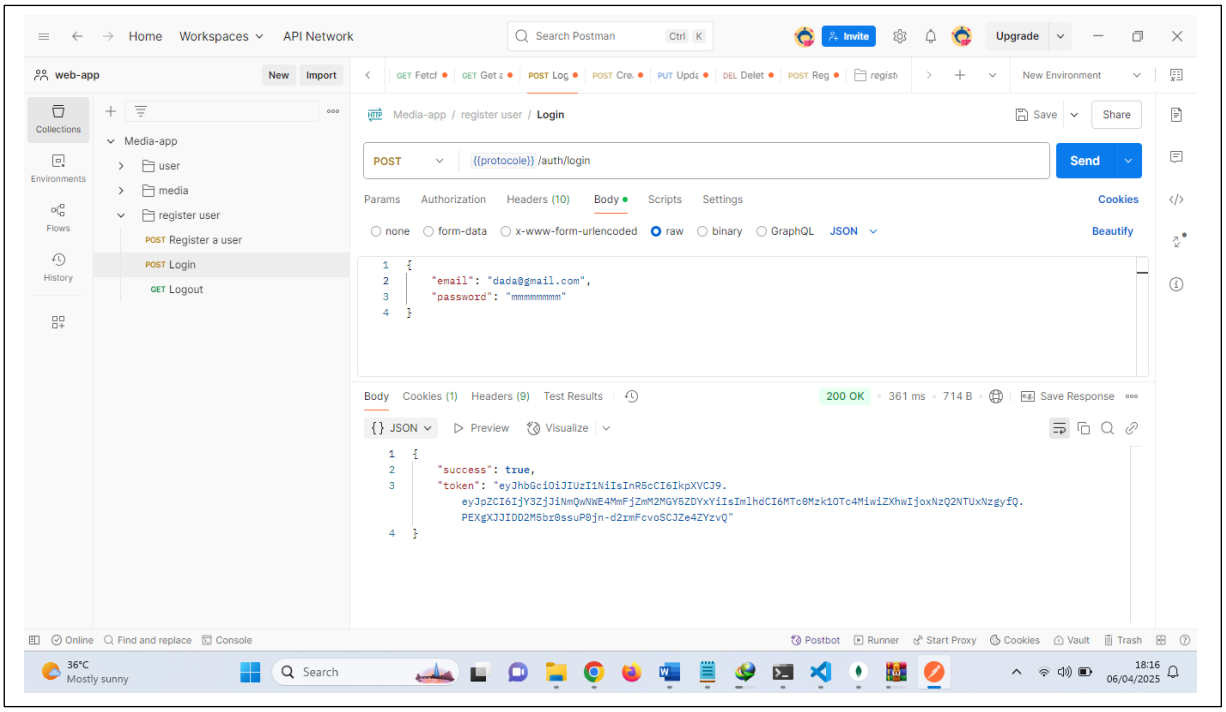
User Authentication:

- Register a new user→ React UI sends credentials → Express.js validates JWT → MongoDB verifies user.

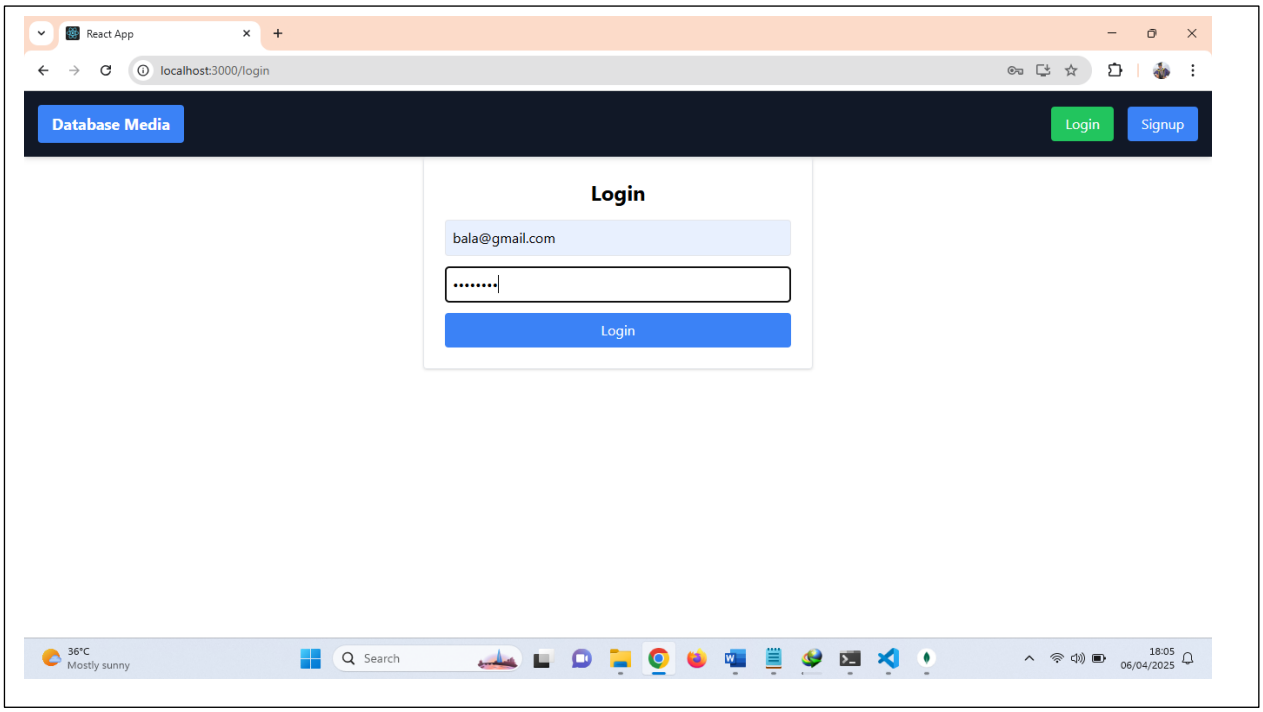


- User logs in → React UI sends credentials → Express.js validates JWT → MongoDB verifies user.

Testing the Login Screen with POSTMAN



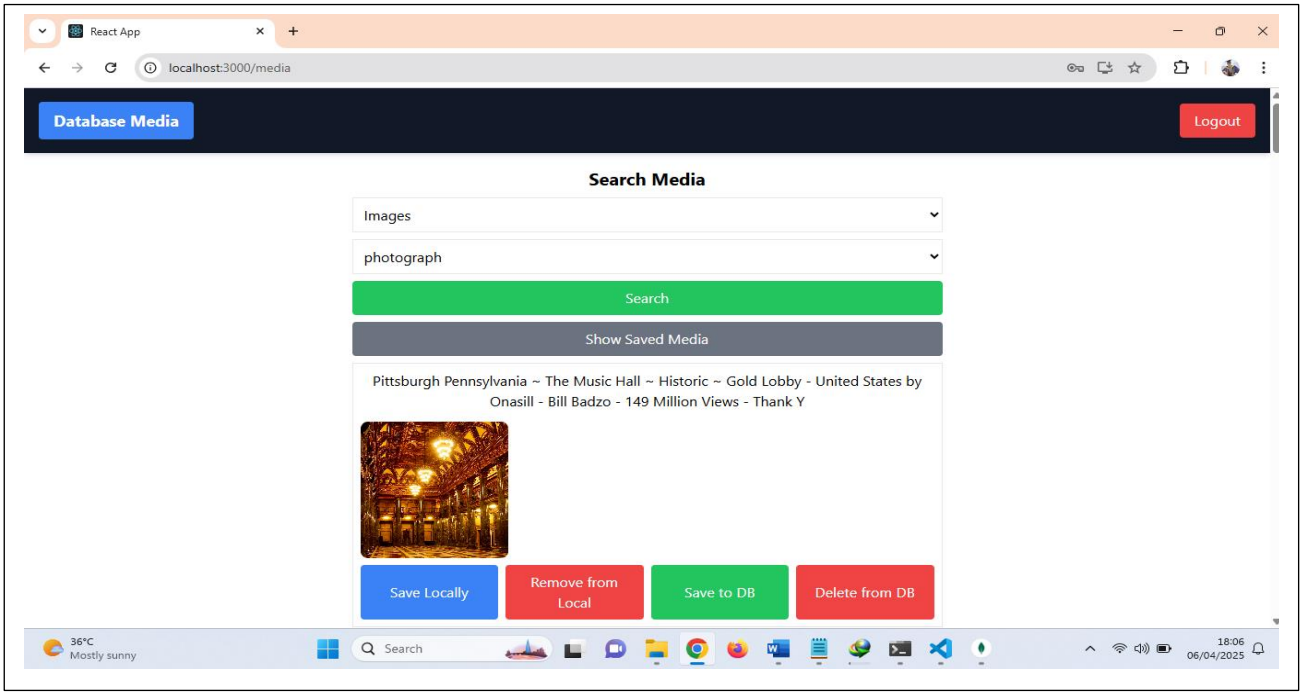
React UI Login Screen



Media Search:

- User enters a query → React UI calls backend API → Express.js fetches results from Openverse API → result is stored in local storage.

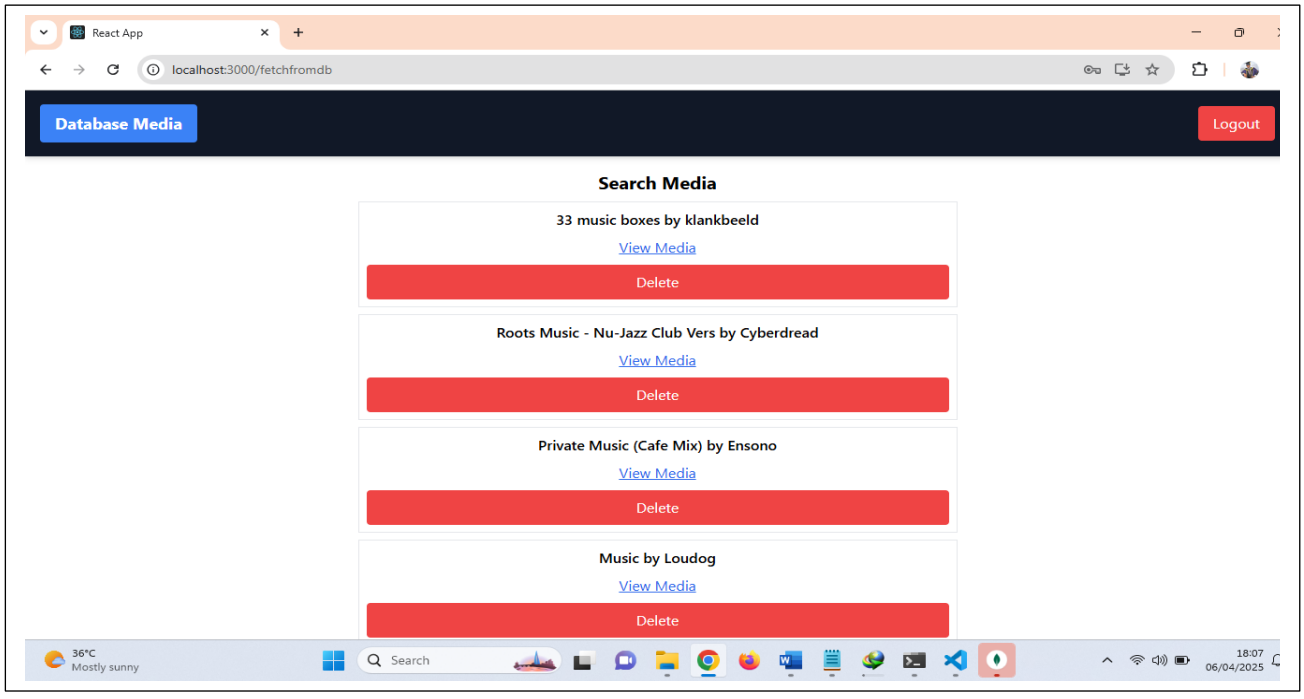
Fetching array of images from the Openverse API



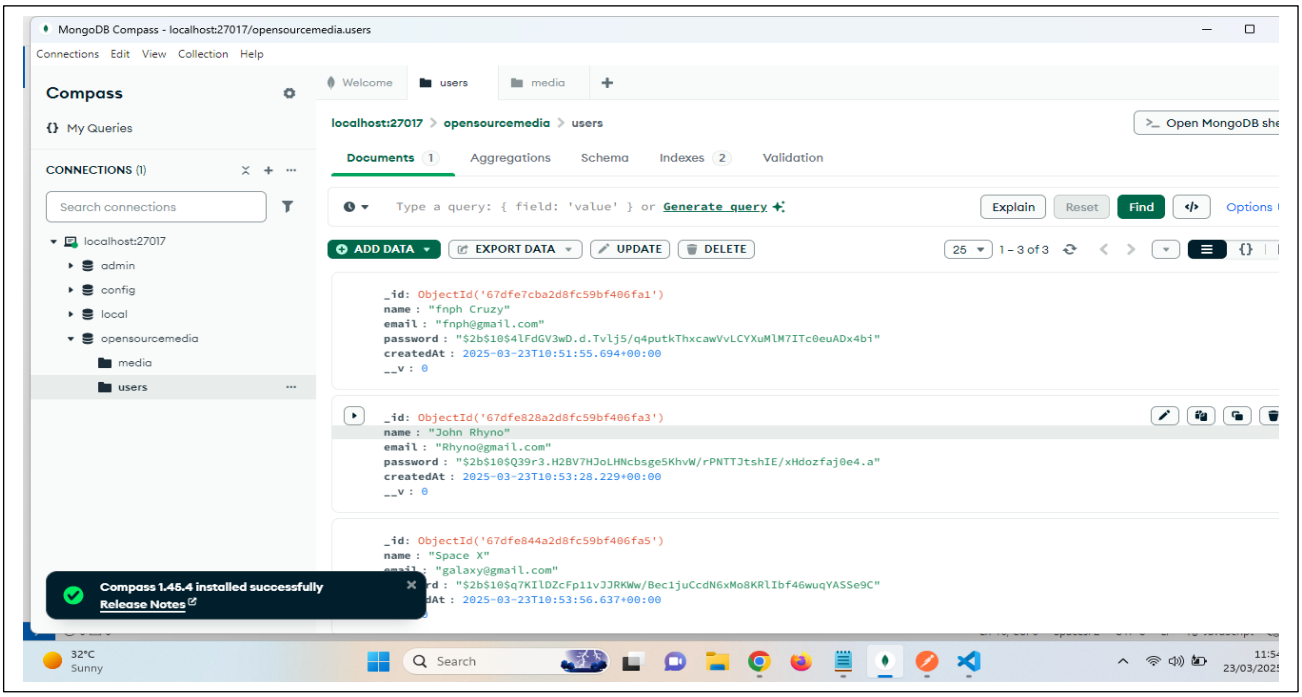
Media Management:

- User saves/deletes media → React dispatches Redux action → Express.js updates MongoDB.

Saved information pulled from MongoDB database



Viewing backend data from MongoDB Compass



Project Timeline and Milestones

Phase	Task Description	Estimated Duration
Phase 1	Requirement Analysis & System Design	1 Weeks
Phase 2	Backend Development (Node.js, Express, MongoDB)	1 Weeks
Phase 3	Frontend Development (React, Redux, Tailwind)	1 Weeks
Phase 4	Integration with Openverse API	3 Days
Phase 5	Testing & Debugging (POST MAN)	2 Days

Total Duration: **3 Weeks and 5 Days**

Risk Assessment and Mitigation Plan

Risk Factor	Potential Impact	Mitigation Strategy
API Downtime	Search feature failure	Implement caching and fallback mechanisms
Security Vulnerabilities	Data breaches	Use JWT authentication, HTTPS, and validation
Performance Bottlenecks	Slow user experience	Optimize database queries, use indexing
Unclear Requirements	Scope creep	Regular stakeholder meetings, agile methodology
Deployment & Compatibility Issues	Failure in production	Use Docker for consistent environments

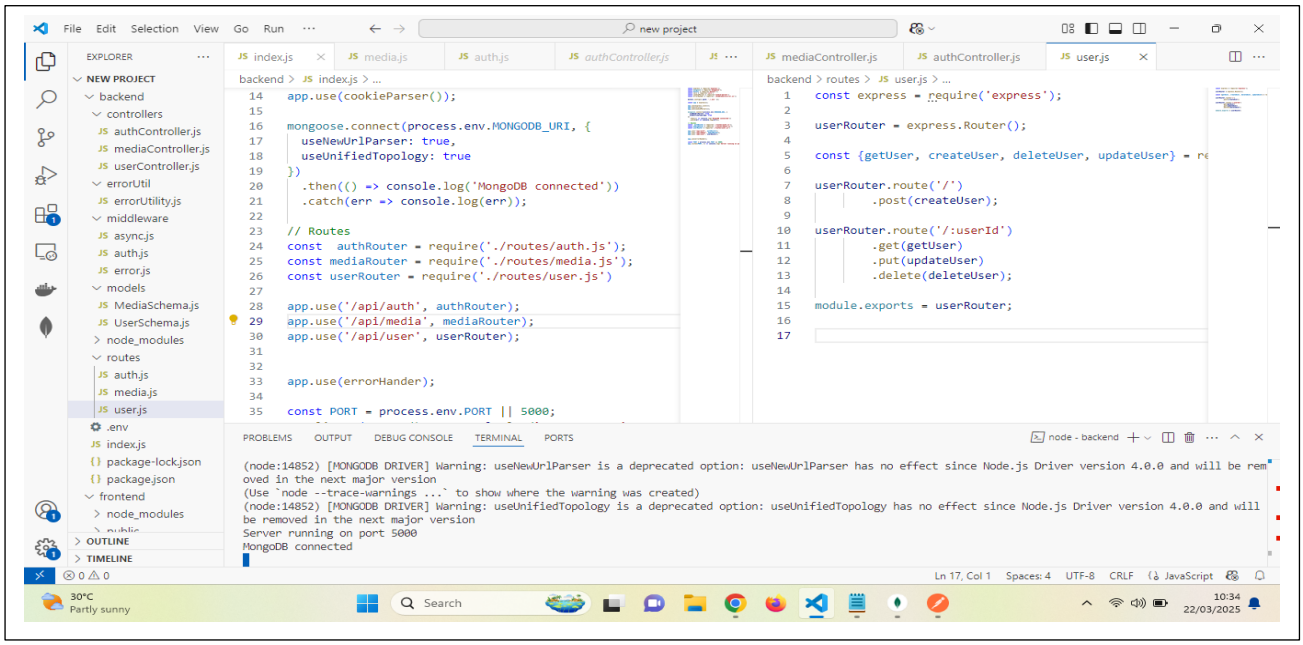
3.0 Project Development and Management Process

Effective project management and iterative development processes were central to successful completion of the openverse Media Application. In this section, major areas of the development process, feature driven development, and best practices in software engineering are outlined.

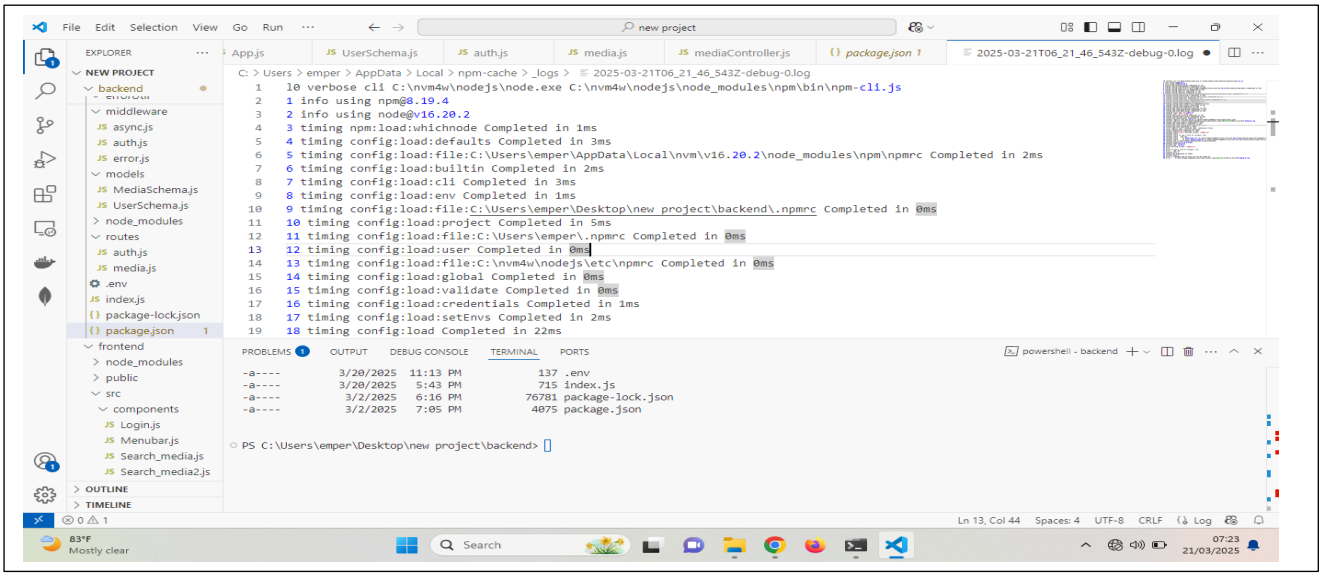
GitHub Repository and Development Logs

- GitHub Repository after containerizing media-app: https://github.com/IbrahimWale/dockerized-mediaapp
- GitHub Repository before containerizing media-app: https://github.com/IbrahimWale/media-app
- Commit History and Progress Tracking: The project utilized a formal commit methodology with very lengthy commit messages.
- Agile Methodology: Incremental feature increment development cycle with sprint oriented.
- Documentation & Trade-offs: Written out decisions, i.e., schema evolution of databases and authentication schemes.

Developing ExpressJs API and Routers functionalities



Tracing syntax error from log file

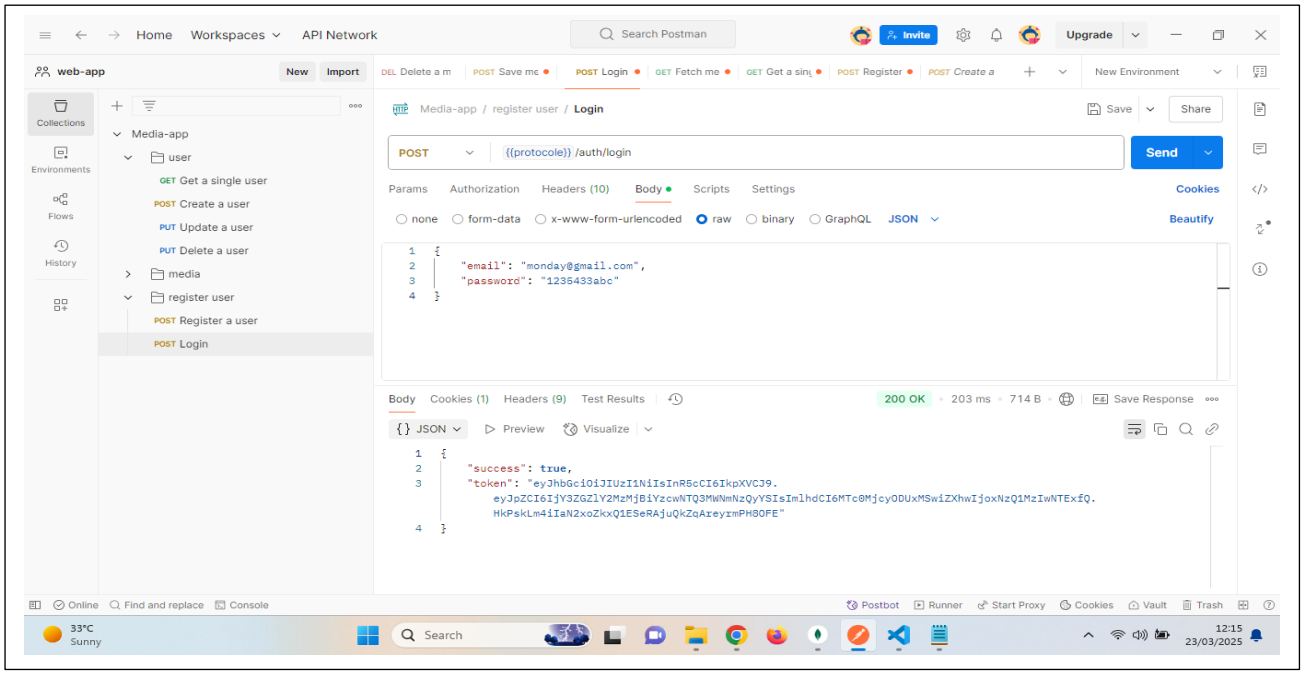


4.1 Test Cases and Coverage

- Unit Tests: Covered API routes, authentication flows, and database interactions using POSTMAN.

Evidence of testing and documentation using POSTMAN:

https://documenter.getpostman.com/view/42850199/2sB2cX91qQ



Containerization Approach

- Docker Compose was used to orchestrate multi-container setups, ensuring seamless integration between services.
- Scalability: Containers allow horizontal scaling and easy deployment to cloud platforms.

Reference

Jones, M., Bradley, J. and Sakimura, N. (2015) 'JSON Web Token (JWT)'. Available at: https://doi.org/10.17487/RFC7519,.

MerkelDirk (2014) 'Docker', *Linux Journal* [Preprint]. Available at: https://doi.org/10.5555/2600239.2600241.

Redux Essentials, Part 1: Redux Overview and Concepts | Redux (no date). Available at: https://redux.js.org/tutorials/essentials/part-1-overview-concepts (Accessed: 26 April 2025).

Shen, Z., Van Krimpen, H. and Spruit, M. (2019) 'A Lightweight API-Based Approach for Building Flexible Clinical NLP Systems', *Journal of Healthcare Engineering*, 2019. Available at: https://doi.org/10.1155/2019/3435609.

Stonebraker, M. (2010) 'SQL databases v. NoSQL databases', *Communications of the ACM*, 53(4), pp. 10–11. Available at: https://doi.org/10.1145/1721654.1721659.

Tilkov, S. and Vinoski, S. (2010) 'Node.js: Using JavaScript to build high-performance network programs', *IEEE Internet Computing*, 14(6), pp. 80–83. Available at: https://doi.org/10.1109/MIC.2010.145.

Fielding, R. (2000) Architectural Styles and the Design of Network-based Software Architectures. Doctoral Dissertation. University of California, Irvine.