Introduction

This document provides an overview of an application that creates a dynamic backend using a form generator. The application returns a configuration JSON file and creates a database for the schema present in the configuration file. The schema format is {"form_name":"f","db_name":"db","form_fields":{"main":{"isEncrypted":true,"primaryKey":true}}}. The application also creates endpoints for inserting and looking up the data as well as templates for the UI of the operators. The application is built in Flask and updates to a Firebase Cloud Firestore instance. Additionally, the application implements multiple ciphers such as AES, DES, Caesar, and Blowfish in Python. Furthermore, the application includes a load balancer made in TypeScript that implements port forwarding using HTTP proxy and multiple algorithms such as round-robin, weighted round-robin, random, and URL-hash for balancing the load.

Dynamic Backend Creation

The application creates a dynamic backend using a form generator that returns a configuration JSON file. The JSON file contains information about the form name, database name, and form fields. The schema format is {"form_name":"f","db_name":"db","form_fields":{"main":{"isEncrypted":true,"primaryKey":true}}}. Based on this information, the application creates a database schema and creates endpoints for inserting and looking up data.

Database Creation

The application creates a database for the schema present in the configuration file. The database schema is created using the information present in the configuration JSON file. The schema format is {"form_name":"f","db_name":"db","form_fields":{"main":{"isEncrypted":true,"primaryKey":true}}}. The application uses this information to create the necessary tables in the database.

Endpoints Creation

The application creates endpoints for inserting and looking up data. These endpoints are based on the schema present in the configuration JSON file. The endpoints are created using Flask, and they allow the user to interact with the database by inserting and looking up data.

UI Template Creation

The application creates templates for the UI of the operators. These templates are based on the schema present in the configuration JSON file. The templates are created using Flask, and they allow the user to interact with the database by inserting and looking up data.

Firebase Cloud Firestore Integration

The application updates to a Firebase Cloud Firestore instance. This integration allows the user to access the data stored in the database from anywhere in the world. The application updates the Firestore instance whenever new data is inserted or when existing data is updated.

Cipher Implementation

The application implements multiple ciphers such as AES, DES, Caesar, and Blowfish in Python. These ciphers are used to encrypt and decrypt the data stored in the database. The ciphers provide an additional layer of security to the data stored in the database.

Load Balancer Implementation

The application includes a load balancer made in TypeScript that implements port forwarding using HTTP proxy and multiple algorithms such as round-robin, weighted round-robin, random, and URL-hash for balancing the load. The load balancer distributes the incoming requests to different servers to ensure that no single server is overwhelmed with requests. This ensures that the application can handle a large number of requests without crashing or slowing down.

Conclusion

The application provides a dynamic backend creation tool that allows the user to create a backend based on the schema present in the configuration JSON file. The application creates a database, endpoints for inserting and looking up data, and UI templates for interacting with the database. Additionally, the application updates to a Firebase Cloud Firestore instance, implements multiple ciphers for encrypting and decrypting data, and includes a load balancer for distributing incoming requests.