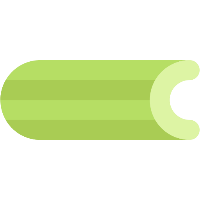
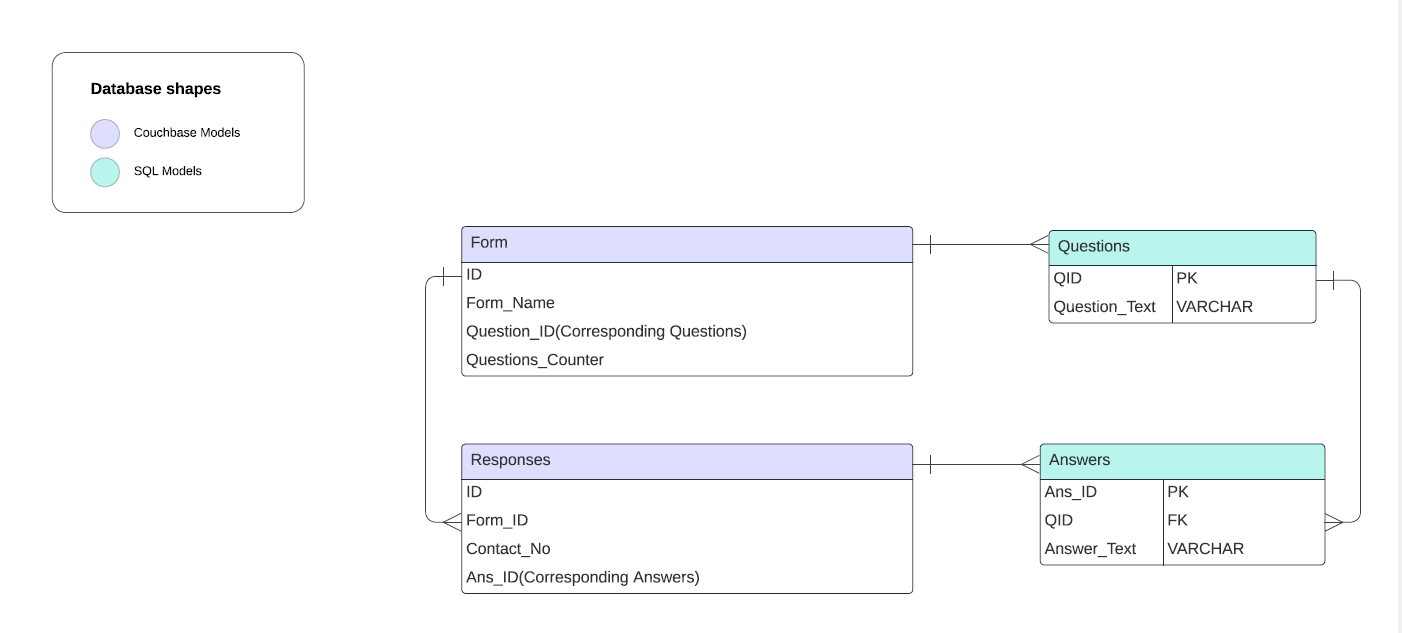
 **Custom Forms Assignment**

**Tech Stack**

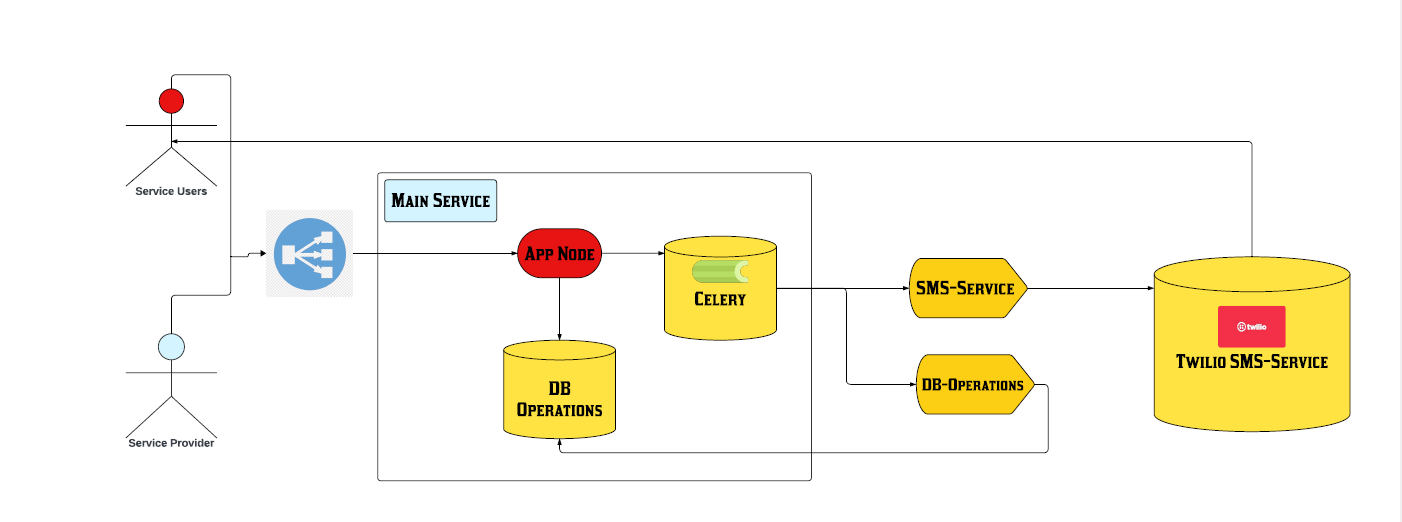


**Database**

****

1. In our Python application, we've implemented a hybrid approach to data storage using both Couchbase and MySQL databases. Couchbase, a NoSQL database, is utilized for storing flexible and unstructured data such as responses to forms. Its schema-less nature allows us to adapt quickly to changing data requirements, making it ideal for managing the varying structures of form responses efficiently.
2. **Couchbase's** scalability and high availability features make it an excellent choice for storing **forms** and it’s **responses**. The dynamic nature of form submissions often results in varying data structures, which can be efficiently managed by Couchbase's schema-less design. This flexibility allows us to adapt quickly to changes in form structures or data requirements without the need for schema migrations.
3. By using **MYSQL**  for **Questions** and **Answers** , With SQL queries, we can perform various analyses such as aggregations, filtering, and joining of data to extract valuable insights and metrics. This capability enables us to derive meaningful insights from the question and answer data stored in MySQL tables, empowering decision-making processes and enhancing user experiences based on data-driven insights.

**Architecture**

****

**Overview**

This project comprises a main app.py service and plug-and-play services for extensible functionality. The main service acts as the core application, while plug-and-play services, such as Twilio SMS,Google Sheets etc, can be easily integrated for various use cases.

### **Main Service**

The main service is the central component, processing requests, implementing logic, and interfacing with databases. The blueprint pattern in Flask promotes a modular and scalable architecture, making it easier to develop and maintain complex web applications.

### **Plug-and-Play Services**

These services integrate smoothly into the architecture, operating independently and enhancing the capabilities of the application as required.

**Key Characteristics and Benefits**

* **Scalability**: Independent service scaling.
* **Modularity**: Focused services for maintainability.
* **Decoupled Communication**: Resilient async interaction through taskqueue implementation.
* **Flexibility**: Extensible via plug-and-play services.

## Monitoring and Logging

## 1. Implemented Logging.tracebacks:

## Integrate logging in the application to capture tracebacks whenever exceptions occur.

## Use try-except blocks to catch exceptions and log them with appropriate severity levels.

## Include traceback information in the log message to provide context for debugging.

## Configure logging settings to control log levels, output format, and log file rotation.

## Can Review and Implement:-

## 1. Log Files Extraction:

## Set up a centralized logging system where log files from all instances of the application are stored.

## Implement a log management tool or service to collect and store log files centrally.

## Use log aggregation tools like Elasticsearch, Logstash, and Kibana (ELK stack) to search, analyze, and visualize log data.

## Consider integrating with cloud-based log management solutions such as AWS CloudWatch Logs or Google Cloud Logging for scalability and ease of management.

## 2. Creating a Log Model:

## Develop a database model to store log entries in a structured format.

## Define fields such as timestamp, severity level, log message, source module, and traceback information.

## Use an ORM (Object-Relational Mapping) library like SQLAlchemy to interact with the log database.

## Implement CRUD operations to insert, retrieve, update, and delete log records.

## Ensure proper indexing and optimization for efficient querying and analysis of log data.

## 3. Suggestions for Log File Extraction:

## Regularly archive and compress log files to save storage space and facilitate long-term retention.

## Implement log rotation to manage log file sizes and prevent them from growing indefinitely.

## Use log shipping or file transfer protocols to move log files to a central location for analysis.

## Implement access controls and encryption mechanisms to secure log files during transmission and storage.

## Scalability and Fault Tolerance

## To enhance scalability and bolster fault tolerance, App Engine Microservices leverage a container orchestration solution that provides several key benefits:

## Automated Scaling: App Engine automatically scales the microservices based on incoming traffic, ensuring optimal resource utilization and responsiveness during peak loads.

## High Availability: The platform ensures high availability by distributing microservices across multiple availability zones, minimizing downtime and ensuring continuity of service.

## Efficient Load Balancing: App Engine employs intelligent load balancing techniques to evenly distribute incoming traffic among microservice instances, preventing overload on any single instance.

## Seamless Updates: Deploying updates to microservices is seamless and non-disruptive, with App Engine handling the deployment process transparently, including versioning and rollback capabilities if needed.

## Self-Healing: In case of failures or crashes, App Engine automatically restarts or replaces unhealthy microservice instances, ensuring continuous operation without manual intervention. This self-healing capability enhances the resilience of the system and minimizes service disruptions.