iJETRM
International Journal of Engineering Technology Research & Management
Published By:
https://www.ijetrm.com/

DYNAMIC API URL GENERATION SYSTEM

Hemalatha V[1]
Assistant Professor, Department of Computer Science and Engineering, N.S.N College of Engineering and Technology, Tamilnadu, India.

Gandhiraja S[1], Keerthi V[2], Naveenhasan S[3]
PG Student, Department of Computer Science and Engineering, N.S.N College of Engineering and Technology, Tamilnadu, India.

ABSTRACT
Low-code platforms and automated development tools have become more popular as a result of the increasing demand for quick and effective backend development. This paper presents a Dynamic API URL Generation System that enables users to create fully functional CRUD APIs for MongoDB databases without writing any backend code, regardless of their level of technical expertise. The system dynamically generates Express.js-based RESTful APIs, including route definitions, data models, and real-time documentation, by letting users enter a custom schema through an easy-to-use interface. Constructed with a modular architecture utilizing Node.js, Express.js, React.js, and MongoDB, the platform facilitates secure access through JWT authentication, in-browser API testing, and scalable, multi-user API generation. To guarantee reliable and traceable operations, it also incorporates automated logging and error handling. The entry barrier is greatly reduced by the suggested system.

Keywords:
Dynamic API Generation, Low-Code Development, CRUD Operations, Schema Automation, JWT Authentication, API Logging.

INTRODUCTION

By allowing users to define schemas via a web interface and create RESTful API endpoints without knowing any code, the Dynamic API URL Generation System streamlines backend development. The MERN stack (MongoDB, Express, React, and Node.js) upon which it is built guarantees secure authentication through JWT, error validation with logging, and real-time operations for smooth database interactions. As low-code/no-code solutions gain popularity as a means of accelerating application scalability, the system improves accessibility for non-programmers while streamlining workflows for seasoned developers by streamlining CRUD operations and incorporating metadata handling.

OBJECTIVES

The Dynamic API URL Generation System makes it easier for developers and non-developers to manage database operations by enabling users to create fully functional CRUD APIs without writing code. It allows for schema-based customization with flexible field names and data types by automating the creation of RESTful APIs based on user-defined MongoDB schemas. JWT authentication provides secure access, and a user-friendly interface driven by React.js makes it easier to define schemas, preview APIs, and conduct in-browser testing. The system supports real-time API testing through an embedded test environment by dynamically generating API endpoint URLs based on schema inputs and MongoDB URIs. With its scalable Node.js and Express.js architecture, it integrates extensive logging and monitoring for auditing, analytics, and debugging, and guarantees modularity for multi-user environments.

METHODOLOGY

The Dynamic API URL Generation System is designed with a modular approach, making it easy to integrate, secure, and scale. It kicks off with a component-based design, using React.js for the frontend, Node.js with Express for the backend, MongoDB for data storage, and JWT for secure access. By adopting a RESTful API

iJETRM
International Journal of Engineering Technology Research & Management
Published By:
https://www.ijetrm.com/

model, it ensures smooth interactions with modern web tools. The authentication system is robust, utilizing bcrypt for password hashing and JWT tokens to guarantee that only authorized users can create and access APIs. Users can define collection names, fields, and data types through a schema definition interface, while the backend checks inputs before dynamically generating Mongoose models and CRUD endpoints. These routes POST, GET, PUT, DELETEare created on the fly with Express.js, so there's no need to restart the server. Each user is linked to their own MongoDB instance or Atlas URI, which provides data isolation and customization. Plus, a logging and analytics module tracks API request details like HTTP method, endpoint URL, status code, timestamp, and user ID, making it easier to audit, debug, and optimize.

A real-time preview and testing environment showcases all generated API endpoints on a user dashboard, allowing for live JSON response validation and sample data entry to verify functionality. The system is deployed on scalable cloud platforms like AWS or Heroku, ensuring it remains highly available and performs well. With continuous integration practices and updates driven by user feedback, the system stays efficient, making API generation simpler, more secure, and ready for future enhancements.

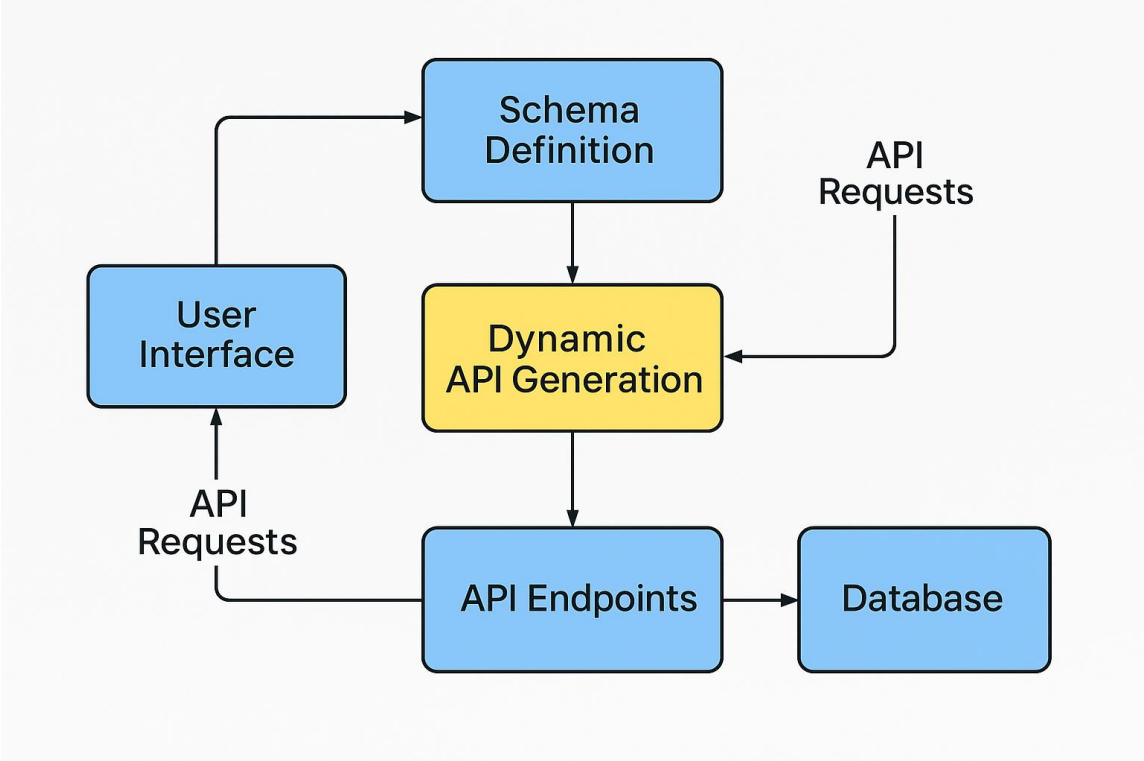

Figure 1 Dynamic API generation working flowchart

RESULTS AND DISCUSSION

The Dynamic API URL Generation System went through extensive testing to ensure it could effectively automate backend development, with a strong emphasis on usability, performance, scalability, and reliability. It was set up in a controlled test environment where several users could create and manage their own custom schemas. Each schema was designed to dynamically generate RESTful API endpoints, which could be accessed through both the built-in interface and external tools like Postman. The CRUD routes POST, GET, PUT, DELETE were accurately mapped to the user-defined schema structures, guaranteeing standardized JSON responses and proper error handling with HTTP status codes like 400 for bad requests and 404 for missing resources.

Usability testing revealed that even non-technical users found it easy to define schemas and interact with APIs using the user-friendly React.js frontend. Features like real-time validation and API previews significantly improved the user experience, while the system efficiently managed simultaneous requests from multiple users, showcasing its scalability and ability to handle concurrent processing. Performance testing on the backend

iJETRM
International Journal of Engineering Technology Research & Management
Published By:
https://www.ijetrm.com/

showed that Node.js and Express.js could support dynamic route generation without noticeable delays, and Mongoose facilitated smooth interactions with the database. Schema constraints were enforced, and real-time database updates were accurately reflected across different sessions.

For security, the system implemented JWT-based authentication to prevent unauthorized access to API endpoints, ensuring that user data remained isolated through server-side middleware protections. The logging and analytics modules effectively captured essential metadata, including timestamps, request types, and status codes, making audits and debugging a breeze. However, the current version is missing features like rate limiting, role-based access control, and an analytics dashboard, which are on the roadmap for future updates.

ACKNOWLEDGEMENT

The authors want to take a moment to sincerely thank their project guide and faculty members for their invaluable guidance, insightful feedback, and unwavering support throughout this journey. Their expertise and mentorship have been crucial in shaping the direction of this research and fine-tuning its implementation.

We also want to express our gratitude to our friends and teammates, whose collaboration, discussions, and encouragement have played a significant role in moving this project forward. Their readiness to brainstorm ideas and offer constructive feedback has been key in overcoming challenges and improving the quality of our research.

CONCLUSION

The Dynamic API URL Generation System makes backend automation a breeze by letting users create CRUD endpoints without any coding. It utilizes MongoDB URI and schema definitions, and it's built on Node.js, Express.js, and Mongoose. This setup allows for real-time API deployment while keeping things secure with JWT authentication and password encryption.

That said, it does have its limitations: it only works with MongoDB, doesn't support SQL, and has some restrictions on custom validations, schema management, and dashboards. You might notice a dip in performance when handling large-scale dynamic requests, and features like rate limiting, API versioning, and role-based access control are still in the works.

REFERENCES

1. M. Muehlbauer and P. A. Kirschner, "Cloud Computing and Service-Oriented Architectures: Concepts and Applications," IEEE Software, vol. 37, no. 3, pp. 68-75, 2020.
2. J. Chapin, "Serverless computing: Economic and technical challenges," IEEE Cloud Computing, vol. 7, no. 2, pp. 34-43, 2020.
3. S. Jung and Y. Park, "API Security: Challenges and Solutions," IEEE Transactions on Network and Service Management, vol. 17, no. 4, pp. 2201-2210, 2020.
4. M. Jefferson, "Serverless Architectures on AWS: With Examples Using AWS Lambda," IEEE Cloud Computing, vol. 6, no. 4, pp. 40-49, 2019.
5. A. Smith and E. Grant, "Understanding Cloud Computing: A Comprehensive Guide," IEEE Transactions on Cloud Computing, vol. 8, no. 5, pp. 1503-1515, 2018.