Research Article *Open Access*

WM PUBLICATIONS

Journal of Scientific Engineering Advances

ISSN: 3071-0723

DOI: doi.org/10.63721/26JSEA0142

Development and Implementation of a REST API for Project Management with Hapi.js and MySQL

Marco Antonio Celis Crisostomo[1*], Francisco Miguel Hernández Lopez[2], Emmanuel Vega Negrete[2] and Jorge Alberto Cárdenas Magaña[2]

[1]Department of Sustainable Agricultural Innovation, National Technological Institute of Mexico / José Mario Molina Pasquel y Henríquez Technological Institute, Tamazula Academic Unit, Mexico

[2]Department of Electromechanical Engineering, National Technological Institute of Mexico / José Mario Molina Pasquel y Henríquez Technological Institute, Tamazula Academic Unit, Mexico

Citation: *Marco Antonio Celis Crisostomo, Francisco Miguel Hernández Lopez, Emmanuel Vega Negrete, Jorge Alberto Cárdenas Magaña (2026) Development and Implementation of a REST API for Project Management with Hapi.js and MySQL. J. of Sci Eng Advances 2(3) 1-6. WMJ/JSEA-142*

Abstract

In this article, the design, development, and implementation of a Representational State Transfer Application Programming Interface (REST API) aimed at the comprehensive management of collaborative projects in organizational environments is described. The solution was built using the Hapi.js framework within the Node.js ecosystem, supported by a MySQL relational database, and incorporates robust authentication mechanisms, role-based authorization, and structured validation of input data. The system architecture includes five interconnected functional modules: user management with status and role control, project administration with status tracking, task management with priority levels, chronological tracking records, and document attachment control linked to both projects and specific tasks.

Regarding security, the platform uses password encryption through bcrypt, stateless authentication with JSON Web Tokens (JWT), and a password recovery workflow that sends a temporary key via email. To validate all incoming data, the Joi library was used, ensuring that the information is correct before performing any database operations. The UUID standard was employed for the creation of unique identifiers within the system resources.

The normalized relational database design supports five main entities (users, projects, tasks, tracking records, and documents), establishing referential integrity relationships that ensure information consistency. The obtained results demonstrate that the proposed solution significantly reduces access and information manipulation times compared to traditional monolithic systems, provides a uniform and predictable interface based on the HTTP protocol, and facilitates integration with different types of clients such as web, mobile, or desktop applications. Hapi.js was chosen because its route configuration is declarative and because it offers native schema validation, which helps make the code easier to maintain and scale.

***Corresponding author**: Marco Antonio Celis Crisostomo, Department of Sustainable Agricultural Innovation, National Technological Institute of Mexico / José Mario Molina Pasquel y Henríquez Technological Institute, Tamazula Academic Unit, Mexico.

Submitted: 22.05.2026 **Accepted**: 26.05.2026 **Published**: 05.06.2026

Keywords: REST, API, Hapi.Js, MySQL, Node.Js

Introduction

In the current environment of digital transformation, organizations are required to implement information systems that enable them to coordinate distributed work teams, monitor the progress of strategic initiatives, and ensure the traceability of every activity throughout the project lifecycle. Project management has progressively evolved from the use of spreadsheets and shared documents toward specialized platforms that centralize information, facilitate communication among team members, and generate auditable records of all modifications performed [1].

RESTful APIs have become the dominant paradigm for data exchange among heterogeneous systems. They have been widely adopted due to their conceptual simplicity, the use of the HTTP protocol as a universal transport layer, and their technological independence from the clients consuming them [2]. Currently, globally recognized organizations such as GitHub, Stripe, Twilio, and Google expose their core services through REST APIs, which has promoted the standardization of design patterns and best practices in API development [3].

Within the Node.js ecosystem, numerous frameworks are focused on API development. Among the most prominent are Express.js, Fastify, NestJS, and Hapi.js. Hapi.js is characterized by its explicit configuration philosophy, in contrast to the minimalist approach of Express.js, resulting in a system where each route, validation method, and security policy is deliberately and transparently declared [4]. This characteristic makes it a particularly suitable option for enterprise projects in which code traceability and response standardization are essential requirements.

The topic addressed in this work originated from the needs identified in medium-sized organizations (companies) that lack a unified system for project management, including task assignment, progress documentation, and access control for different user profiles. These organizations commonly rely on diverse combinations of tools—emails, spreadsheets, and instant messaging platforms—whose lack of integration generates information duplication, difficulties in change auditing, and security risks associated with the improper handling of credentials [5].

In this context, the article presents the development of a secure and comprehensive REST API intended to serve as the central backend for a project management application. The solution addresses the following specific objectives: (1) to design a normalized relational database schema that adequately models the domain entities; (2) to implement a role-based authentication and authorization system ensuring that each user can access only the resources corresponding to their privileges; (3) to provide well-defined endpoints for CRUD operations related to projects, tasks, users, tracking records, and documents; and (4) to guarantee exhaustive validation of all incoming data in order to preserve system integrity.

Materials and Methods

General System Architecture

The solution implemented a loosely coupled layered architecture in which the API server acts as a bridge between client applications and the data persistence layer. This separation of responsibilities allows any type of client (single-page web application, native mobile application, or desktop client) to interact with the system through standard HTTP requests without needing to know the internal implementation details of the server [6].

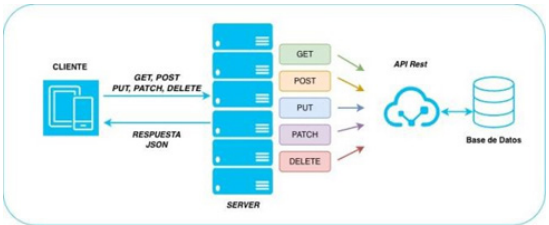

Figure 1: API Architecture

The architecture follows the REST principles defined by Roy Fielding [2]: a uniform interface based on resources identified through URIs, stateless communication in which each request contains all the information required to be processed, the use of HTTP methods (GET, POST, PUT, PATCH, DELETE) to express the semantics of operations, and the transfer of representations in JSON format. Each system resource (user, project, task, tracking record, document) is available through a predictable URL consistent with REST conventions.

Technology Stack

The main dependencies used and the justification for their selection are described below.

Hapi.js (version 21.4.7) was selected as the server framework due to its plugin system, built-in route validation, and explicit configuration approach. Unlike Express.js, Hapi.js does not rely on third-party middleware for key functionalities such as payload validation or structured error handling, thereby reducing the attack surface and simplifying the dependency chain [4, 8].

MySQL (using the mysql2 driver version 3.20.0) was selected as the relational database management system because of its maturity, broad production support, and the inherently relational nature of the project management domain, where entities contain well-defined foreign key relationships that benefit from the ACID guarantees provided by the InnoDB engine [9].

bcrypt (version 6.0.0) uses the adaptive Blowfish hashing algorithm, considered an industry standard for secure password storage due to its resistance to brute-force attacks through the use of a configurable cost factor [11]. jsonwebtoken (version 9.0.3) is responsible for creating and verifying JSON Web Tokens, enabling stateless authentication without the need to store sessions on the server.

Database Design

The relational schema was designed following normalization principles up to the Third Normal Form (3NF), eliminating redundancies and ensuring that each attribute depends exclusively on the primary key of its entity [9]. The model includes five main tables.

Figure 2: Database Schema

The users table stores the identity information of each person registered in the system. The role field is defined as an ENUM with three possible values: admin, manager, and citizen, enabling the implementation of Role-Based Access Control (RBAC). The status field manages the account lifecycle through the values pending, active, blocked, and deleted, ensuring that a newly registered user must complete an activation process before being allowed to operate within the system. The temp_password and recovery fields support the password recovery workflow: when a user requests a password reset, an encrypted temporary password is generated and stored in temp_password, while the recovery field is set to 1; after authentication using the temporary password, both fields are cleared.

The projects table models the main initiatives of the system. The id_manager field establishes a foreign key relationship with the users table, assigning a primary manager to each project. The project status evolves through the values planning, in_progress, and completed, providing a clear view of the lifecycle of each initiative. The start_date and end_date fields enable temporal control of projects.

The tasks table breaks down projects into concrete units of work. Each task is mandatorily linked to a project through the id_project field. The priority field (high, medium, low) facilitates work planning, while the status field (pending, in_progress, completed) reflects the individual progress of each activity. This level of granularity allows teams to plan and monitor work in detail.

The tracking table implements a collaborative activity log. Each record associates a textual comment with a specific user and a particular task, together with an automatically generated timestamp. This table acts as an audit log documenting who made each comment and at what time, providing complete traceability of task progress.

The documents table allows files to be attached to projects or tasks. The design considers that a document may be associated either with a project (id_project) or a task (id_task), with both relationships being optional to provide greater flexibility. The type field classifies the file as pdf, image, or other, facilitating filtering operations during queries.

API Functional Modules

The API is organized into independent route-based modules, allowing the system to maintain a scalable, secure, and easily manageable structure. Each module groups endpoints related to a specific system functionality, facilitating separation of responsibilities and code maintenance.

Module	Main Route	Description
Authentication	/auth	Manages user registration, login, JWT generation, and password recovery through temporary keys.
Authentication	/auth	Manages user registration, login, JWT generation, and password recovery through temporary keys.
Projects	/proyectos	Manages the creation, retrieval, updating, and status changes of projects registered in the system.
Tasks	/proyectos/{id }/tareas	Manages tasks linked to a specific project, including priorities, statuses, and operational tracking.
Tracking	/tareas/{id}/se guimientos	Records comments and observations associated with tasks to maintain chronological traceability of the performed work.
Documents	/documentos	Controls the registration and association of attached files related to projects or tasks within the platform.

The modular route structure allows the API to maintain an organized communication flow between clients and backend services, facilitating component reuse and the implementation of centralized security controls. Furthermore, the separation of functionalities simplifies integration with web or mobile applications and enables future system expansions without affecting the stability of existing modules.

Development Methodology

The project was developed following an iterative and incremental methodology, organizing the work into two-week sprints. In each iteration, a complete functional module was developed, from database migration to endpoint implementation and testing, ensuring the continuous delivery of functionality. Version control was managed using Git, following the GitFlow workflow for handling development branches, features, and fixes [17]. The endpoints were tested using Postman, documenting each test case with its input parameters and expected responses.

Results and Discussion

Server Implementation and Route Configuration

The Hapi.js server was configured with route validation options globally enabled, ensuring that any route parameter, query string, or payload that did not comply with the defined schema was automatically rejected through descriptive error messages. The implemented route structure strictly followed REST conventions, where resources are represented by plural nouns (/ users, /projects, /tasks), HTTP methods define the operation to be executed, and route parameters identify specific resources (/v1/projects/ {project_id}/ tasks/ {task_id}). In addition, all routes included the version identifier "v1," allowing compatibility to be maintained with future API updates without affecting existing implementations.

Hapi.js plugins enabled the JWT authentication scheme to be registered as a reusable component, allowing each route to define its authentication policy within a single configuration line. This declarative approach, compared to the manual incorporation of middleware into each route in Express.js, significantly reduced

code duplication and minimized the risk of exposing sensitive routes [4, 8]. Furthermore, each group of endpoints was independently organized and stored within the Git repository, improving version control, facilitating modular maintenance, and supporting the progressive evolution of the system.

Additionally, semantically correct HTTP responses were implemented across all endpoints: status code 200 for successful queries, 201 for successfully created records, 204 for deletions without response content, 400 for validation errors, 401 for invalid credentials, 403 for role-restricted access, and 404 for nonexistent resources. Status codes were handled uniformly, simplifying integration with clients capable of automatically managing errors based on HTTP responses.

API Connectivity Testing

In order to verify the correct operation of the developed REST API, the Postman tool was used as a testing environment for each implemented route. Within this platform, an organized collection of endpoints grouped by functional modules was created to enable the structured execution of operations related to authentication, users, projects, tasks, tracking records, and documents. Each request was configured with its corresponding HTTP methods, headers, parameters, and request bodies in order to validate the expected behavior of the system under different usage scenarios.

The tests verified both the correct transmission of information to the server and the responses generated by the API. For this purpose, HTTP status codes, returned JSON structures, JWT-based authentication, and error messages associated with validation or access restrictions were validated. Tests were performed using both valid and invalid data to ensure that the system responded consistently to validation errors, incorrect credentials, and incomplete requests, thereby guaranteeing the robustness and reliability of the implemented endpoints.

The use of Postman collections also facilitated the maintenance and evolution process of the project, since it allowed each test performed during development to be documented and reused. This organization enabled test cases to be re-executed after modifications to the API, ensuring that new developments did not affect existing functionalities. This approach contributed to more precise control over system behavior and helped guarantee the stability of each released version of the API.

Conclusions

This work demonstrated that it is possible to design and implement a complete, secure, and maintainable REST API for comprehensive project management using Hapi.js and MySQL with an open-source technology stack supported by a broad developer community. The proposed architecture satisfactorily fulfilled the four specific objectives established in this study: the normalized relational design guarantees data integrity; the JWT-based RBAC system ensures that each user operates within the limits of their assigned privileges; the CRUD endpoints cover all necessary operations for the five domain entities; and Joi validation prevents the entry of malformed or malicious data.

The selection of Hapi.js proved to be an excellent decision for this type of project. Its philosophy based on explicit configuration required every design decision to be clearly documented within the source code, resulting in a system with predictable and auditable behavior. In addition, the native integration of schema validation reduced the amount of repetitive infrastructure code, allowing development efforts to focus on the business logic specific to each module.

As future work, the first proposed improvement is the implementation of a caching layer using Redis for high-frequency queries such as active project listings, which would reduce database load in environments with many concurrent users. Second, the development of a real-time notification module through WebSockets is proposed to alert assigned users whenever a tracking record is added or the status of a task is modified. Third, the incorporation of automated testing with code coverage as part of the continuous integration pipeline is suggested to guarantee zero regression in future system modifications.

Ultimately, the architecture documented in this article may serve as a methodological reference for the development of REST APIs in academic and industrial projects of similar complexity, both because of the clarity of its design and the accessibility of the employed technology stack.

Acknowledgment

The authors would like to express their sincere gratitude to the companies and developer communities that create and maintain coding platforms such as Visual Studio Code, as well as the open-source technologies and programming languages used throughout this project, including JavaScript, SQL, Node.js, Hapi.js, MySQL, JSON Web Token (JWT), and related open-source development tools. Their continuous contributions to the software development ecosystem provide accessible and efficient resources for research, innovation, and technological advancement.

Special thanks are also extended to the Instituto Tecnológico José Mario Molina Pasquel y Henríquez for the allocation of time, facilities, and academic spaces that enabled the development of this research work. The institutional support provided significantly contributed to the successful completion of this project.

References

1. Project Management Institute (2000) A guide to the project management body of knowledge (PMBOK Guide). Project Management Institute. https://caricom.org/wp-content/uploads/PMI_Project_Management_Body_of_Knowledge_Guide.pdf.
2. Fielding R T (2000) Architectural styles and the design of network-based software architectures. University of California, Irvine. https://roy.gbiv.com/pubs/dissertation/fielding_dissertation.pdf.
3. Richardson L, Amundsen M, Ruby S (2013) RESTful web APIs: services for a changing world. " O'Reilly Media, Inc https://www.oreilly.com/library/view/restful-web-apis/9781449359713/.
4. Hapi.js Core Team (2024) Hapi.js-The Simple, Secure Framework Node.js Developers Trust. Recuperado de https://hapi.dev/.
5. Kerzner H (2025) Project management: a systems approach to planning, scheduling, and controlling. John Wiley & Sons. https://download.e-bookshelf.de/download/0009/5864/93/L-G-0009586493-0018377107.pdf,
6. Fowler M (2012) Patterns of enterprise application architecture. Addison-Wesley. https://sar.ac.id/stmik_ebook/prog_file_file/EFCofwzsj0.pdf.
7. Tilkov S, Vinoski S (2010) Node. js: Using JavaScript to build high-performance network programs. IEEE Internet Computing 14: 80-83.
8. Casciaro M, Mammino L (2020) Node.js Design Patterns, 3rd ed. Packt Publishing. https://www.packtpub.com/en-us/product/nodejs-design-patterns-9781839214110.
9. Codd E F (1970) A relational model of data for large shared data banks. Communications of the ACM 13: 377-387.
10. Joi Core Team (2024) The Most Powerful Schema Description Language and Data Validator for JavaScript. Recuperado de https://joi.dev/.
11. Provos N, Mazières D (1999) A future-adaptable password scheme. In Proceedings of the USENIX Annual Technical Conference. FREENIX Track 81-91.
12. Jones M, Bradley J, Sakimura N (2015) JSON Web Token (JWT). RFC 7519. Internet Engineering Task Force (IETF). https://datatracker.ietf.org/doc/html/rfc7519.
13. Nodemailer Contributors (2024) Nodemailer: Send Email from Node.js. Recuperado de https://nodemailer.com/.
14. Leach P, Mealling M, Salz R (2005) A Universally Unique IDentifier (UUID) URN Namespace. RFC 4122, IETF https://datatracker.ietf.org/doc/html/rfc4122.
15. Wiggins A (2017) The Twelve-Factor App. Heroku. Recuperado de https://12factor.net/.
16. OWASP Foundation (2023) OWASP Top Ten 2023. Recuperado de https://owasp.org/www-project-top-ten/.
17. Driessen V (2010) A Successful Git Branching Model. Recuperado de https://nvie.com/posts/a-successful-git-branching-model/.
18. Ell T (2023) Node.js framework benchmarks: Fastify, Express, Hapi comparison. Journal of Web Engineering 22: 541-562.
19. Masse M (2011) REST API Design Rulebook. O'Reilly Media https://www.oreilly.com/library/view/rest-api-design/9781449317904/.
20. mysql2 Contributors (2024) mysql2: Fast MySQL driver for Node.js. Recuperado de https://github.com/sidorares/node-mysql2.