International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 05 | May 2026 www.irjet.net p-ISSN: 2395-0072

A Metadata-Driven No-Code Backend Platform with Automated Cloud Deployment

Mrs .Muskan, Mr. Shaik Roshan zameer, Mrs. S. Vijaya lakshmi, Mrs, B.Chandrasekhar

[1]*Mrs. Muskan*
B.Tech Student, Dept. of CSE, Mahatma Gandhi Institute of Technology(A), Hyderabad, Telangana, India
[2]*Mr.Shaik Roshan zameer*
B.Tech Student, Dept. of CSE, Mahatma Gandhi Institute of Technology(A), Hyderabad, Telangana, India
[3]*Mrs. S. Vijaya lakshmi*
Assistant Professor, Dept. of CSE, Mahatma Gandhi Institute of Technology(A), Hyderabad, Telangana, India
[4]*Mrs. B. Chandrasekhar*
Assistant Professor, Dept. of CSE, Mahatma Gandhi Institute of Technology(A), Hyderabad, Telangana, India

---

Abstract -*The rising demand for rapid software delivery has accelerated adoption of No-Code and Low-Code (NC/LC) platforms, yet these tools still struggle to bridge the gap between schema design and production-ready cloud deployment. This paper presents a No-Code Backend platform that allows users to design database schemas visually and instantly obtain fully functional REST APIs without writing any server-side code. The system is constructed as a white-labeled extension of the NocoDB open-source platform, augmented with a custom Deploy Service microservice that provisions cloud infrastructure and deploys the backend to Railway with a single click. The architecture comprises two runtime components: (i) the Core Application, which manages schema definition, metadata persistence, and dynamic API generation through a metadata-driven request resolution engine that translates visual schema actions into real-time SQL operations and routable API endpoints; and (ii) the Deploy Service, which orchestrates Railway cloud provisioning via GraphQL mutations and CLI-based source deployment through a six-stage automated pipeline. Experimental evaluation confirms successful end-to-end deployment with real-time status tracking, automatic environment variable configuration, multi-stage Docker build assembly, and public URL generation—all without requiring any DevOps expertise from the user. The platform significantly reduces backend development time and removes the technical barrier for non-developers, students, and small teams seeking to prototype or deploy production-ready backend services rapidly.*

Key Words: No-Code Backend, Metadata-Driven API, Single-Click Deployment, Railway Cloud, NocoDB, Low-Code Platform, Automated DevOps, REST API Generation

1. INTRODUCTION

The current landscape of software engineering is defined by intense pressure to accelerate time-to-market. This imperative has catalyzed the rise of No-Code and Low-Code (NC/LC) platforms designed to democratize development and reduce dependency on specialized programming expertise. Surveys indicate that over 94% of enterprises across sectors now utilize low-code solutions, confirming their transition from experimental tools to foundational infrastructure [1].

Despite this adoption, a critical gap persists: existing NC/LC platforms rarely provide a seamless path from visual schema design to fully automated cloud deployment. Users must still navigate cloud consoles, configure CI/CD pipelines, manage environment variables, and handle containerization—tasks that require substantial DevOps knowledge and defeat the purpose of a no-code experience.

1.1 Problem Statement

Traditional backend development imposes high entry barriers through required expertise in server-side programming, infrastructure management, containerization, and deployment orchestration. Even existing NC/LC tools primarily address the frontend or partial automation, leaving deployment as a manual, error-prone step. Furthermore, the generated code in many AI-assisted or template-driven platforms lacks proper governance, security validation, and cloud-readiness [2][3].

This work addresses these limitations by presenting an integrated system where schema design, API generation, and cloud deployment form a single, uninterrupted workflow requiring no technical expertise beyond filling a deployment form.

2. RELATED WORK

Research in the NC/LC domain spans security governance, CI/CD automation, model-driven engineering, and

serverless deployment. We review the most relevant contributions.

Waqas et al. [4] conducted a multivocal review of NC/LC tools within DevOps pipelines, demonstrating measurable reductions in integration time, though noting gaps in architectural performance tuning. Our work directly addresses this gap through automated Railway provisioning. Lopez et al. [5] formalized security policy enforcement in low-code environments using visual guardrails, an approach we incorporate through automatic JWT secret generation and environment variable isolation for each deployed instance.

Kim et al. [6] showed that Model-Driven Engineering (MDE) can automate testing and deployment steps in enterprise platforms. Our metadata-driven API resolution engine extends this concept to real-time request handling. Nguyen et al. [7] proposed traceability mapping from requirements to serverless functions; we adapt this by linking each schema table directly to its generated REST endpoint. Rossi et al. [8] documented resilience patterns for serverless microservices; our deploy pipeline incorporates fallback domain generation to handle transient Railway provisioning failures.

Smith et al. [9] highlighted governance risks in LLM-generated backends; our template-driven, deterministic code generation avoids these pitfalls by using validated NocoDB runtime logic rather than generative AI for core API handling. Singh et al. [10] addressed schema consistency challenges in NC/LC data engines; our dual-persistence model—where every UI schema action simultaneously updates both metadata and physical database state—directly resolves this class of problem.

Table-1: Comparison of Backend Platforms

Platform	No-Code	1-Click Deploy	Custom White-Label	Open Source	DB Agnostic
Firebase	Partial	No	No	No	No
Supabase	Partial	No	No	Yes	Yes
Appwrite	Partial	No	Partial	Yes	Yes
NocoDB	Yes	No	No	Yes	Yes
Proposed System	Yes	Yes	Yes	Yes	Yes

3.1 Core Application

The Core Application consists of a Vue/Nuxt frontend (packages/nc-gui) and a Node.js/Express backend (packages/nocodb). Its three primary responsibilities are authentication and session management, schema definition with persistent state alignment, and dynamic API generation.

Authentication uses JWT tokens issued as xc-auth headers. These tokens authenticate both dashboard operations and the downstream Deploy Service's configuration retrieval, eliminating the need for re-authentication during deployment. Schema definition triggers a dual-persistence model: every user action (create table, add column, define relationship) simultaneously writes an abstract metadata record and executes the corresponding DDL/DML operation on the physical database. This ensures the abstract definition and physical schema are always synchronized—a prerequisite for correct API behavior.

API generation is entirely metadata-driven. At request time, the runtime resolves: (i) authentication and permissions; (ii) project/table/view metadata; (iii) request intent (CRUD); (iv) query translation for the configured database client; and (v) response shaping with relation expansion. Generic CRUD handlers consult current metadata to determine validation rules, type coercion, and relational joins, ensuring that the API surface updates automatically upon any schema change without requiring code redeployment.

3.2 Deploy Service

The Deploy Service is a standalone Express.js microservice (port 3001) that bridges the No-Code Backend application and the Railway cloud platform. It exposes three endpoints: POST /api/deploy for initiating deployment, GET /api/deploy/:id/status for status polling, and GET /api/deploy for listing deployments.

Deployment is executed through a six-stage pipeline: (1) Project configuration extraction via NocoDB internal RPC using the passed authToken; (2) Railway project creation via GraphQL mutation projectCreate; (3) Service creation within the new Railway project; (4) Environment variable injection (PORT, NC_AUTH_JWT_SECRET, NC_DB, extraEnv); (5) Public domain generation via serviceDomainCreate; and (6) Docker context assembly and CLI-based source upload via railway up. A project-scoped token is created via projectTokenCreate to authenticate the CLI deployment, ensuring correct context isolation.

3.3 Docker Build Context

The assembled deployment context includes a dynamically generated multi-stage Dockerfile. Stage 1 (backend-builder) compiles TypeScript and bundles assets via Webpack using node:16-alpine. Stage 2 (runner) installs production dependencies, copies compiled artifacts,

injects the white-labeled UI bundle, and sets the entry point to docker/index.js. This source-based approach eliminates any dependency on external container registries.

4.2 Metadata-Driven API Resolution

The central innovation of the Core Application is its metadata-driven request resolution. When an API request arrives, middleware validates the xc-auth JWT and checks permissions against project and table metadata. A robust query builder then translates abstract parameters—filters, pagination, sorting—into optimized SQL tailored for the configured database client. The result is normalized and returned as a standardized JSON response. This architecture means that defining a new table in the UI instantly creates a fully functional REST endpoint for CRUD and relational operations with no additional code.

4.3 Deployment Pipeline Details

The POST /api/deploy endpoint validates four required parameters: nocodbUrl, authToken, projectId, and railwayToken. The collectProjectConfig function uses fallback resolution (xcProjectGetConfig → projectGetById → projectGet) to robustly retrieve database connection details and build the NC_DB string in the format: <client>://<host>:<port>?u=<user>&p=<password>&d=<database>. After Railway infrastructure is provisioned and environment variables are injected, the assembleDeployDir function creates a temporary directory containing the Dockerfile, backend source, and branded UI bundle. The railway up command is then executed in this directory with RAILWAY_TOKEN set to the project-scoped token. Deployment status is tracked in an in-memory Map and exposed via the polling endpoint.

4.4 User Interface

The deployment workflow is initiated from the project list page via a custom rocket-icon deploy button. This button opens the Deploy Wizard UI, passing projectId, projectTitle, nocodbUrl, and authToken as query parameters. The wizard presents a three-step interface: Review (auto-populated project details), Configure (Railway token and optional settings), and Deploy (real-time status with BUILDING, DEPLOYING, and SUCCESS states). Upon success, the final public Railway URL is displayed to the user.

5. EXPERIMENTAL RESULTS AND DISCUSSION

5.1 Experimental Setup

The system was tested in a controlled development environment with the NocoDB instance running on port 8081 and the Deploy Service on port 3001. Multiple deployment scenarios were executed: projects with SQLite, MySQL, and PostgreSQL backends; deployments with and without extra environment variables; and sequential deployments to verify independent Railway project creation. The Railway cloud platform was used for all cloud deployment tests, accessed via both its GraphQL API and CLI.

5.2 Results

All tested configurations successfully completed the six-stage deployment pipeline. The system correctly extracted project configurations across all three fallback API paths, generated valid NC_DB connection strings, created isolated Railway projects and services, injected environment variables without leakage between deployments, assembled valid multi-stage Docker contexts, and executed railway up without path or token context errors. Real-time status polling accurately reflected Railway deployment states (BUILDING → DEPLOYING → SUCCESS), and final public URLs were retrieved and displayed to users upon successful deployment. The white-labeled UI was confirmed present in deployed instances. The full end-to-end workflow—from clicking the deploy button to receiving a live URL—completed within Railway's standard build time, with no additional user interaction required after the initial wizard configuration.

5.3 Discussion

The modular separation between the Core Application and Deploy Service proved effective: schema changes and API requests to the Core Application were unaffected during deployment operations. The in-memory deployment tracking, while functional for development use, represents a known limitation—deployment history is lost on service restart. The requirement for a Railway API token is an expected UX friction point for first-time users. These limitations are addressable through persistent storage and OAuth-based token management, as detailed in Section 6. Compared to existing platforms (Firebase, Supabase, Appwrite, NocoDB), the proposed system uniquely combines a no-code visual builder with automatic REST API generation and a fully automated, one-click cloud deployment pipeline, without requiring any cloud account pre-configuration beyond providing an API token.

6. CONCLUSIONS

This paper presented a No-Code Backend platform that integrates visual schema design, metadata-driven REST API generation, and single-click cloud deployment into a unified, user-friendly workflow. The system successfully eliminates the need for server-side programming, infrastructure configuration, and DevOps expertise for backend deployment. The dual-persistence model ensures schema-API consistency, while the six-stage automated

deploy pipeline reliably provisions and deploys instances to Railway with no manual steps.

Experimental evaluation confirmed the system's effectiveness across diverse project configurations. Future work will focus on: (i) replacing in-memory deployment tracking with persistent database storage; (ii) extending cloud platform support beyond Railway to AWS, GCP, and Azure; (iii) implementing OAuth-based Railway authentication to remove manual token entry; (iv) adding CI/CD pipeline integration and version control support; and (v) introducing role-based team collaboration features for enterprise use.

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of those whose encouragement and guidance served as a beacon throughout this work. Success is the result of hard work and perseverance, but steadfast of all is encouraging guidance.

The authors would like to express their deepest and most sincere gratitude to Mrs S. Vijaya Lakshmi, Assistant Professor, Department of Computer Science and Engineering, Mahatma Gandhi Institute of Technology (A), Hyderabad, and Mr B. Chandrasekhar, Assistant Professor, Department of Computer Science and Engineering, MGIT, for their constant mentorship, invaluable technical insights, unwavering encouragement, and moral support at every stage of this project. Their timely feedback and constructive suggestions were instrumental in shaping the direction and quality of this work.

The authors extend their heartfelt thanks to Dr. C.R.K. Reddy, Professor and Head of the Department, CSE, MGIT, for his timely support, valuable suggestions, and for fostering a research-oriented environment within the department that motivated this work.

The authors also wish to express sincere gratitude to Prof. G. Chandra Mohan Reddy, Principal, Mahatma Gandhi Institute of Technology (A), Hyderabad, for providing excellent working facilities, computational resources, and an environment conducive to innovative project development.

Special thanks are due to all the faculty members and staff of the Department of Computer Science and Engineering, MGIT, who directly or indirectly supported this work through their guidance, resources, and encouragement throughout the course of the project. The authors are also grateful to their families and peers for their patience and continuous motivation during the development and writing of this paper.

REFERENCES

[1] Waqas, M., Ali, Z., Sánchez-Gordón, M., & Kristiansen, M. (2024). Using Low-Code and No-Code Tools in DevOps: A Multivocal Literature Review. DevOps and Low-code/No-code Tools.

[2] Johnson, C., et al. (2025). DeVAIC: Detecting Vulnerabilities in AI-Generated Python Code. International Journal on Science and Technology, 16. doi:10.71097/ijsat.v16.i1.3718

[3] Smith, B., et al. (2024). Generative AI and LLM-Based Applications: Governance and Risk. IEEE Computer, 58. doi:10.1109/MC.2025.3547073

[4] Waqas, M. et al. (2024). Using Low-Code and No-Code Tools in DevOps. Chapter, Feb 2024.

[5] Lopez, F., et al. (2024). Formalizing Security Policy Enforcement in Low-Code Platforms. IEEE Transactions on Software Engineering.

[6] Kim, E., et al. (2023). Model-Driven Development and Continuous Delivery in Enterprise Systems. ACM International Conference Proceedings.

[7] Nguyen, J., et al. (2023). Enhancing Traceability from Requirements to Serverless Functions. Springer Lecture Notes in Computer Science.

[8] Rossi, N., et al. (2023). Resilience Patterns for Serverless Microservices in Low-Code Environments. IEEE Cloud Computing.

[9] Smith, B., et al. (2024). Generative AI and LLM-Based Applications: Governance and Risk. IEEE Computer.

[10] Singh, Q., et al. (2025). Dynamic Schema Considerations in Low-Code Data Processing Engines. ScienceDirect Journal of Systems and Software.

[11] Perez, P., et al. (2025). AI-Native Software Engineering: Impact on Software Development Lifecycle. IEEE Computer, 58, 101-104.

[12] Gao, M., et al. (2024). Low-Code AgentScope: Multi-Agent Platform for API Integration. ArXiv. doi:10.48550/arXiv.2501.14755v2

[13] Bui, N.D.Q., et al. (2023). CodeTF: One-stop Transformer Library for State-of-the-art Code LLM. ArXiv. doi:10.48550/arXiv.2306.00029

[14] Author A, et al. (2022). Stateful Serverless Computing with Crucial. ACM Transactions on Software Engineering and Methodology, 31(3).