[ expleo ]

Think bold, act reliable

# Citizen Science Application database for personal register

27 Sptember 2023

## Contents



## 1 Introduction

This report discuss the choice of the database and scheme of datas to use for personal register for this application :

**Multi-user application that will give recommendations to users, so they reduce ghg emission : Each user needs to have id, name, country and regions, then he can provide others information are specific to his country/regions and agriculture type.**

## 2 NoSQL vs SQL

Some problems with SQL : The application needs to accommodate frequent changes in data structure (e.g., adding new country-specific or region-specific fields such "Burning Savana"), it can be challenging to manage this types of datas cause we will have large sparse database. We need to do schema changes in an SQL database.
This will lead to some issues like : Storage Overhead,Slow Query Performance, Data Retrieval Complexity, Increased Memory Usage, Indexing Challenges, Data Import and Export

The solution is to do a schema Changes but it's look challenging in sql database. Another solution is to store users data using NoSQL database.

# 3 NoSQL Data base choice

## 3.1 Data base models

This is a short presentation for the four most NoSQL database models present in the literature.

- **Key-Value Stores:** are best suited for simple and fast retrieval of data based on a unique key.

  \+ High performance, scalability, and simplicity.

  — Limited query capabilities and not suitable for complex relationships between data.

  **Use case :** When there is a need to store and retrieve data with high throughput and low latency. Data access patterns are primarily lookup by a unique identifier (the key). You don't need complex querying or indexing.

- **Document Stores:** are great for semi-structured or unstructured data where each item can have different fields and structures.

  \+ Flexible schema, good for hierarchical data, and supports indexing for efficient querying.

  — May not be as performant as key-value stores for simple lookups.

  **Use case :** For semi-structured or unstructured, like JSON or XML documents that need to be queried and indexed data based on attributes within documents with flexible model.

- **Wide-Column or Tabular Stores:** Wide-column or tabular stores excel at handling large volumes of data with complex query requirements, especially in analytical and time-series use cases.

  \+ Excellent for complex querying, scalability, and support for time-series data.

  — Might have a steeper learning curve, and not ideal for transactional data.

  **Use case :** You have structured data with many attributes. Complex queries, including aggregations and analytics, are a primary use case. Scalability and high availability are critical.

- **Graph Databases:** are designed for data with complex relationships, making them ideal for social networks, recommendation engines, and knowledge graphs.

  \+ Efficient traversal of relationships, schema flexibility, and expressive querying with graph-specific languages.

  — May not perform as well as other models for simple data retrieval. Consider Using Graph Databases

## 3.2 My recomendation :

In my opinions, the best database model should be Document Stores (MongoDB) or key values (Amazon DynamoDB & Amazon Elastic Cache, Azure Table Storage & Azure Redis Cache ...). Another possibility could be a multimodel database that supports multiple storage models, but it will be difficult to implement.

# 4 From users inputs to database

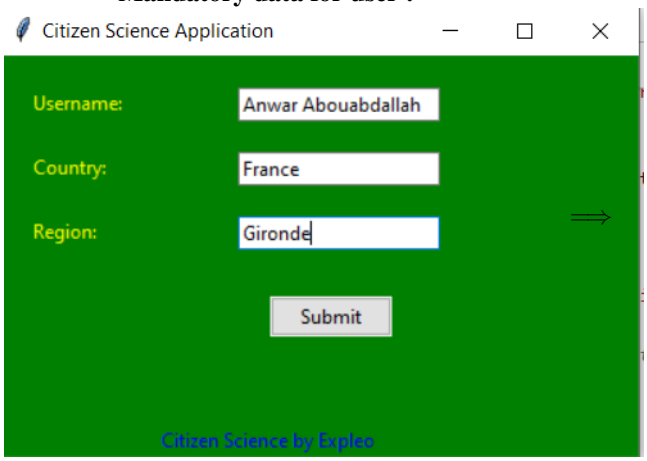
After a user sign up, the app should ask about first information such country, usernames and regions and create a route to structure datas into database.

1. **Front-End Data Collection:**

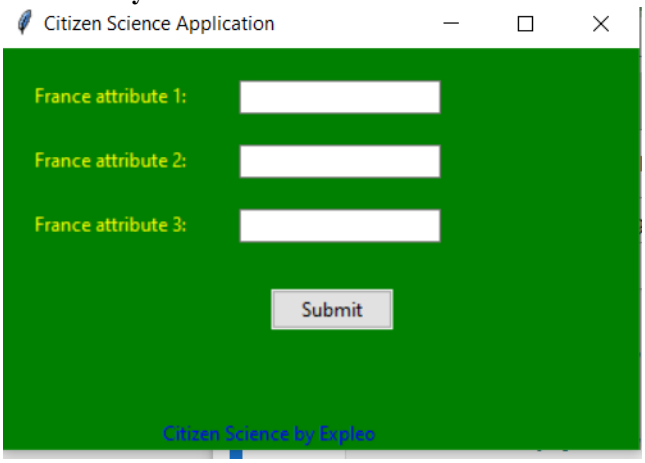
   - User interface (UI) application that allows users to input their data. This UI can include forms, text fields, dropdowns, and other input elements.

Example of front end[1] :

**Mandatory data for user :**




**Mandatory data to have recommendations :**




2. **User Input:**
   - Users interact with the UI to provide their data, such as name and any other relevant information your application requires. They get notification emails/application to ask them to complete their profiles by providing more data related to their country/regions/type of culture.
3. **Data Validation:** This validation helps prevent the submission of invalid data.
   - Firstly, we will implement client-side data validation to ensure that user input is correctly formatted and meets any required criteria.
   - Then, if there are enough users, the data validation can use some machine learning techniques such anomaly detection (Random Cut Forest) or ML models to predict the expected values for specific input fields based on historical data patterns. Another ml use case could be the combination of anomaly detection to identify missing data points or data gaps and ML models to impute missing values based on patterns in the existing data (knn for example).
4. **Data Serialization:**
   - We need to serialize the collected user data into a structured format that can be transmitted over the internet. JSON is a common choice for this purpose.
     Example of data in json format:

     54d39dd1-5be4-48f9-a0ac-6ff189d99098: {
     'Username': 'John Snow',
     'country': 'France',
     'country attributes': {'France1': 'at1', 'France2': 'at2'},
     'regions': 'Gironde' }

     In this example, each user have a uuid.
5. **HTTP POST Request:**
   - Use a mechanism like a button click or form submission to trigger an HTTP POST request from the front end to your back-end server.
     In the example, we use get method to save the data on a database, so when a user (for example me click) on submit this new colums is added to the database :
     '17cae163-9a66-454c-ae54-3751e6567e77': {'Username': 'Anwar Abouabdallah', 'country': 'France', 'country attributes': {}, 'regions': 'Gironde'}}

[1] Simple tkinter examples

In this example the data is stored in a dictionnary to simplify because I'm developping localy, but for the app we need to use a database such mongodb to benefit from the cloud. So we need the user data to be sent to the server for processing.

6. **Request Configuration:**
   - Configure the POST request to include the serialized user data in the request body. Set the appropriate headers, such as the `Content-Type` header, to indicate that you are sending JSON data.
7. **Back-End Handling:**
   - On the back end, create an API endpoint or route that corresponds to the URL where the front end sends the POST request.
8. **Request Parsing:**
   - In your back-end code, parse the incoming POST request and extract the JSON data from the request body.
9. **Front-End Feedback:**
   - In your front-end code, handle the response received from the back end. You can display a confirmation message/email to the user to let them know that their data has been added successfully.

## 4.1 Scheme

| **UUID** | **Attributes** |
|---|---|
| 54d39dd1-5be4-48f9-a0ac-6ff189d99098 | { "Username": "John Snow", "country": "France", "country attributes": { "France1": "at1", "France2": "at2" }, "regions": "Gironde" } |
| bba1fe40-e64c-4a9f-b51b-a7347bc8e0ea | { "Username": "John Mow", "country": "South Africa", "country attributes": { "South Africa1": "at1", "South Africa2": "at2" }, "regions": "Mpumalanga" } |
| f4f5434f-f200-4ef6-9a3f-f447a2d50dd8 | { "Username": "John Floow", "country": "Ireland", "country attributes": { "Ireland1": "at1", "Ireland2": "at2" }, "regions": "Ulster" } |

Table 1: Example of user Data

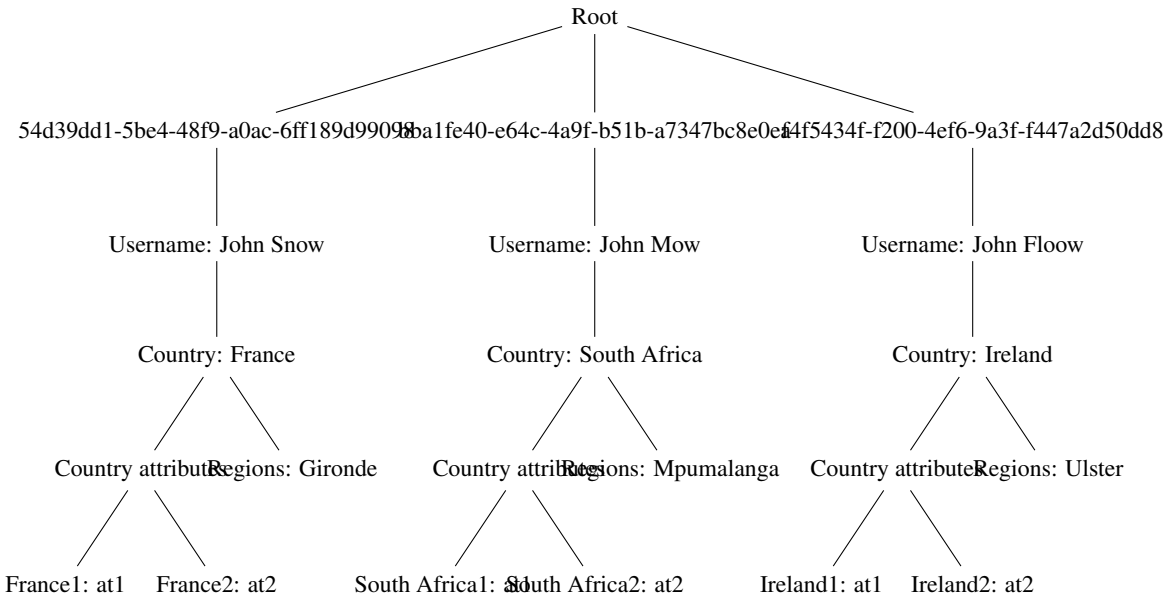



Figure 1: Data Structure visualisation

In the data structure you provided, the primary key is the UUID (Universally Unique Identifier) assigned to each user and act as the primary keys in your scheme.

Using this model we can query data, we can make the country as secondary key and querry it using this.

## 4.2 ETL, crawling and querrying the database azure tools:

- **Azure Data Catalog** Azure offers a Data Catalog known as **Azure Purview**, it's comprehensive data governance service that includes a data catalog that allows you to discover, classify, and manage metadata about your data assets, simplifying the process of discovering and understanding your data sources.
- **Crawler and Data Discovery** Azure Data Factory is a versatile data integration service that enables you to create data pipelines to extract data from various sources.
- **ETL (Extract, Transform, Load)** Azure provides several services for ETL tasks:
  - **Azure Data Factory** offers ETL capabilities with data flows, data wrangling, and transformations using mapping data flows or custom activities.
  - **Azure Databricks** is a powerful platform for ETL using Apache Spark. You can create data transformation workflows and execute Spark jobs for ETL tasks.
- **Data Querying and Analytics** For data querying and analytics in Azure, you can choose from various services:
  - **Azure Synapse Analytics** (formerly Azure SQL Data Warehouse) is a robust analytics platform for querying structured data using SQL.
  - **Azure Data Lake Analytics** allows running U-SQL queries on data stored in Azure Data Lake Storage, suitable for big data analytics and complex transformations.
  - **Azure SQL Database** supports SQL-based querying for structured data, suitable for transactional and analytical workloads.
  - **Azure Analysis Services** is a managed analytics service for creating and managing tabular and multidimensional data models for interactive analysis and reporting.
  - **Azure Stream Analytics** is used for real-time data processing and querying streaming data from various sources.

## 4.3 Needs

We need to design data engineering process using ETL to be able to retransform our database to sql one or to join it with other databases such climate databases management knowledge bases or to adapte it to some futures features engineering before machine learning operations.