QuikAPIs - Automated API Generationand DataManagement Platform with Built-insecurity

Rishab Pendam[1*],Mahek Upadhye[1], Nilesh Patil[2],Sridhar Iyer[2]

[1]Computer Engineering, Dwarkadas J. Sanghvi College of Engineering, Mumbai, India
[2]Professor, Computer Engineering, Dwarkadas J. Sanghvi College of Engineering, Mumbai, India
*Corresponding author: rishabpendam@gmail.com

Received:19.04.2024 **Revised: 25.05.2024** **Accepted: 24.06.2024**

ABSTRACT
The demand for web applications has increased in such a way that efficient and user-friendly solutions for backend development need to be created, especially for less technically inclined people. This paper intends to discuss the design of a novel website platform that will automate the creation of backend application programming interface (API) significantly, reducing the complexity involved in database and API creation. It allows the user to register, create custom databases by specifying attributes like name, datatype, uniqueness, and mandatory fields, and have instantaneous, working CRUD APIs with respect to their database. Zero code is given as a solution, and allows the user to create the backend structure needed for full-stack projects with ease and without even a line of code to be written for the backend. The platform bridges the gap for nontechnical users and backend development, or those who want an easy, efficient, and scalable way of developing websites. In this paper, the architecture and design considerations of the platform are outlined, which could be regarded as democratizing web development.

Keywords:API generation platform, CRUD operations, Node.js, MongoDB, Database automation, Encryption using AES

1. INTRODUCTION

In modern web development, one often has to master a long list of both frontend and backend technologies in order to get a full-fledged application up and running. The process of setting up databases and APIs—especially for individuals just starting to program or with less experience in programming—actually represents a very serious bottleneck. By convention, the creation and management of APIs over databases involve complex code writing, server configuration, and smooth data operations, reserved for technically savvy people.

This paper proposes a solution for easing the development of the backend by automatically creating CRUD APIs. On this platform, a user will be able to create an account, customize a database by just specifying the fields of a database, like name, type, unique, and whether it is required or not, and hence generate APIs which automatically work with that very database. Built with the use of modern technologies such as React on the frontend, Node.js in the backend, and MongoDB for handling databases. This no-code platform will enable users to build backend infrastructures in an extremely easy way by dragging and dropping components without the need to write code.

This is taken care of by making sure the data containing user information is encoded, as in databases and API details, for privacy and security. The encoding ensures that the data is personal to each user and does not get intercepted or used by some other entity without due permission. By offering this sort of encryption mechanism itself, the platform assures users confidentiality and security regarding their data.

It also supports a section that gives users a step-by-step guide on how to integrate the auto-generated APIs into their frontend code. In addition, it has a database management dashboard to keep the overview of databases by its users. In this interface, a user is able to create, view, edit, and delete entries from their database through the website and is guaranteed to be in full control of their data without having to touch the backend code.

The cardinal idea of this web-based server was to lower the barrier to entry with which one makes web development conceivable to users who have no significant technical skills to produce a full-stack application. The key benefits of such a solution are that it will save time by automatically generating back-end APIs, provide extensive usage guidance, and ensure the security of user data. It would allow users to concentrate their efforts on other aspects of Web development, such as the user experience and front-end design. This paper discusses the architecture and design of the platform that sets it apart and which promises to make a paradigm shift in both the workflows of non-technical users and those of developers

2. RELATEDWORKS

[1] examines the impact of Low/No-Code (LNC) development on digital transformation and software development. The study analyzes benefits such as rapid development and citizen programming, alongside limitations including customizability issues. The author reviews major LNC platforms, highlighting the integration of AI/ML as a key trend. The paper's discussion of LNC's role in democratizing development aligns with the research undertaken in this paper on automatic API generation. The paper underscores the growing importance of accessible development tools in driving digital transformation across industries.

Low-Code No-Code (LCNC) platforms have rapidly evolved as tools for simplifying complex software development, enabling broader accessibility. [2] highlight the use of platforms such as Microsoft Power Platform, SetXRM, and vf-OS for tasks ranging from social media and global news analysis to disaster prediction and process digitization. These platforms allow for automation in data collection and AI-driven insights, significantly reducing the need for manual coding. The principles behind LCNC align closely with automatic API generation, where democratizing backend development through automation enables both developers and non-developers to build and manage APIs efficiently.

The authors in [3] provide an in-depth analysis of Spring Boot's core concepts and architectural principles. Their study highlights how Spring Boot reduces redundant coding through its integration capabilities with Spring Data JPA and relational databases. By streamlining data storage and access, Spring Boot enhances backend efficiency and simplifies the development process. The insights provided in this paper are instrumental for understanding how modern frameworks like Spring Boot contribute to efficient backend development, complementing the objectives of automating backend processes and democratizing development through tools and methodologies.

[4] explore the potential of low-code platforms (LCPs) in simplifying and democratizing the development of recommender systems. Their work addresses the challenges faced by developers in creating tailored recommender systems due to the complexity and specialized knowledge required. By leveraging LCPs, the authors demonstrate how graphical interfaces and drag-and-drop utilities can streamline the development process. Their approach involves re-implementing well-established recommender systems using a low-code paradigm, highlighting the feasibility of developing these systems with very minimal coding.

The study presented in [5] offers a comprehensive exploration of low-code and no-code development platforms, emphasizing their transformative impact on software development. By analyzing the benefits and limitations of these platforms, the authors highlight their role in accelerating development cycles and fostering innovation. The study also anticipates the future effects of low-code and no-code solutions on industries and society, noting their potential to reshape digital transformation and software development practices.

[6] discusses the transformative potential of no-code and low-code software development platforms, highlighting their role in accelerating application development and democratizing programming. The article illustrates how these platforms, such as those used in New York City's and Washington, DC's COVID-19 response, allow for rapid deployment of solutions without traditional coding. The article also touches on the evolution of programming from assembly languages to more intuitive, high-level coding environments, and considers future advancements such as AI-driven code generation.

[7] provides a comprehensive gray literature review on the adoption of low-code and no-code technologies, driven by the accelerated digitalization due to the COVID-19 pandemic. The study addresses the growing software demand and the challenge of meeting it amidst a developer shortage. These low-code and no-code tools, which allow users with minimal programming knowledge to build applications via graphical interfaces, are explored for their benefits and limitations.

[8] examine developer challenges with low-code software development (LCSD) by analyzing 5,000 Stack Overflow posts. They identify 13 topics, including customization, platform adoption, and integration, with a focus on dynamic event handling. The study highlights significant difficulties in development and automated testing phases, offering insights for improving LCSD platforms and practices.

This study in [9] explores no-code web development, highlighting its role in enabling non-programmers to build web applications. Using both qualitative and quantitative methods, the study shows that no-code platforms simplify development and broaden accessibility, though they also present certain challenges.

3. METHODOLOGY

The proposed platform aims to ease the backend development process by eliminating the need for the user to create a database and an API manually and also the time and expertise needed to do so. This approach describes the basic phases of the user specific process of on-boarding, creating a database, allowing generation of application programming interfaces, and managing databases. Each stage is clearly defined enabling users to create complete backend support systems without the need to write or understand any backend programming.

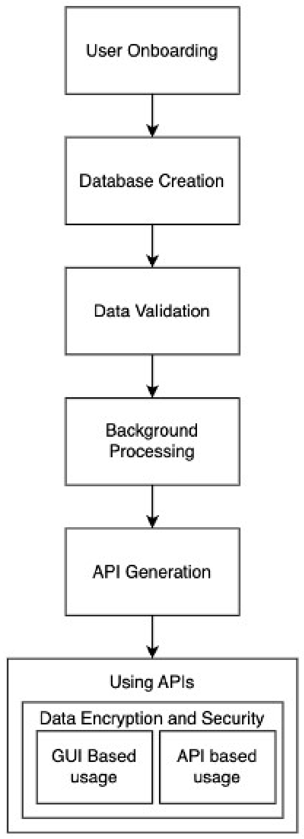


Figure 1. Proposed Methodology

3.1 User Onboarding

An onboarding process is necessary for the verification of users so that only verified users can use or access the platform. Security and privacy will be assured through the use of various techniques such as OTP and email verification, authentication through JWT tokens, and password hashing sensitive data. This assurance allows for verifiable user identity with secure data.

3.1.1 Nodemailer

Nodemailer is a Node.js module applied to send emails with ease. In this project, Nodemailer plays a key role in sending OTPs to users at the time of registration or verification. As soon as the user submits the registration details, an OTP will be sent to the email address provided by him. In turn, he has to input the OTP for verification to complete the registration process. This ensures that only genuine users get access to the platform.

Nodemailer uses a Transporter object that defines the service and authentication to send emails. The transport is used to send OTPs, as well as other forms of email communication. Following is a brief overview of sending emails using Nodemailer logic in the platform:

1. The transporter is set up by the Gmail service and the credentials for the platform.
2. An email is prepared and sent to the email address of the user, with information about the OTP besides other details.
3. The OTP is time-bound, meaning verification should be done within the stipulated time.

This no doubt helps in ensuring that a user's email has been verified and that no unauthorized access has been given.

3.1.2 JSON Web Token (JWT)

JWT (JSON Web Token) is compact, URL-safe means to represent claims between two parties. The application makes use of JWT to verify and authenticate users upon successful registration and login. Upon successful login, a JWT token is created and returned to the user. This same token is then used in API calls by him to authenticate subsequent actions taken by him within the platform. These involve:

1. The creation of the token when the user logs in, signed by a secret key.
2. Checking of the token each time by the user's request with the help of a middleware. Here, the

middleware verifies whether the token is valid or not and verifies if the request has come from the authenticated user or not.

3. Store the user information in the token like User ID for the identification of the user without showing sensitive password.

One of the popular ways of protecting API endpoints is JWT, and this introduces an additional layer of security that ensures system resources are accessed only by users.

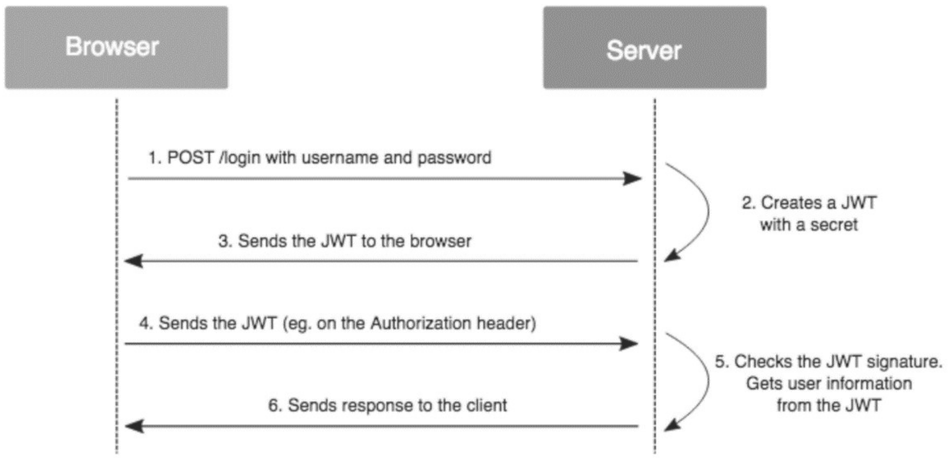

Browser
Server
1. POST /login with username and password
2. Creates a JWT with a secret
3. Sends the JWT to the browser
4. Sends the JWT (eg. on the Authorization header)
5. Checks the JWT signature. Gets user information from the JWT
6. Sends response to the client

Figure 2. Overview of JWT Usage

3.1.3 Bcrypt for Password Hashing

Bcrypt is a password-hashing function designed to be computationally intensive and resistant to brute-force attacks. Bcrypt is used in this project for the storage of user passwords through hashing before storing in the database. This means that should the database becompromised, the actual passwords will still remain secure since Bcrypt hashes cannot easily be reversed.

The process works as follows in the platform:

1. This is utilized by Bcrypt for hashing a given password whenever a user creates an account or updates their password. It then saves the hash into the database.
2. Whenever the user logs in, a similar hash of the entered password is performed and compared with the stored hash, guaranteeing that the match of passwords never includes the actual password in plaintext.

The platform utilizes Bcrypt to ensure that even in the events of a data breach or intrusion, users' passwords remain private.

Email and OTP verification using Nodemailer, authentication of users via JWT token, and password hashing using Bcrypt are all integrated to ensure the onboarding is safe. This safeguards the information of the users from unauthorized access and ensures that sensitive data is encrypted. This becomes very important information for each user in generating private APIs and safely handling the user's data on the platform.

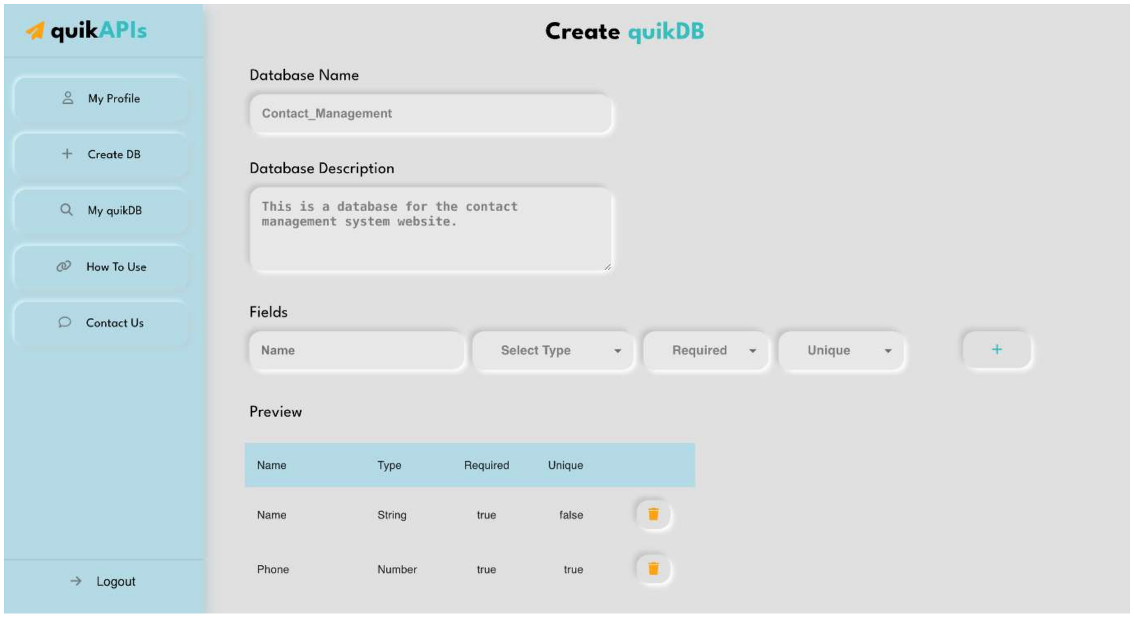
3.2 Database Creation

Creating a database is an essential part of making it possible for users to define the structure of their data. This has been done in a way that it is as intuitive as possible and easy to do using a user-friendly graphical interface, or GUI, that will walk a user through defining their database schema, without them needing to actually write any kind of backend code. Made for people with little technical knowledge who would be able to develop the result in the form of a complete, working backend by adding details to the form fields themselves.

3.2.1 User Input for Database Creation

Creating a database is an essential part of making it possible for users to define the structure of their data. This has been done in a way that it is as intuitive as possible and easy to do using a user-friendly graphical interface, or GUI, that will walk a user through defining their database schema, without them needing to actually write any kind of backend code. Made for people with little technical knowledge who would be able to develop the result in the form of a complete, working backend by adding details to the form fields themselves.

The user will be asked to specify the following aspects to enable the system to generate database:
1. Database Name: The name of a database is basically its identity. Since it is the name by which the database would be referred to in the system, naming should be strictly done by following some naming conventions that help avoid conflicts or errors.
2. Database Description: Description fields contain a short, user-friendly explanation of the purpose of the database. This is not necessary for functional purposes in the backend application but lets all the users easily identify and manage their databases as they might be handling multiple projects.
3. Database Fields: Fields basically decide the structure and the nature of the data that would be saved in the database. For every field inserted by the user, he/she would have to enter the following details:
i) Field Name: Name of the attribute, which is the key name to keep particular values in the database.
ii)Field Type: The data type of the field lets it stipulate that it should store the variable as String, Integer, Boolean, and other data types. In other words, it restricts the field to store only a particular type of data.
iii) Required: A Boolean attribute that represents whether this field is required. When set to "required", a user is forced to provide a value for this field in creating or updating a database record. It is very important for data integrity-this will make sure something imperative will always be there.
iv) Unique: This field is marked as unique and must hold a value unique amongst the other entries of its own kind. For instance, a field in a user database is marked unique, say an email. No two records in that database are allowed to have the same email. Thus, the unique property keeps key data unique and prevents duplication of entries.



Database Name: Contact_Management

Database Description: This is a database for the contact management system website.

Name	Type	Required	Unique
Name	String	true	false
Phone	Number	true	true

Figure 3. Snapshot of Database Creation

3.2.2 Importance of Schema Design

A well-designed schema is one of the most important aspects of data consistency, integrity, and performance in querying. Traditionally, a schema would have been designed for the most part with heavy technical knowledge and deep understandings, with a great deal of know-how concerning the languages of databases, such as SQL or NoSQL.
At this juncture, the zero-coding methodology will ensure that the schema is visually defined through a GUI-based interface. Instead of delving deep into writing verbose code, a user can create his database structure by filling out forms representing the same form structure to the underlying schema. This gives the power to users of a GUI to generate a robust schema without understanding the intricacies of backend development. Those user-defined fields are the name, type, required, and unique, which constitute basic units to create a well-organized and effective schema.
Given the schema's provision through a zero-code interface, what is guaranteed for the user in designing a database is that it will:

1. Consistent: The data integrity is forced by the required and unique constraints.
2.Optimized: A well-structured schema ensures the database is optimized for performance and scalability.

3.Customizable: The flexibility in defining custom fields allows the user to mold up the structure of the database in view of particular requirements.

3.3 Data Validation

Data validation is an important aspect in development in that it checks information input by the user against certain set standards and constraints. Basically, data validation is quite crucial in ensuring that the database remains intact and functional; otherwise, a lot of errors could arise that ought not to have been there in the first place. Data validation is quite essential since it ensures that data accuracy and uniformity are attained by proving the information exists in a certain format, where it respects the constraints that have been defined. This kind of validation not only maintains the quality of the data stored but also plays a very significant role in ensuring the security of the database against threats. By preventing the introduction of malicious inputs, such as those that could result in injection attacks or other types of data manipulation, validation serves to protect the system against vulnerabilities. In addition, it makes sure that data compatibility matches the database schema in order to avoid problems during retrieval or manipulation of data and likewise that all fields should be well-defined and structured. Second, good data validation enhances user experience because it provides immediate feedback and suggests how users should fix their mistakes before the actual database is created. This way, the users can avoid certain potential problems, and their database functions well right from the beginning.

During the creation of a database on this platform, several checks are performed to ensure that valid input data cannot yield any type of opportunity for potential problems.

1. Special Characters: The system verifies that the database name or fields do not contain special characters which could conflict with operations of either the database or any queries. Special characters may raise an issue in the database system regarding syntax error or unwanted behavior.
2. Escape Sequences: This also checks that the name and fields do not contain any escape sequences-smashing up queries or another backend process to get executed. There are ways an attacker can use escape sequences in a malicious manner to inject harmful code into the database system.

3.4 Background Processing

After the user submits the schema via the interface, several key backend processes are executed by the system to create and configure the database accordingly, as per the settings of the user. It first creates a folder in the backend directory structure with the name of the unique ID of the user. This folder serves as a dedicated space for storing the user's database models and related files. A file is then created which contains the code representation of the schema supplied by the user. This file is named based on the name given for the database, with a timestamp appended to its name so there can be several models of the same name and avoid any file naming conflict.

Mongoose-used in this system-is an Object Data Modeling library for MongoDB and Node.js-to translate the schema definition provided by the user into a MongoDB-compatible format. It provides a way for developers to specify a schema for their data, gives them data validation out of the box, and even allows them to define models that can be used in queries and interaction with the database. In the background, the system actually converts the user's schema into Mongoose schema format, with each field defined against specific properties such as type, required, and uniqueness.

It generates the JavaScript file that defines the Mongoose schema. Based on the defined input schema, it creates a string of properties the system needs to define with their types, if required or not, and unique or not. This string is used to create the content of the JavaScript file to include the required code in Mongoose inorder to define and export the model. It uses Node.js's file system (fs) module to write this content to a file in the user's folder. In this way, it's guaranteed that the schema will be translated into code, and this file will be stored for later use correctly. This is the background processing that provides automation to translate user-defined schema into functional code in order to speed up the process for creating a database and efficiently manage their database models without having to interact with the underlying backend code. In this respect, the model is created and the user-specific schema of the database is set up in the MongoDB cluster of the website.

3.5 API Generation

APIs have been responsible for the process of information exchange and interaction between either parts or other ends of a software system. The functionality of APIs relies on the following CRUD operations: Create, Read, Update, Delete.

Basically, APIs are a set of rules and protocols in developing and interacting with software applications. They can also stand in the middle of a web application, frontend, and backend, which enables them to pass data and requests from one to the other. With APIs, users can perform an action on the data at the database side by just sending HTTP requests to specified endpoints. These include the following:

1. Create: This is an operation that enables the user to add new records into the database. In API terminology, it is generally carried out through a POST request, which involves sending new

information to the server for insertion into the database.
2. Read: This operation provides the facility for users to retrieve information from the database. It reads data through a GET request, which may fetch all records or particular records matching certain criteria.
3. Update: This operation gives rights to the user for editing of already existing records in the database. The PUT or PATCH request is sent along with the updated data to update the records at the server.
4. Delete: This enables a user to remove any records from the database. A DELETE request will be provided to identify what must be deleted based on an identifier, which could be an ID.

3.5.1 Defining the Endpoint Structure

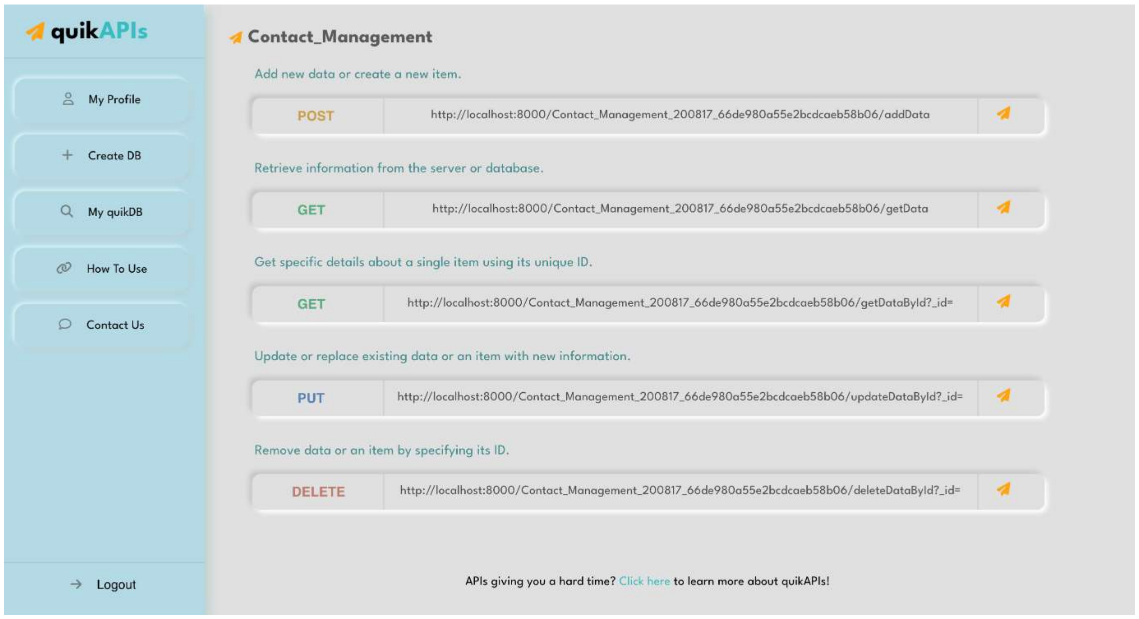
After designing and saving the database schema, the next thing is generating APIs that can interact with the database. The API endpoint structure is built to support CRUD activities on the newly created database model. API endpoints follow a standard pattern based on the model's name and are constructed as below:

- POST /addData: Adds new records to the database.
- GET /getData: Retrieves all records from the database.
- GET /getDataById?_id=: Retrieves a specific record by its unique ID.
- PUT /updateDataById?_id=: Updates a specific record by its unique ID.
- DELETE /deleteDataById?_id=: Deletes a specific record by its unique ID.

The API for each operation follows a distinct format, structured around the database and the backend hosting domain. The API endpoints are constructed as:

[BackendDomain]/[DBName]_[Timestamp]_[UniqueNonce]/[Endpoint]

This structure ensures that each API call is uniquely tied to its corresponding database instance.
By incorporating a timestamp and a unique nonce, the format guarantees security, organization, and traceability of requests, allowing seamless interaction with the backend.



Contact_Management

Add new data or create a new item.

POST http://localhost:8000/Contact_Management_200817_66de980a55e2bcdcaeb58b06/addData

Retrieve information from the server or database.

GET http://localhost:8000/Contact_Management_200817_66de980a55e2bcdcaeb58b06/getData

Get specific details about a single item using its unique ID.

GET http://localhost:8000/Contact_Management_200817_66de980a55e2bcdcaeb58b06/getDataById?_id=

Update or replace existing data or an item with new information.

PUT http://localhost:8000/Contact_Management_200817_66de980a55e2bcdcaeb58b06/updateDataById?_id=

Remove data or an item by specifying its ID.

DELETE http://localhost:8000/Contact_Management_200817_66de980a55e2bcdcaeb58b06/deleteDataById?_id=

APIs giving you a hard time? Click here to learn more about quikAPIs!

Figure 4. Sample API endpoints for CRUD operations based on user-defined schema

Normally, POST is used for forms that need to send data to the server for processing. This generally means new records are created, or the data submitted needs to be processed in some form. For example, a user is supposed to fill out the registration form and send it to the server in order to create his account.
The GET retrieves data from the server without modifying it. This gets information from a specific resource. Suppose a user wants to access his profile or search for certain data; through this, it sends a request to the server through GET. This retrieve operation presents the data without any change in it.
PUT updates the existing data present at a particular resource. If there is any modification or replacement of data, then the modified data is transferred to the server through a PUT request. Suppose a user wants to update his profile information; the PUT request is done to the server, which removes the previous details and replaces them with the new one provided by the user.

DELETE is used to identify the deletion of a particular resource from the server. This method will be used in cases when there is a need to delete some data. Suppose a user wants to remove his account or wants to remove an item that comes in his list. In both cases, the client sends a DELETE request to the server for deleting a certain resource in the database.

3.5.2 Defining Routes and Controllers

Routes define the path or URLs through which clients will interact with a web application. They usually map to specific handlers, which are defined in controller files. Each route defines a pattern for a URL and an HTTP method-pattern, such as GET, POST, PUT, and DELETE, which will define the way requests will be processed. By defining routes, the developers are offering the endpoints where users can operate and manipulate resources within the application.

Controllers contain the logic to handle such requests. They will process the incoming data, execute database queries or updates, and create responses to send back to the client. Controllers act as an intermediary between the route definitions and the data layer of a system and ensure proper execution of business logic. In general, a controller function is responsible for serving one route or another and for one HTTP method serving one of the four possible major actions depending on what the incoming request was. Routes deal with mapping API endpoints to controller functions. A route defines what happens with requests to specific endpoints. The system's routes file leverages the Express.js library in order to configure routing in support of CRUD operations:

- router.post('/*/addData', middlewareFileName, addData)
- router.get('*/getData', middlewareFileName, getData)
- router.get('*/getDataById', middlewareFileName, getDataById)
- router.put('*/updateDataById', middlewareFileName, updateDataById)
- router.delete('* /deleteDataById', middlewareFileName, deleteDataById)

Universal controller functions handle CRUD operations dynamically for any model. Function operations are as below:

1. 'addData' function: This is used to add records into the database. It extracts model and schema based on the URL, then creates a new instance of model with the data provided in the request body. Then, the new record is saved into the database.
2. 'getData' function: This would retrieve all records in the database related to a particular model by querying the database for data and returning it so that all entries could be viewed by clients.
3. 'getDataById' function: This retrieves one record depending on its ID. It queries the database for a particular ID specified and returns the found record, which a client could access.
4. 'updateDataById' function: Updates an already existing record stored in the database. This function finds the record by its ID and makes the updates included in the body of the request so that the record gets the most recent changes.
5. 'deleteDataById' function: This function removes, by ID, a record from the database. It deletes that record and thus enables data management by removing entries that are not needed anymore.

The project utilizes a universal controller that dynamically manages CRUD operations based on the URL parameters and user-specific information. Instead of creating separate controllers for each model, the controller fetches the relevant model and schema details directly from the request. It extracts the model name and user ID from the URL and uses this information to dynamically load the appropriate model and its schema.

This approach allows the system to handle various models without hardcoding specific details, making the application more scalable and easier to extend. For instance, as new models are introduced or new features are added, the controller can adapt without requiring significant changes to the existing codebase. By dynamically managing operations based on user tokens and URL parameters, the universal controller improves efficiency and simplifies the process of adding new models or endpoints to the system.

This design provides flexibility, ensuring smooth management of user data while keeping the codebase clean and adaptable for future extensions.

3.6 Using APIs

Once the APIs are generated, it would be considerably easier for a user to integrate them into their application. During this integration, users can perform all intended data operations, which include creating, reading, updating, or deleting records. It is achieved by embedding the provided API endpoints into the code of an application that enables interaction with the backend specified through CRUD functionalities.

After integration, one is supposed to test that the API actually works as it should. This means sending some requests across the API endpoints and checking the responses for correctness with regard to the

expected results. Testing forms an instrumental stage which will help in declaring whether the APIs are working as they're supposed to, and identify issues which probably need fixing.
Besides integration and testing, the website also allows the user to monitor and manage the data. Tools to view and interact with the data is provided on the website so that users can assure themselves of proper handling of the data and thus the overall integrity of the data. This total monitoring capability supports effective management of data and API interactions that make the application robust and reliable.

3.7 Data Encryption and Security

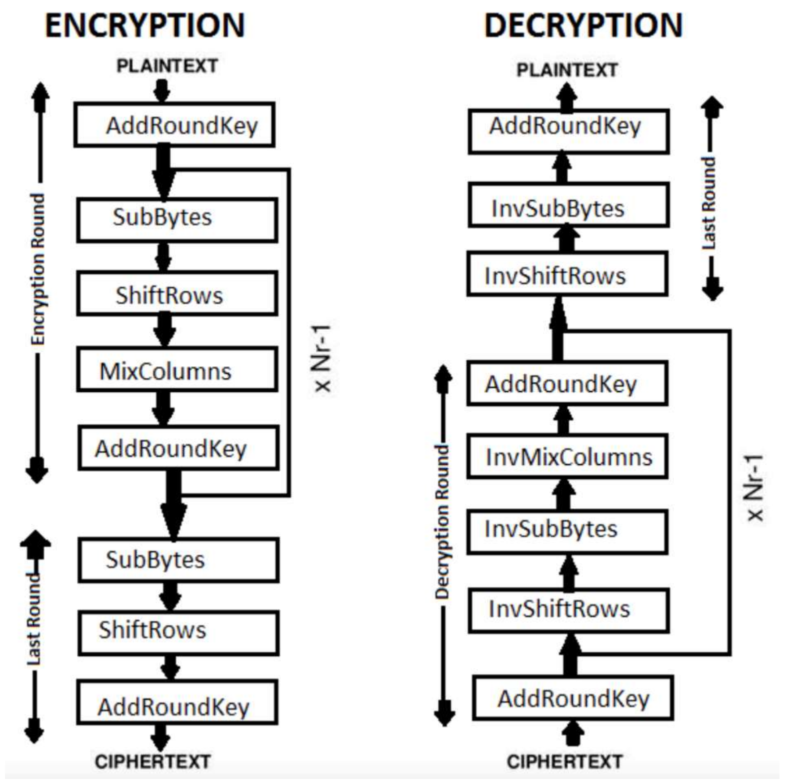


Figure 5. AES encryption and decryption process

It has become very crucial in today's digital era to safeguard the user data. The system is fully designed, keeping in view all the security measures for safeguarding all data stored by the users and keeping it private and secure. It ensures that comprehensive techniques of encryption are applied so that no user's data gets shared with any other person than the user themselves.
It is a basic security measure whereby sensitive information is transformed into an unreadable form to unauthorized persons. Only the relevant decryption key can decrypt the encrypted data and be readable. This is a way of safeguarding such data against any form of attack, like hacking or breach of data, whereby user information would otherwise be compromised.
The encryption utilized for this project is the Advanced Encryption Standard, which ensures that users' data enjoys the most paramount level of security. AES is a symmetric encryption process that is widely approved to be dependable and effective in securing data. It forms the standard of encryption applied by various governments, organizations, and financial institutions globally because of its solid security features.
How the AES Algorithm Works:

1. Key Generation: AES is a symmetric-key block cipher; the same key is used for both encryption and decryption processes. In this system, the encryption key is obtained from user-specific information-a password private to the user and token. Each user will have his own encryption key, making it unique. With it, we include elements that might be unique to the user, such as password or other secret information, thus generating a key unique to each user and further securing their data.
2. Encryption Process: After key generation, the generated key will be used for encryption of user data. The plaintext data gets converted into a ciphertext that is in a scrambled and unreadable format. AES operates multiple rounds of encryption operations, substitution, and permutation in nature, on the underlying data to be secure. This guarantees that any unauthorized access to the

ciphertext will not result in the extraction of data unless the correct decryption key is used.

3. Decryption Process: The encrypted cipher text gets decrypted using the same key whenever the authorized users need to access their data. Therefore, the decryption process will include the reverse of encryption operations applied to convert the ciphertext back into readable plaintext. These measures enable users to work with their data in a secured manner.

JavaScript can achieve AES by using the 'crypto' package. This will give a very easy and effective way to operate with encryption and decryption of AES. Crypto is a built-in package in Node.js. It can do many cryptographic jobs.

By using AES encryption, it ensures that all of the user data remain confidential and out of reach for unauthorized use. It introduces an additional layer of security whereby integration of user-specific information during the generation of keys includes that the encryption key has uniquely been generated for every user; this approach ensures well-protected integrity of sensitive information.

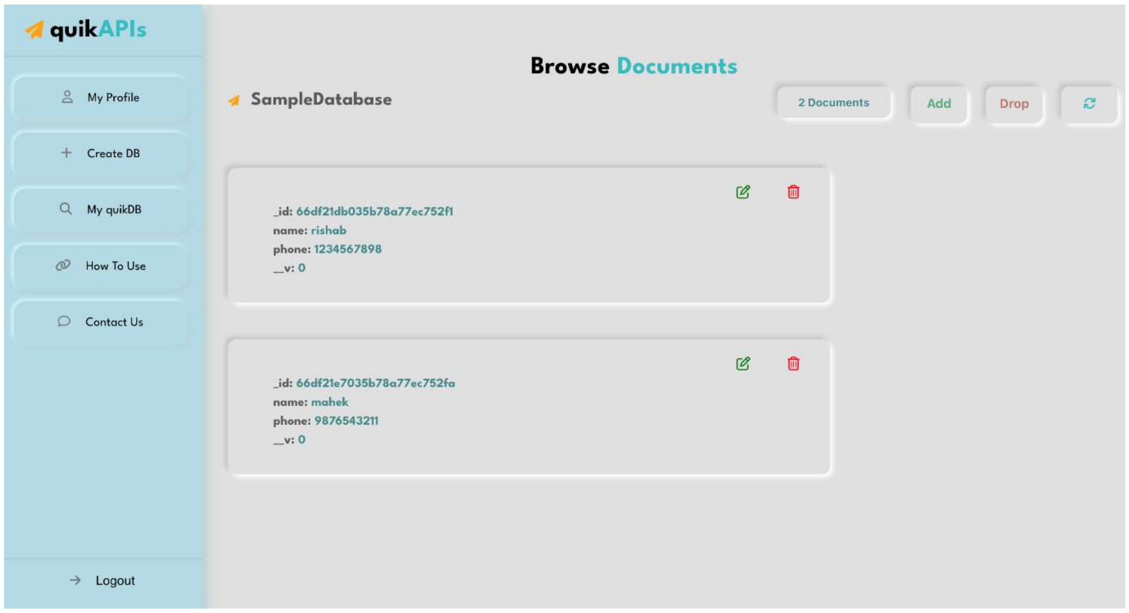
3.8 Database Management

The GUI of the site provides a powerful channel towards the manipulation and interaction with user-generated databases, enhancing their user experience through intuitive visual tools and controls. The single intuitive GUI-based dashboard will let users monitor all created databases. This overview enables users to track multiple databases and showcases the database assets clearly and organized.

3.8.1 Data Interaction and Entry

The user can view their collections for easy access and viewing of single records. The GUI allows for editing of already existing records, creation of new entries, and deletion of records. For extended use of data management, the user is also allowed to drop whole databases if need be; this will enable them to clean up their data structures or change the structure of data presentation.

The interface will provide a GUI form for adding new data, consistent with the schema specified in the creation of the database. This ensures that data entry is done with defined schema constraints in the form of required fields and uniqueness constraints. It has form validation mechanisms that disallow users from sending incomplete or bad data; hence, it will maintain stored information integrity.



_id: 66df21db035b78a77ec752f1
name: rishab
phone: 1234567898
__v: 0

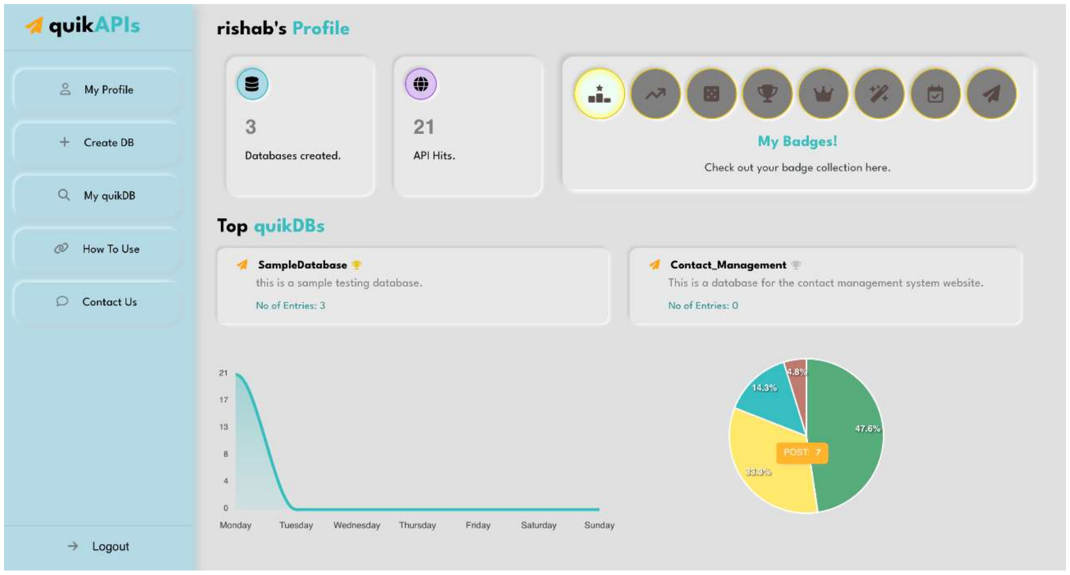
_id: 66df21e7035b78a77ec752fa
name: mahek
phone: 9876543211
__v: 0

Figure 6. Snapshot of the database

3.8.2 Interactive Dashboard

The project encompasses an interactive dashboard for illustrating API usage and performance in real-time. These dashboards give views of the total number of hits, the frequency of hits by different kinds of HTTP methods, such as GET, POST, PUT, and DELETE, among other statistics over the usage of APIs. Visualization of API activities allow the observation of the users interacting with their APIs, identification of the usage patterns, and informed decisions about how best they should use their APIs.

It follows therefore that, altogether, the management of the database through the GUI on the website will present an integrated and user-oriented approach to the maintenance of databases, allowing users to handle their data and APIs while retaining all high levels of control and visibility over their operations and presentation.



3
Databases created.

21
API Hits.

Check out your badge collection here.

SampleDatabase
this is a sample testing database.
No of Entries: 3

Contact_Management
This is a database for the contact management system website.
No of Entries: 0

Figure 7. Dashboard Insight

4. System Algorithm

```
Algorithm 1 Schema Creation and API Generation
Require: User-submitted schema data
Ensure: APIs for CRUD operations are generated
Step 1: Create User Folder
Create a directory for the user based on their unique ID
if Directory does not exist then
    Create directory
end if
Step 2: Generate Schema Code
Convert user-provided schema into Mongoose schema format
Format schema into JavaScript code
Save the schema code to a file with a unique timestamp
Step 3: Create APIs
Generate base URL for API endpoints
Define endpoints for CRUD operations:
/addData (POST)
/getData (GET)
/getDataById?_id= (GET)
/updateDataById?_id= (PUT)
/deleteDataById?_id= (DELETE)
Step 4: Set Up Routing
Configure routes for CRUD operations in the routing file
Map routes to appropriate controller functions
Step 5: Implement Controllers
Define controller functions for each API endpoint
addData: Add new data to the database
getData: Retrieve all data
getDataById: Retrieve data by ID
updateDataById: Update data by ID
deleteDataById: Delete data by ID
Step 6: Test and Monitor APIs
Test APIs to ensure they function as expected
Monitor API usage and data updates
Implement error handling and validation
Step 7: Data Privacy and Security
Encrypt user data using an encryption algorithm
Ensure data privacy and secure storage
Use user-specific information for key generation
```

This section describes the basics of algorithms and program codes that are the core of the automated database management and API generation system. The dynamically generated database schema and CRUD APIs are based on the inputs provided. These dynamically generated APIs were structured to create and handle them, and to integrate with the database. The algorithms described below detail the procedure of schema creation, API endpoint generation, and data processing. Included are code listings that show, in practice, how these algorithms are implemented, including the backend logic along with the routing mechanisms employed in the system. This algorithm will explain the technical framework and operational processes involved.

5. Conclusion

Because the world is going increasingly automated, many complex processes are becoming easier to use, and these tools help augment that productivity and accessibility. The creation of this platform was a huge step toward reducing the technical barrier for users who were not backend-savvy; these users could define the schema of their database and hence obtain an API from that automatically. It does look much more efficient to any user for having a system where data management and API generation are dealt with much more easily for quick integration in web applications.
The project focuses on the ease of use of tools in modern web development. It provides users with the ability to create, track, and manage databases using GUI; this makes sure even the non-expert user benefits from automated backend management. Further, it also keeps the focus on data privacy and security, and even encryption-a production-ready application of the platform would require protection of sensitive information.
This could potentially bring down development time, improve workflow efficiency, and enable a broader class of users to interact with backends with relatively less technical expertise. Tools of this nature form a great deal of the demand in the present scenario, wherein automation, scalability, and security form an integral part of any software ecosystem.

6. Future Scope

The project has the potential for further enhancement, expanding its functionality to meet evolving user needs. One key area of future development is the incorporation of relationships between databases, such as using primary keys from one database as foreign keys in another. This feature would facilitate more complex data models, allowing users to create interconnected databases. For instance, a customer database could be linked with an orders database by using the customer ID as a foreign key in the orders collection. Such relationships would improve data integrity and streamline data retrieval across related databases.
Another promising direction for future work is the integration of machine learning capabilities into the system. By analyzing the types of databases and schemas users create, the system could provide intelligent suggestions for fields and features tailored to specific use cases. This would allow users to benefit from data-driven insights, improving the design of their databases and APIs with minimal manual effort. Incorporating machine learning models could optimize the creation process, making the system even more user-friendly and adaptive to different requirements.
These potential advancements will further enhance the tool's utility, making it more robust, flexible, and aligned with modern data management and automation trends.

REFERENCES

[1] Yan, Z.: The impacts of low/no-code development on digital transformation and software development. Journal Name 1(1), 1–10 (2024)

[2] Author(s): Algorithms have evolved from machine code to low-code-no-code (lcnc) in 20 years. Algorithms 16(2), 108 (2023) https://doi.org/10.3390/a16020108

[3] Hubli, S.C., Jaiswal, D.R.C.: Efficient backend development with spring boot: A comprehensive overview. International Journal for Research in Applied Sciences and Engineering Technology (IJRASET) 11(1), 123–135

[4] Sipio, C.D., Ruscio, D.D., Nguyen, P.T.: Democratizing the development of recommender systems by means of low-code platforms. In: Proceedings of The FirstLowCode Workshop (LowCode 2020), p.9. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145

[5] Ramesh, R.K., P, M.D.: Revolutionizing software development: The rise of no code/low code development solutions in digital era. International Journal for Multidisciplinary Research (IJFMR) 6(2), 1–2 (2024)

[6] Woo, M.: The rise of no/low code software development—no experience needed? Engineering (Beijing) 6(9), 960–961 (2020) https://doi.org/10.1016/j.eng.2020.07.007

[7] Silva, J.X., Lopes, M., Avelino, G., Santos, P.: Low-code and no-code technologies adoption: A gray literature review. In: SBSI '23: Proceedings of the XIX Brazilian Symposium on Information Systems, pp. 388–395. https://doi.org/10.1145/3592813.3592929

[8] Alamin, M.A.A., Malakar, S., Uddin, G., Afroz, S., Haider, T.B., Iqbal, A.: An empirical study of developer discussions on low-code software development challenges. IEEE (2020).

[9] SK, M.M., AC, G., Basavaraj, Meghana: No-code web development. International Research Journal of Modernization in Engineering Technology and Science 5(6), 2959 (2023) https://doi.org/10.56726/IRJMETS42320

[10] Low-Code Development Platform Market Research Report – Global Industry Analysis, Trends and Growth Forecast to 2030. Accessed: 2022-11-27 (2021). https://www.researchandmarkets.com/reports/5184624/ low-code-development-platform-market-research

[11] Rokis, K., Kirikova, M.: Challenges of low-code/no-code software development: A literature review. In: Perspectives in Business Informatics Research, pp. 3–17. Springer (2022). https://doi.org/10.1007/978-3-031-16947-2 1

[12] Smith, J., Johnson, M., Anderson, L.: The rise of no-code development: Empowering Non-programmers in web development. In: Proceedings of the International Conference on Web Development (ICWD) (2022)

[13] Jones, A., Lee, S.: Comparative analysis of no-code development platforms for web Journal of Software Engineering Research 25(2), 145–160 (2020)

[14] Brown, C., Davis, R.: Exploring the theoretical foundations of no-code development: A conceptual framework. Journal of Software Engineering 42(4), 567–582 (2018)

[15] Rodriguez, M., Gomez, P.: No-code web development: A review of tools and technologies. Journal of Web Engineering 38(1), 89–104 (2019)

[16] Patel, R., Kumar, S.: A comparative study of no-code development platforms for rapid web application development. International Journal of Computer Applications 178(5), 12–18 (2020)

[17] Lee, H., Kim, S., Park, J.: Empowering end users through no-code web development tools: A user study. Journal of Human-Computer Interaction 37(2), 245–260 (2021)

[18] Nguyen, T., Nguyen, H., Tran, L.: Exploring the benefits and limitations of no-code web development: A case study. International Journal of Advanced Computer Science and Applications 10(3), 450–460 (2019)