iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 10 **Issue:** IV **Month of publication:** April 2022

DOI: https://doi.org/10.22214/ijraset.2022.41827

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue IV Apr 2022- Available at www.ijraset.com

A System on Automated Database and API (Application Programming Interface) Management

Riddham Gadia[1], Rushank Shah[2], Shashank Varshney[3], Vinaya Sawant[4]
[1, 2, 3, 4]*Department of Information Technology, Dwarkadas J. Sanghvi College of Engineering, Mumbai, India*

Abstract: ***Web development is a skill that is high in demand, but acquiring it requires a lot of hard work and experience. Managing the frontend is easy but managing the backend and the database is highly daunting. It requires lots of practice. Hence this makes web development complicated. So, a brief analysis has been done on the existing system like Amazon API Gateway, Firebase, and Google Apigee. Also, this paper shows a comparative study of various data transfer formats and techniques. This paper mainly focuses on a novel approach for automated database and API management that resolves the drawbacks in the existing systems. This system will ensure ease of use and a faster development process in terms of front-end and back-end connectivity. The new system focuses on building an architecture that is both scalable and secure from a backend and database perspective.***

Keywords: Cloud Computing, Firebase, Amazon API Gateway, Google Apigee, REST, SOAP, XML, JSON, Automated Database Management System.

I. INTRODUCTION

Backend is an integral part of all systems but developing it is not easy. A lot of things must be managed throughout, and this leads to a lot of errors along the way. The existing systems like Google Apigee, Amazon API Gateway, and Firebase try to make backend development easy but still we need to write a lot of amounts of code and all these platforms are language dependent. This paper discusses these technologies and the pros and cons of all these technologies. Along the way, it will also discuss different formats of data transfer namely, JavaScript Object Notation (JSON) and Extensible Markup Language (XML). This paper also briefly touches on the two famous types of patterns for data transfer, Simple Object Access Protocol (SOAP) and Representational State Transfer (REST). To overcome the drawbacks of the existing systems, the paper also proposes a system to automate the API and Database Management System.

II. RELATED WORKS

A. Google Apigee

Apigee [3] is a tool that is offered by Google Cloud. It is used to exchange data across cloud services and applications. It is an API Management tool. It makes it easier to produce and deploy modern apps. Apigee helps in creating and developing RESTful APIs and helps in managing them.

The features of Apigee are:

1) It automates the API development and documentation process.
2) It supports public cloud, hybrid cloud, and multi-cloud.
3) It offers monitoring tools for security, optimization, analytics, etc.
4) It can scale applications.

B. API gateway

Amazon API Gateway [1] is a service that is provided by AWS. It is used for securing, monitoring, publishing, maintaining, and creating REST and HTTP API. API developers can create APIs or use any existing service provided by AWS. Developers can also make use of AWS Cloud to access their data. The API created by Amazon API Gateway can be used in any application. We can also make the API available to third party app developers. API Gateway creates [1]

1) HTTP-based APIs that have Restful architecture
2) It enables client and server communication in a stateless manner.
3) It implements all the standard HTTP methods like PUT, POST, GET, PATCH, and DELETE

API Gateway acts as an intermediary between the client who initiates the request and the service that will the client is consuming. It is not the destination for the HTTP request. All the requests made by the client will be routed to the integration after passing through the authorization and validation. Integration can be an HTTP endpoint, or it can be a user written Lambda function. The integration can also be any other service provided by AWS.

C. Firebase

Firebase [2] is an app development platform that provides different tools and technologies to make tasks easier and automate the workflows for faster production of applications. One of the tools provided by Firebase is Firebase Cloud Firestore. It is a NoSQL database provided by Firebase which has prebuilt SDKs for different programming languages like JavaScript, Java, and Python. These SDKs help in communicating with the database and come with an amazing toolset for different Create, Read, Update and Delete operations. It is quite easy to use and is very reliable because the data is synced across all the clients communicating with the database. It has various advantages like Offline Synchronization, Serverless architecture, and the pay-as-you-go model. It works on the principles of ACID (Atomicity, Consistency, Integrity, and Durability) and terminates an entire transaction even if one of the operations within the transaction fails. It can automatically scale when the user's demands increase, and the base performance is quite consistent. It is available on Web, Android, and iOS applications and has native SDKs for all three platforms.

Using Firestore is very easy as you just need to set up the project and import SDKs and communicate with the database. But it is language restricted. It does not support all the languages. This is one of the biggest disadvantages of using Firestore. For example, as of now, you cannot use Firestore with Rust or Haskell directly with native SDKs. This restricts the users using these languages from using Firebase with them.

D. SOAP vs REST

SOAP [5][6] is a web-based architecture that has three main components: a service requester, service registry, and service provider. The service provider provides the services, that accept and executes requests from consumers, the service registry is a network-based directory that consists of the available services. The service provider publishes its service description in a service registry for the service requester to find, the service requester then finds the service description in the registry. A service requester is used to bind with the service provider. A service description is used for this purpose. The requester interacts with the web service implementation. The communication standard is based on XML or SOAP protocol. REST [5][6] stands for Representational State Transfer which is based on ClientServer Architecture Model where the client sends the request to the server and thus an appropriate response is sent back to the client. REST is designed on three principles, and they are Addressability, statelessness, and uniform interface. Addressability refers to the idea of where resources or data can be reached through a unique identifier or URI. Statelessness states that every HTTP request is unique and is independent and unrelated to the previous request. Uniform Interface refers to the set of standards or HTTP methods that are used to access the resources, i.e like GET, PUT, POST, DELETE, etc [12].

E. JSON vs XML

JSON stands for JavaScript Object Notation. It is used for sending and receiving data from the webserver. It is a JavaScript object. The data in JSON is represented in the form of key-value pairs.

Syntax of JSON: var obj= { key: value
};

XML [4] stands for Extensible Markup Language. It is also a data format for sending and receiving data to and from a server. It is like HTML where we use tags to different attributes.

Syntax of XML:

```
<student>
  <name>John</name>
  <age>21</age>
</student>
```

There are various similarities between JSON and XML as mentioned below:

1) JSON and XML are self-describing in nature as data is in a human readable format
2) They are primarily used as a data interchange format
3) They both can be easily parsed and hence used by different programming languages.
4) Both the formats can be retrieved by using HTTP requests methods like GET, POST, PUT, etc.
5) The following are the difference between JSON AND XML [13]:
6) JSON supports arrays whereas XML supports Name spaces
7) The XML format has much better security than JSON format.
8) XML uses tag structure whereas JSON uses JavaScript Objects to represent data.
9) XML is a markup language whereas JSON is a data format.

III.OUTCOME OF LITERATURE WORK

Table 1. Comparative study between Firebase, Amazon API Gateway, and Google Apigee

Parameters	Firebase	Amazon API Gateway	Google Apigee
Need to write code?	Yes	Yes	Yes
Language Dependent?	Yes	No	No
Easy to use?	No	No	No
Requires Previous Knowledge?	Yes	Yes	Yes
Developer Friendly?	No	No	No
Presence of Database?	Yes	Yes	Yes

Based on various parameters, this section compares Firebase, Amazon API gateway, and Google Apigee. The parameters are needed to Write Code, Language Dependency, Easiness, Previous Knowledge dependency, developer friendliness, and presence of a database. Based on the review made, Firebase seems to perform better than the other two services. Apart from the above-stated traits, Firebase provides many more features like Firebase Cloud Messaging and App testing.

Both JSON and XML have a variety of uses. But JSON is better than XML for the following reasons [13][19]:

1) JSON is faster because parsing large XML files takes a huge amount of time, resulting in network load.
2) JSON is much faster than XML when used in AJAX applications.
3) The data structure used in JSON is a map that is more predictable and easier to use than XML's tree data structure.
4) JSON is less verbosity than XML
5) JSON is preferred when we are using REST-based architecture in comparison to XML.

Web services are used widely over the internet, it has become an important factor for communication between applications. The REST-based services are better than the SOAP-based [14][15]. REST-based services provide lower network traffic, low latency, and lower message size as compared to the SOAP-based web services, thus indicating that the REST-based services provide much better performance than the SOAP-based services whether it is wired or wireless communication networks. The REST-based services are lightweight, and flexible with low overhead hence it is the most used web service in the current standards [14].

A. Drawbacks of the Existing Systems

1) Firebase

a) Complex query processing is very difficult in Firebase because it depends on a flat data hierarchy. Queries like reversing the order of certain items are not possible to execute in Firebase.
b) Firebases' focus is to share data between different applications and therefore have fewer security standards.
c) As Firestore uses a NoSQL-based database, implementing constraints is not possible on the database level and everything has to be managed at the code level.
d) Firebase has very high pricing.
e) Firebase has limited libraries for different programming languages.

2) Apigee

a) It is very difficult to debug JavaScript in the default console provided by Apigee. The console also does not include any console log and tracing. This results in wastage of error and debugging time.
b) Apigee is very costly and is not suitable for any small scale enterprise.
c) We cannot use JSON.stringify on Apigee Objects which makes it difficult to work with Apigee Objects.

3) Amazon API Gateway:

a) AWS API Gateway requires Lamba functions which require coding in languages like JavaScript or Python.

b) Architecting high-availability applications require a lot of experience in working with AWS since the API Gateway is the only point of entry for the frontend applications to communicate with the APIs. This makes using AWS less beginner friendly.

c) The configuration of API with AWS API Gateway requires a lot of manifestation processes, which adds an extra layer of difficulty for developers to build their applications with the platform.

Current solutions are trying to solve the problem of building and managing the database have certain drawbacks which are stated above. To solve these problems, a system can be made which makes backend coding much more developer-friendly. This will reduce the workload tremendously. In the proposed solution, a new system will be designed which will try to resolve the issues in the existing systems. This system will serve as a backend for all projects. Using the system, all users can store, retrieve, update, and delete the data through the APIs. The users will be provided with HTTP/HTTPS endpoints based on Rest API concepts. These endpoints can be used directly inside a web application or an app. By using routes users don't have to worry about the backend. They will just have to create the frontend and use the proposed system for the backend. The system will make web development user-friendly.

IV. PROPOSED SOLUTION

In this section, a system has been proposed that resolves the issues present in all the similar existing systems. The proposed system is divided into modules. Each module has a specific use and has been explained in this section. A comparison with all the existing systems showing all the demerits of them in comparison to the proposed system has also been done in this section.

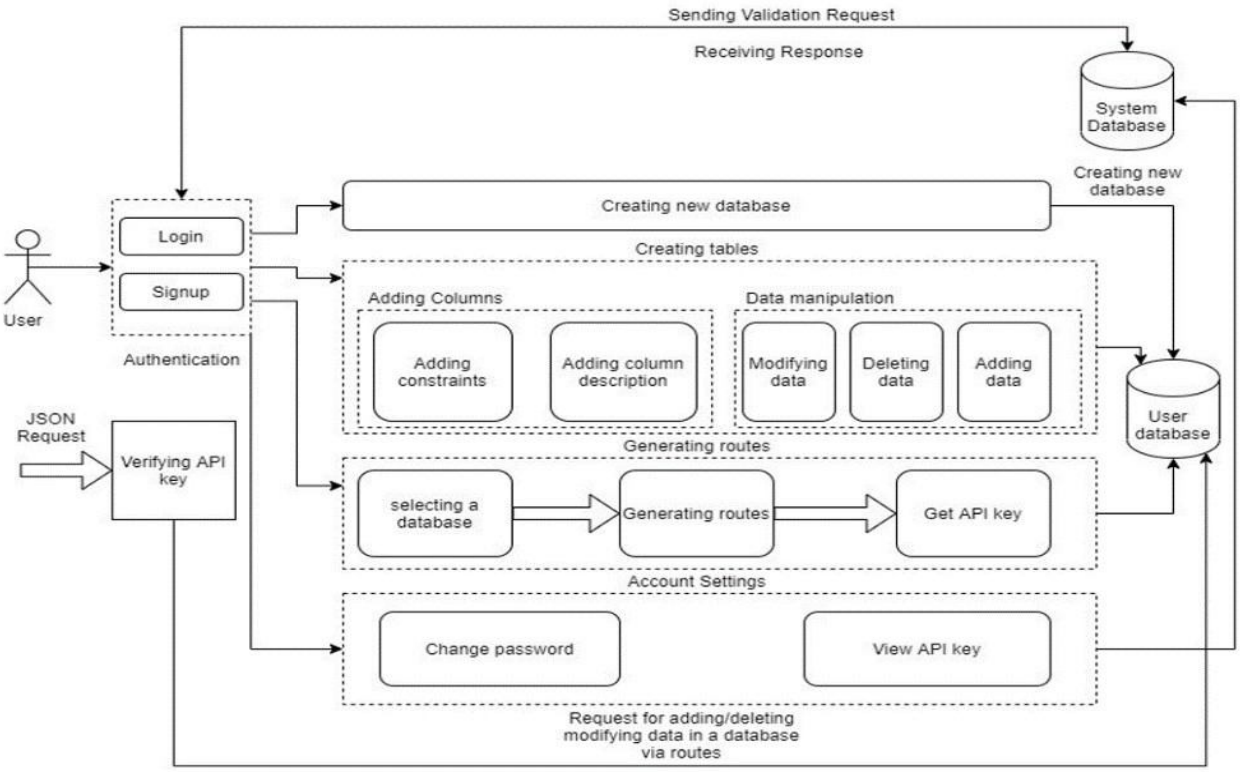


Fig. 1. System Architecture

Figure 1 shows a high-level system architecture of the proposed system. It depicts the life cycle of the user's interaction with the system. In a nutshell, the user creates a new account on the portal and receives an authentication key. Then, the user creates a new database and configures the database with row and column constraints. After this, the user can add rows to the database using the inbuilt GUI or by creating a new API. These APIs can be integrated directly with their frontend application. The user will also have the feature to change passwords and view API keys on their dashboard.

A. Authentication

This module will perform the authorization of the user to ensure that authentic users will access the system. Each user will have a username and password. If the username or password is wrong, the user won't be allowed to access the system. This ensures the security of the system. It consists of two options. If a new user is using the system, he or she can create an account using sign-up. If a user already has created the account, then he or she can login. The data of the users will be stored in a system database. We will be using a tokenization system to authenticate the API requests so that the system can identify requests made by a specific user and ensure the security of the system. The tokenization has been done using libraries like JWT (JSON Web Tokens) [16].

B. Creating New Database

This module is for creating a new database. Users can only access this module if the user is authenticated. Each user can create as many databases as he wants. For each database, various details will be asked if a user wants to create a new one. Multiple tables can be added under a particular database. For that, the user must access the creating tables module. For implementing the database, NoSQL will be used [17].

C. Creating Tables

After creating a new database, the user can create as many tables as he wants. This module will be responsible for making all the changes in the tables of the database. Users can perform all the CRUD (create, read, update, delete) operations in this module. In this module individual columns must be created. For each column, the user can select all the constraints. Based upon whether a particular entity is null or not appropriate description can be added in the column. This module will directly communicate with the user-created database.

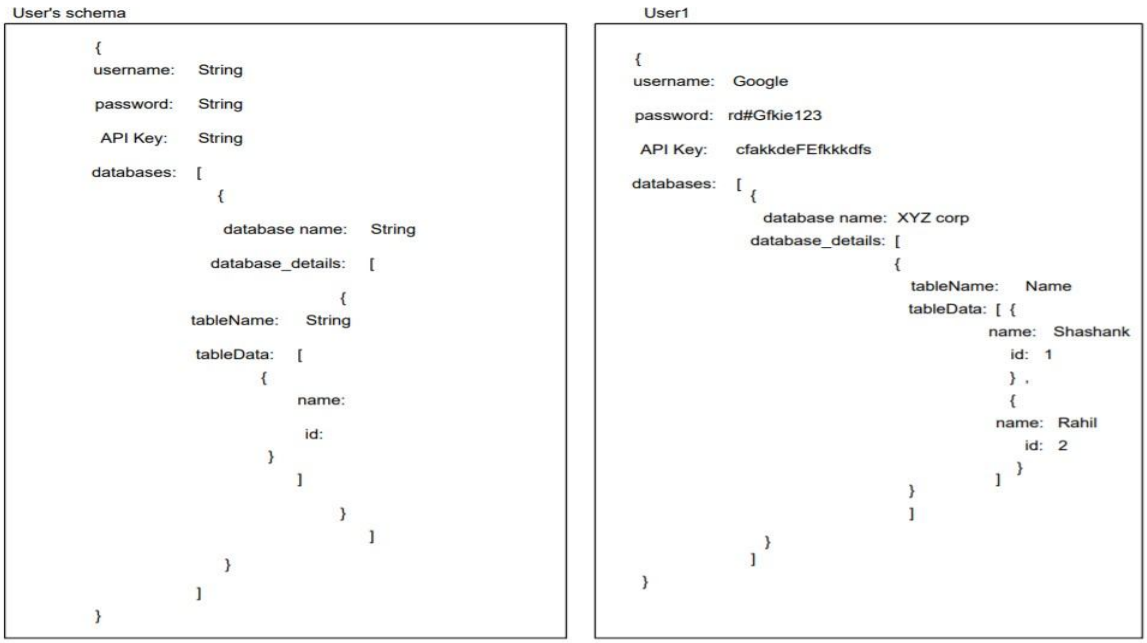


Fig. 2. Database Schema

Once the tables are created, a new record of the user is created in the main database as shown in Figure 1. Every user will have four common attributes, namely username, password, API Key, and databases. Once the user registers to the portal, these three attributes will be initially saved to the main database. Now, whenever a user creates a new database, the system will automatically append a new database to the databases attribute of the user. As and when the user creates tables for this database, the database appends a new table to the tables attribute of the database.

For example, a user creates a new account with the username as Google and password as abcd, his password will be hashed and stored in the database. Next, an autogenerated API key will be handed to the user which will be used to authenticate the user's requests. Now, let's say the user creates a database named 'XYZ Corp'. This XYZ Corp has a table named 'Name'. Then the user creates a new table data with attributes name and id. This is how the database will look when everything is stored in the database.

D. Generating Routes

This module will be responsible for generating HTTP endpoints that can be used in all applications. To generate routes, the user must be validated. After validation, the user must select a database. Upon selection of the database, the user will be given both the route and the API key. This API key will be used to ensure validation. It will increase the security of the system whenever third-party applications are using the system. A user must have an API key as a part of the request sent to the system. If the key is not present, the response won't be sent.

E. Account Settings

This module is used to update or add account details. The user can modify his password and view the API key.

1) *User Database:* This is the database that is created by the user. Each user can create as many databases as he wants. Based upon which user is accessing the system, an appropriate database will be selected.

F. Verifying API key

This module is used when an application tries to use the HTTP endpoints to perform the CRUD operations in the database. To ensure all trusted applications access the database, each user is given an API key. This API key will be a part of every request. This module will make sure that in each request appropriate API key is present. If the API key is not present or is incorrect, the request won't be directed further. This protects the system from malicious applications.

G. System Specification

The following system specifications are necessary to use the proposed system:

1) The user should have an active internet connection
2) A browser like Google Chrome or Mozilla Firefox is required
3) A reliable operating system is required for all the frameworks to run efficiently [18].

Table 2. Comparative analysis of proposed system and existing systems

Parameters	Firebase	Amazon API Gateway	Google Apigee	Proposed Solution
Need to write code?	Yes	Yes	Yes	No
Language Dependent?	Yes	No	No	No
Easy to use?	No	No	No	Yes
Requires Previous Knowledge?	Yes	Yes	Yes	No
Developer Friendly?	No	No	No	Yes
Presence of Database?	Yes	Yes	Yes	Yes

As mentioned in table 2, we can see that the proposed solution outperforms the existing solutions in many ways. For instance, Firebase requires you to use existing APIs for different programming languages. However, it lacks support for programming languages like Rust or Haskell. In contrast, the proposed solution tries to overcome this drawback by providing a single link-based API that the user can directly incorporate in their application. Moreover, AWS API Gateway and Google Apigee are convoluted systems. They require a lot of pre-processing before using them in the frontend applications. But the proposed solution is GUI-based and intuitive, hence making the user's process of building the backend easier and faster. Overall, the proposed system is easier to comprehend and use and does all the heavy lifting stuff by default, making it a better solution than the existing systems.

Features of the system:

1) No need to code to connect to a database
2) Transfer of data will take place with JSON
3) It can be used both in web and app
4) Users will get HTTP/HTTPS endpoint
5) User-friendly GUI
6) The entire backend will be provided
7) API keys will be provided to ensure the security of the system.

V. IMPLEMENTATION

In this section, the implementation of the project will be discussed. This section will show some wireframe layouts of the design of the web application and how a user will communicate with the portal. This section also emphasizes real world functionalities of the proposed system and how the user will interact with the system. For the implementation of the above system, React is used in the frontend, Express, and NodeJS in the backend. For the database, NoSQL will be used. A graphic user interface will be provided to the users.

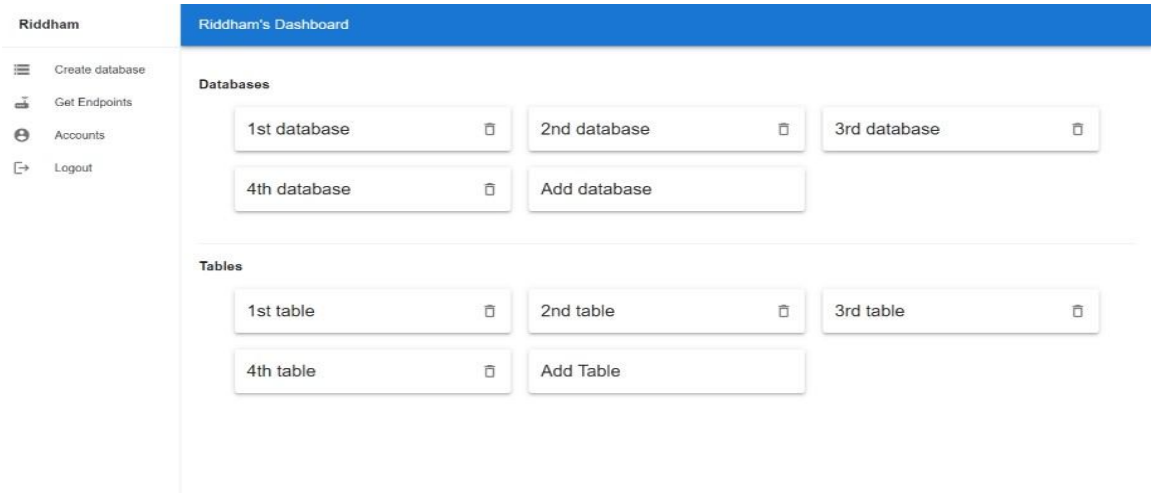


Fig. 3. Create Database Page

Figure 3 describes how a Database creation page will look. On this page, users can create a new Database and add tables and data to them. Users can create multiple databases. When a database is selected, the user will be able to view all the tables enlisted in that database. The user can manually add restrictions to each table and a popup will be displayed for showing the fields. Users can also delete a particular database or a table. The above figure will be a part of creating a database module.

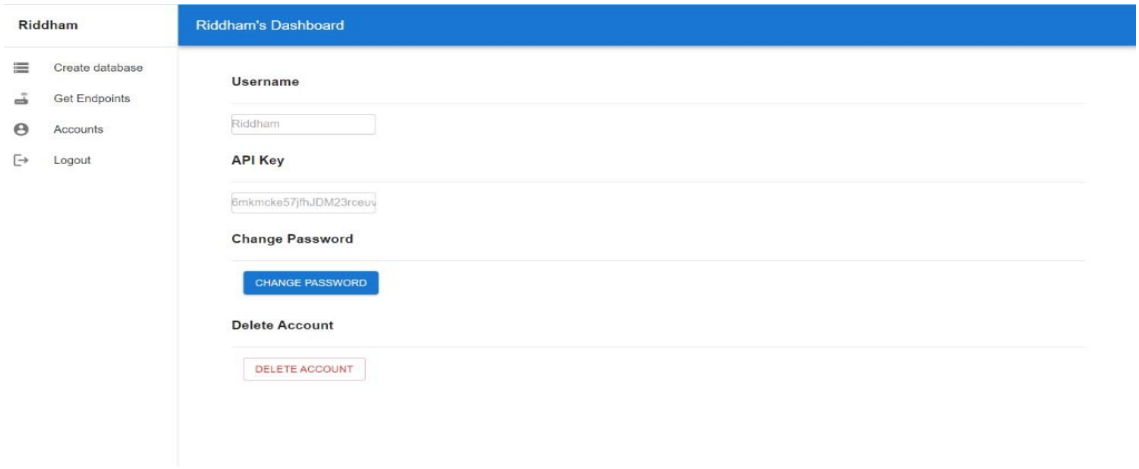


Fig. 4. Accounts Page

Figure 4 describes how a user will manage their account. They can get their API key and change password from this dashboard. Users will also have an option to delete their account if they wish to. The API key will be used in the module where we fetch the endpoints to create APIs for different CRUD options for their databases.

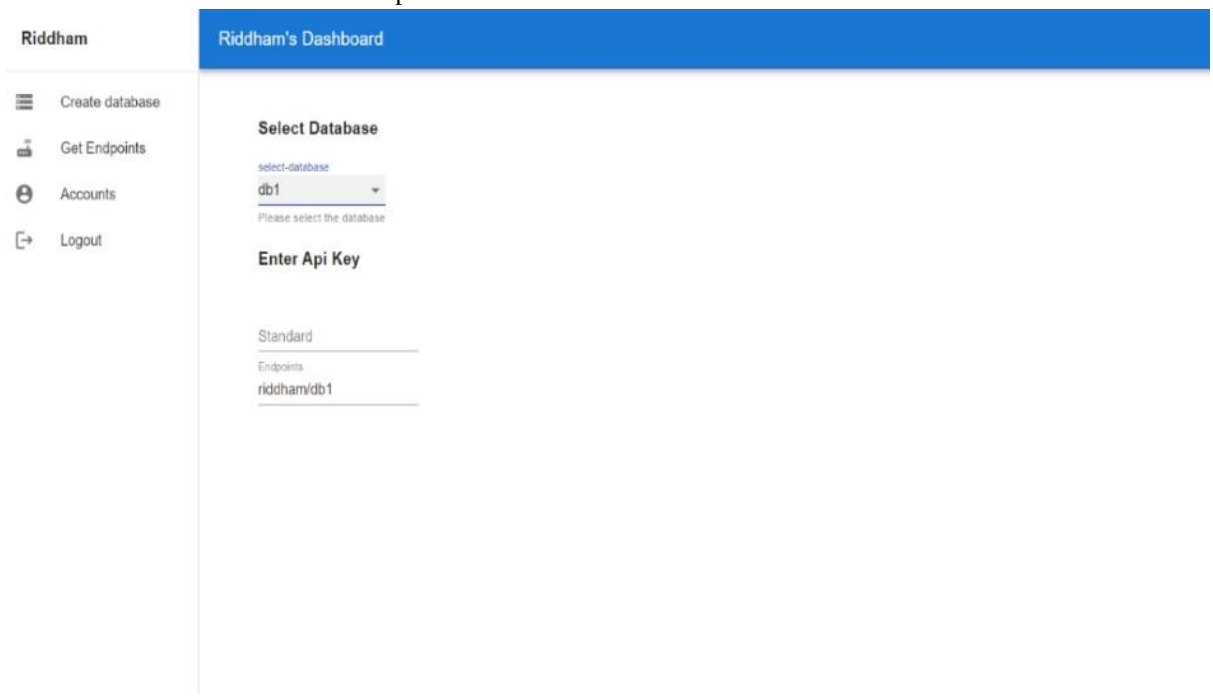


Fig. 5. Get Endpoints Page

Figure 5 describes the page where users can generate Endpoints. These endpoints will be used to fetch data from a database or add some data to the database. Users can select a database from the database dropdown and then configure the API according to the user's need. Once the configuration is added, the user can directly incorporate these APIs in their frontend applications.

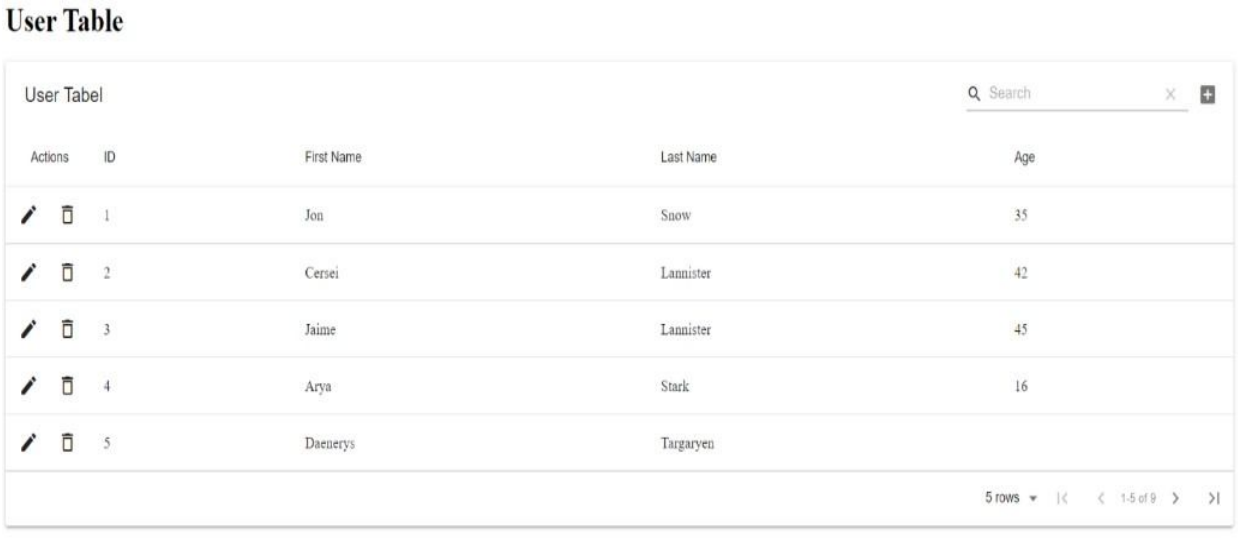


Fig. 6. Database Table

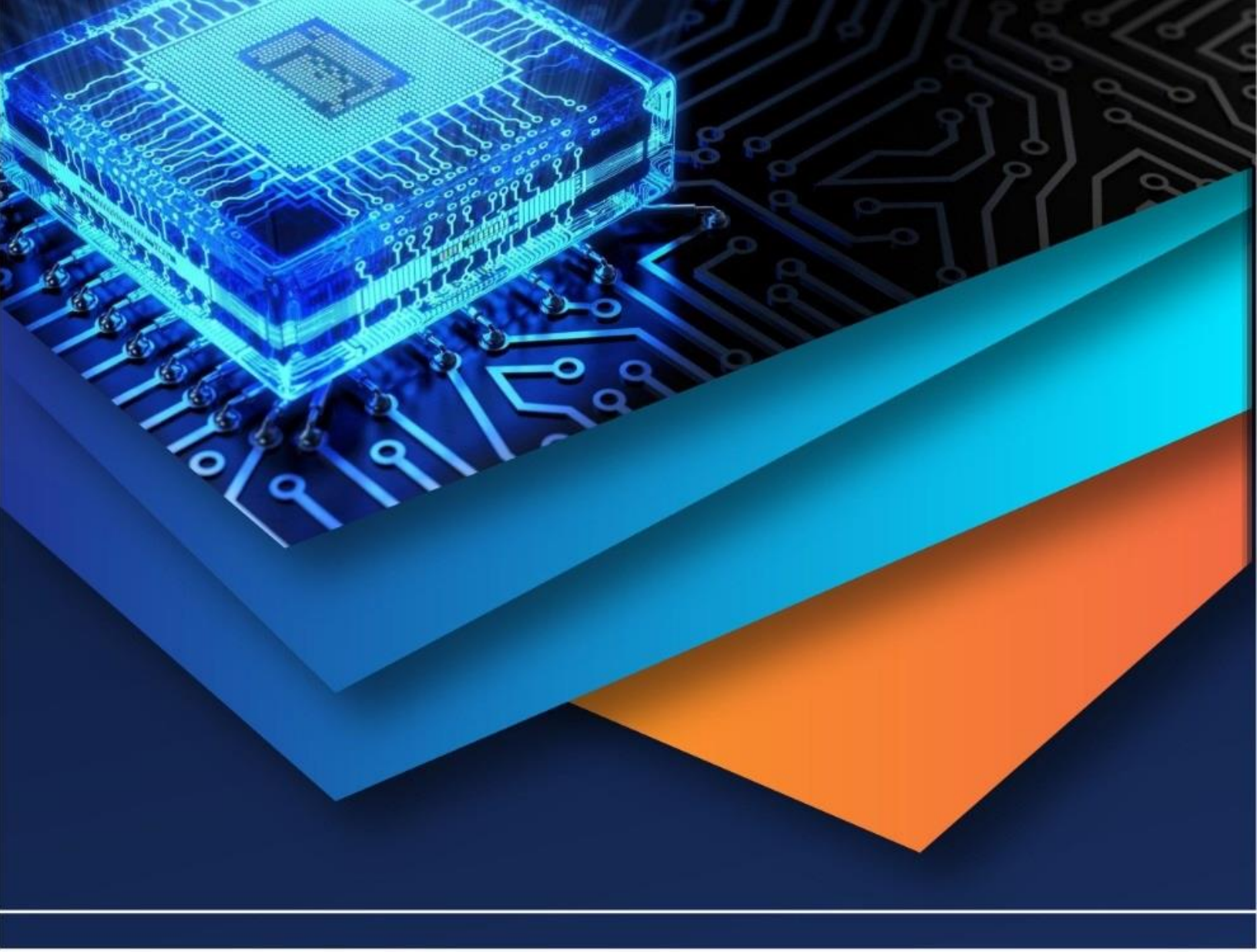
Figure 6 describes how a user can perform operations on the table. Users can add, delete, and edit multiple rows. Each row will have an ID associated with it which will act as a primary key. Depending upon the number of rows, the user can select 5,10, 50, or 100 rows to be displayed on the web page. Users can sort data based on any column. Users can also search for a particular row in the search bar.

VI. CONCLUSIONS

This paper has reviewed the different types of solutions for enhancing and making the development and managing the database in an easier and developer-friendly way. Amazon API Gateway, Firebase, and Google Apigee, all three try to solve this problem using different techniques and methodologies but the end goal is to build a solution for better database and backend management. So, this paper proposes a system that overcomes the drawbacks of the existing systems. The system is built using REST API and JSON as the data transfer technique over SOAP and XML. Using the system user doesn't have to worry about connecting the backend with the database. Users will be provided with HTTP/HTTPS routes that can be used in any application. In conclusion, using the proposed system developers can manage the backend with ease.

REFERENCES

[1] What is Amazon API Gateway? - Amazon API Gateway [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html, last accessed 14-10-21
[2] Documentation | Firebase [Online]. Available: https://firebase.google.com/docs, last accessed 13-10-21
[3] Apigee documentation | Apigee X | Google Cloud [Online]. Available: https://cloud.google.com/apigee/docs, last accessed 12-10-21
[4] JSON vs XML [Online] Available: https://www.w3schools.com/js/js_json_xml.asp, last accessed 14-10-21
[5] Mumbaikar, Snehal, and Puja Padiya. "Web services based on soap and rest principles." International Journal of Scientific and Research Publications 3, no. 5 (2013): 1-4.
[6] Halili, Festim, and Erenis Ramadani. "Web services: a comparison of soap and rest services." Modern Applied Science 12, no. 3 (2018): 175.
[7] Wagh, Kishor, and Ravindra Thool. "A comparative study of soap vs rest web services provisioning techniques for mobile host." Journal of Information Engineering and Applications 2, no. 5 (2012): 12-16.
[8] Castillo, Pedro A., Jose Luis Bernier, Maribel Garcia Arenas, J. J. Merelo, and Pablo GarciaSanchez. "SOAP vs REST: Comparing a master-slave GA implementation." arXiv preprint arXiv:1105.4978 (2011).
[9] Tihomirovs, Juris, and Jānis Grabis. "Comparison of soap and rest-based web services using software evaluation metrics." Information Technology & Management Science (Sciendo) 19, no. 1 (2016).
[10] Su Myeon Kim, Marcel-Catalin Rosu "A Survey of Public Web Services" WWW 2004, May 17–22, 2004, New York, New York, USA
[11] Hatem Hamad, Motaz Saad, and Ramzi Abed "Performance Evaluation of RESTful Web Services for Mobile Devices" Computer Engineering Department, Islamic University of Gaza, Palestine International Arab Journal of e-Technology, Vol. 1, No. 3, January 2010.
[12] Fatna Belqasmi, Jagdeep Singh, Suhib Younis Bani Melhem, and Roch H. Glitho Concordia "SOAP-Based vs. RESTful Web Services A Case Study for Multimedia Conferencing" University Published by the IEEE Computer Society 1089-7801/12/$31.00 © 2012 IEEE.
[13] Nurseitov, Nurzhan, Michael Paulson, Randall Reynolds, and Clemente Izurieta. "Comparison of JSON and XML data interchange formats: a case study." Caine 9 (2009): 157-162.
[14] Zunke, Saurabh, and Veronica D'Souza. "Json vs xml: A comparative performance analysis of data exchange formats." IJCSN International Journal of Computer Science and Network 3, no. 4 (2014): 257-261.
[15] Simec, Alen, and Magdalena Maglicic. "Comparison of JSON and XML data formats." In Central European Conference on Information and Intelligent Systems, p. 272. Faculty of Organization and Informatics Varazdin, 2014.
[16] JSON Web Tokens - jwt.io [Online]. Available: https://jwt.io/, last accessed 14-10-2021
[17] What is NoSQL? NoSQL Databases Explained | MongoDB [Online]. Available: https://www.mongodb.com/nosql-explained, last accessed 14-10-2021
[18] A. S. Tanenbaum, J. N. Herder and H. Bos, "Can we make operating systems reliable and secure?," in Computer, vol. 39, no. 5, pp. 44-51, May 2006, doi: 10.1109/MC.2006.156.
[19] B. Lin, Y. Chen, X. Chen and Y. Yu, "Comparison between JSON and XML in Applications Based on AJAX," 2012 International Conference on Computer Science and Service System, 2012, pp. 1174-1177, doi: 10.1109/CSSS.2012.297.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY