**Adoption of Microservices Architecture**  
Date: September 6, 2023

**1. Architectural Decision**

Adopt a microservices architecture for the census system.

**2. Impacts and Implications**

* **Technology Choice:** The microservices architecture allows for flexibility in technology choice for each service, which can lead to a diverse technology stack.
* **Cost:** The microservices architecture may introduce additional operational complexity and costs due to the management of multiple services and their deployments.
* **Support Implications:** Each microservice will require individual monitoring, maintenance, and support.

**3. Problem & Constraints**

**3.1 Problem**

The problem is to design an efficient and scalable census system capable of handling data collection, statistics generation, and offline mobile data synchronization for census takers.

**3.2 Context**

The system is intended for use by the national statistics department to conduct censuses efficiently and accurately.

**3.3 Scope**

The system should cover data collection from multiple channels (web, mobile), storage, statistics generation, and offline data synchronization.

**3.4 Constraints**

* Limited budget for development and maintenance.
* Regulatory requirements for data security and privacy.
* The need for the system to be highly available and fault tolerant.

**3.5 Assumptions**

* Adequate development resources and expertise are available.
* Network connectivity is available for most census takers.
* Data volume and traffic patterns are subject to fluctuations.

**4. Solution Analysis**

**4.1. Solution Architecture**

The selected solution is a microservices architecture, where the census system is divided into loosely coupled, independently deployable services that handle specific functionalities. Key components include:

* **Client-side Census Capture:** A client application (web and mobile) for capturing census data.
* **API Gateway:** Serves as an entry point for requests, routing them to the appropriate microservices.
* **Microservices:** These include services for data capture, statistics generation, and mobile data synchronization.
* **Databases:** Centralized databases for census data storage and retrieval.
* **Security and Authentication:** Ensures data security and user authentication.
* **Monitoring and Logging:** To track service health and diagnose issues.

**4.2. Solutions Comparative Analysis**

Alternative architectures considered were a monolithic architecture and a serverless architecture:

* **Monolithic:** A single, integrated application would simplify development but could become a bottleneck and hinder scalability and flexibility.
* **Serverless:** Serverless architecture could offer scalability and cost savings but may not be suitable for all components (e.g., databases).

**4.3. Rationale**

The decision to adopt a microservices architecture was made because:

* Scalability: Microservices allow for individual scaling of components, accommodating varying workloads.
* Technology Flexibility: Different services can use the most appropriate technological stacks.
* Maintenance and Deployment: Independent services can be updated and deployed without affecting the entire system.
* Resilience: Failures in one service do not necessarily impact others.
* Offline Functionality: Synchronization functionality for the mobile app is better suited for a microservices approach.

**Authentication and Authorization**  
Date: September 6, 2023

**1. Architectural Decision**

Implement JSON Web Tokens (JWT) as the technology for authentication and authorization in the census system.

**2. Impacts and Implications**

* **Technology Choice:** Implementing JWT for authentication and authorization requires integrating JWT libraries or modules into the application stack, which may vary depending on the programming languages and frameworks used.
* **Cost:** JWT itself is a lightweight and cost-effective technology, but there may be development and maintenance costs associated with implementing and securing JWT-based authentication.
* **Support Implications:** Ongoing support and monitoring of JWT-based authentication are essential to ensure the security and functionality of the system.

**3. Problem & Constraints**

**3.1 Problem**

The problem is to design a secure authentication and authorization system for the census system, ensuring that only authorized users can access and modify census data.

**3.2 Context**

The system is intended for use by the national statistics department to conduct censuses efficiently and accurately.

**3.3 Scope**

The system should cover user authentication, role-based access control, and the protection of sensitive census data.

**3.4 Constraints**

* Regulatory requirements demand robust data security and privacy measures.
* The system must provide various levels of access to census data based on user roles.
* Budget constraints limit the use of costly authentication and authorization solutions.

**3.5 Assumptions**

* Adequate development resources and expertise are available to implement JWT-based authentication and authorization.
* Users will interact with the system through web and mobile applications.

**4. Solution Analysis**

**4.1. Solution Architecture**

The selected solution is to implement JWT-based authentication and authorization:

* **JWT for Authentication:** Users will obtain JWTs upon successful login. These JWTs will contain user claims and roles and will be signed by the server. Users will include their JWTs in the headers of their requests to access protected resources.
* **Role-Based Authorization:** Role-based access control (RBAC) will be implemented to determine users' permissions. Different user roles (e.g., admin, census taker) will be defined, each with specific access rights to census data and system functionalities.

**4.2. Solutions Comparative Analysis**

* **Session-Based Authentication:** Traditional session-based authentication mechanisms could be considered but may introduce server-side session management complexities and might not be as suitable for stateless, distributed systems.
* **OAuth 2.0:** OAuth 2.0 could be an alternative for certain scenarios, especially if the system requires integration with external identity providers or third-party applications.

**4.3. Rationale**

The decision to implement JWT-based authentication and authorization was made because:

* **Statelessness:** JWT is well-suited for stateless, distributed applications, providing scalability and flexibility.
* **Scalability:** JWTs can be efficiently verified without the need for server-side sessions, supporting scalability as the system grows.
* **Fine-Grained Access Control:** Role-based authorization offers granular control over user access, ensuring that users only have access to the resources and functionalities relevant to their roles.
* **Security:** JWTs can be signed to ensure their integrity and can include claims to support additional security measures, such as user role information.

**Selection of Go (Golang) for API Gateway Implementation**  
Date: September 6, 2023

**1. Architectural Decision**

Adopt the Go programming language for building the API Gateway component of the census system.

**2. Impacts and Implications**

* **Technology Choice:** Choosing Golang as the programming language for the API Gateway implies using the Go ecosystem and libraries for building and maintaining the gateway.
* **Cost:** Golang is an open-source language, and many of its libraries and tools are also open source, making it cost-effective for development.
* **Support Implications:** The decision requires access to developers with Golang expertise or a commitment to training and building expertise in the language.

**3. Problem & Constraints**

**3.1 Problem**

The problem is to select a technology stack for building the API Gateway, which will serve as the entry point for requests and routing them to the appropriate microservices in the census system.

**3.2 Context**

The system is intended for use by the national statistics department to conduct censuses efficiently and accurately.

**3.3 Scope**

The API Gateway should handle incoming requests from various clients (web, mobile) and route them to the relevant microservices. It should also perform tasks such as request validation, load balancing, and security enforcement.

**3.4 Constraints**

* The API Gateway must efficiently handle a potentially high volume of incoming requests and distribute them to microservices.
* Development resources and expertise are available, but there may be budget constraints.

**3.5 Assumptions**

* Adequate development resources and expertise are available to implement the API Gateway in Golang.
* The census system's microservices architecture has already been decided upon.

**4. Solution Analysis**

**4.1. Solution Architecture**

The selected solution is to use Golang for building the API Gateway. The API Gateway will perform the following functions:

* **Request Routing:** Route incoming requests from clients to the appropriate microservices based on the requested endpoints.
* **Load Balancing:** Distribute incoming requests across multiple instances of microservices to ensure even load distribution.
* **Security:** Enforce security measures such as authentication, rate limiting, and access control.

**4.2. Solutions Comparative Analysis**

* **Alternative Languages:** Other programming languages (e.g., Node.js, Python) could be considered for building the API Gateway. However, Golang is known for its performance and efficiency, which are essential for handling a high volume of requests.
* **API Gateway Services:** Consider using third-party API Gateway services like Amazon API Gateway or Google Cloud Endpoints. However, these services may introduce limitations and costs.

**4.3. Rationale**

The decision to use Golang for building the API Gateway was made because:

* **Performance and Efficiency:** Golang is known for its efficiency and low-level system programming capabilities, making it suitable for high-performance applications like API Gateway.
* **Concurrency:** Golang's native support for concurrency and parallelism is valuable for handling concurrent requests effectively.
* **Ecosystem and Libraries:** Golang has a rich ecosystem of libraries and packages for building web services and microservices, which can accelerate development.
* **Scalability:** Golang's lightweight footprint and strong support for building scalable applications align with the requirements of an API Gateway.

**Adopt Python for local statistics executable file**  
Date: November 20, 2023

**1. Architectural Decision**

Adopt Python through an .exe file for implementing a local client for the census system statistics generation.

**2. Impacts and Implications**

* **Technology Choice:** Choosing Python as the primary programming language for this local client is very useful, as it is well-suited for data processing and statistics generation. Also, creating an executable file allows us to keep the information private and updated for the DoS analysis.
* **Cost:** Python is open-source and free to use, which can be cost-effective for development and maintenance. Its libraries are open-source and free to use too.
* **Support Implications:** Adequate Python expertise should be available for development and ongoing support.

**3. Problem & Constraints**

**3.1. Problem**

The problem is to select a technology stack for implementing a responsible local client for generating statistics in the census system.

**3.2. Context**

The system is intended for use by the national statistics department to conduct censuses efficiently and accurately.

**3.3. Scope**

The client should process and analyze census data to generate statistical reports and insights. It should provide statistical functionalities for census data visualization, analysis, and reporting.

**3.4. Constraints**

* The client must efficiently process and analyze large volumes of census data.
* Budget constraints may limit the use of costly software solutions.
* Development resources and expertise are available, particularly in Python

**3.5. Assumptions**

* Python is a suitable language for implementing the statistics client.
* The client will have access to census data through API services.

**4. Solution Analysis**

**4.1. Solution Architecture**

The selected solution is to use Python for building the local client for statistics generation. It will be presented as an executable file to receive, process, analyze and generate statistics. The client Will have de following functions:

* **Data Processing:** Process and aggregate census data from the central database or API services.
* **Statistical Analysis:** Perform statistical analysis on census data to generate reports, charts, and insights.
* **Data Visualization:** Create a folder with all the charts and reports of the census’ responses, divided by subfolders for every form’s section.
* **API Integration:** Connect to API services for data retrieval and synchronization.

**4.2. Solutions Comparative Analysis**

* **Frameworks for Web Visualization:** Consider to use a framework like Django for statistics generation to create a client for web visualization.
* **Statistical Software:** Explore the use of specialized statistical software like R or SPSS. However, integrating such software may introduce complexity and licensing costs.

**4.3. Rationale**

The decision to use Django for implementing the statistics client was made because:

* **Python Expertise:** Python is a versatile language with a strong ecosystem for data analysis and statistics.
* **Cost-Effective:** Python is open-source and free, aligning with budget constraints.
* **Scalability:** Python libraries for data analysis provide scalability options as census data volume grows.
* **Experience in Python:** The development team

**React Native for census collector mobile app**  
Date: September 6, 2023

**1. Architectural Decision**

Develop the Mobile App Client using React Native, with AsyncStorage as the local database for offline functionality and synchronization with the central database.

**2. Impacts and Implications**

**Document any implications of the decision, such as technology choice, cost, and support implications.**

* **Technology Choice:** Choosing React Native for mobile app development allows for cross-platform development, and AsyncStorage facilitates offline data storage and synchronization.
* **Cost:** React Native and AsyncStorage are open-source and cost-effective technologies, aligning with budget constraints.
* **Support Implications:** Development teams should have expertise in React Native for mobile apps.

**3. Problem & Constraints**

**3.1 Problem**

The problem is to select a technology stack for implementing the Mobile App Client, which is responsible for collecting census data on mobile devices. The client must support offline functionality and synchronization with the central database.

**3.2 Context**

The system is intended for use by the national statistics department to conduct censuses efficiently and accurately.

**3.3 Scope**

The Mobile App Client should allow census takers to collect data offline, store it locally, and synchronize with the central database when an internet connection is available. It must provide an intuitive user interface for data collection.

**3.4 Constraints**

* The Mobile App Client must efficiently handle data collection and synchronization.
* Budget constraints limit the use of costly software solutions.
* Development resources and expertise are available for React Native.

**3.5 Assumptions**

* Census takers will be using mobile devices to collect data.
* There may be areas with limited or no internet connectivity.
* Data collected offline must be securely synchronized with the central database.

**4. Solution Analysis**

**4.1. Solution Architecture**

The selected solution is to use React Native for building the Mobile App Client and AsyncStorage as the local database for offline data storage and synchronization. The Mobile App Client will have the following functions:

* **Data Collection:** Allow census takers to capture census data on mobile devices using the app.
* **Offline Functionality:** Utilize AsyncStorage for storing data locally on the device, enabling data collection in offline or low-connectivity areas.
* **Data Synchronization:** When an internet connection is available, synchronize locally stored data with the central database (e.g., CouchDB)
* **User Authentication:** Implement user authentication to ensure authorized access to data collection functionalities.

**4.2. Solutions Comparative Analysis**

* **Alternative Mobile App Development Frameworks:** Consider other mobile app development frameworks (e.g., native development, Flutter) for the Mobile App Client. React Native is chosen for its cross-platform capabilities and large developer community.

**4.3. Rationale**

The decision to use React Native for mobile app development and AsyncStorage for offline data storage and synchronization was made because:

* **Cross-Platform Compatibility:** React Native enables cross-platform development, reducing development effort and ensuring the app can run on both Android and iOS devices.
* **Cost-Effective:** Both React Native and AsyncStorage are open-source technologies, aligning with budget constraints.

**MongoDB as Census System Database**  
Date: November 23, 2023

**1. Architectural Decision**

Adopt MongoDB as the principal database for the census system.

**2. Impacts and Implications.**

* **Technology Choice:** Selecting MongoDB involves using a NoSQL database that supports flexible document storage.
* **Cost:** MongoDB is Server-Side Public License and free to use, aligning with budget constraints.
* **Support Implications:** Ensure that there is adequate expertise and support for MongoDB administration and maintenance.

**3. Problem & Constraints**

**3.1 Problem**

The problem is to determine the principal database technology for storing and managing census data efficiently.

**3.2 Context**

The system is intended for use by the national statistics department to conduct censuses efficiently and accurately.

**3.3 Scope**

MongoDB will serve as the central database for storing census data, ensuring data integrity, scalability, and support for replication.

**3.4 Constraints**

* Limited budget for development and maintenance.
* Regulatory requirements for data security and privacy.
* Development and maintenance resources must be available.

**3.5 Assumptions**

* Adequate development resources and expertise are available.
* The database must efficiently handle a potentially large volume of census data.
* MongoDB's support for flexible document storage aligns with the dynamic nature of census data.

**4. Solution Analysis**

**4.1. Solution Architecture**

The selected solution is to use MongoDB as the principal database for the census system. MongoDB will provide the following features:

* **Document-Oriented Storage:** MongoDB's document-oriented NoSQL architecture allows for flexible storage of census data without a predefined schema.
* **Scalability:** MongoDB supports horizontal scalability, allowing the system to handle an increasing volume of census data.

**4.2. Solutions Comparative Analysis**

* **Alternative Databases:** Consider other database solutions (e.g., CouchDB, PostgreSQL) for storing census data. MongoDB is chosen for its document-oriented and ow cost.
* **Relational Databases:** Evaluate the use of relational databases for structured data. However, MongoDB's flexibility is advantageous for accommodating the dynamic nature of census data.

**4.3. Rationale**

The decision to adopt CouchDB as the principal database was made because:

* **Flexible Document Storage:** MongoDB's document-oriented approach allows for flexible and dynamic storage of census data without the need for a rigid schema.
* **Scalability:** MongoDB's scalability features align with the system's requirements to handle a potentially large volume of census data.
* **Replication Support:** MongoDB's built-in replication capabilities ensure data consistency and availability, supporting distributed and fault-tolerant architecture.
* **Cost-Effective:** MongoDB is a Server-Side Public License database, aligning with budget constraints.

This decision addresses the need for a scalable, flexible, and cost-effective database solution for storing and managing census data. MongoDB's features make it well-suited for the dynamic and distributed nature of the census system.

**Adoption of React.js for Forms Client Frontend**  
Date: November 19, 2023

**1. Architectural Decision**

Develop the Forms Client using React.js for the front-end to create a dynamic and user-friendly interface for capturing census data through web-based forms.

**2. Impacts and Implications**

* **Technology Choice:** Choosing React.js for the front-end enables the development of a responsive and interactive user interface for census data collection.
* **Cost:** React.js is an open-source library, contributing to cost-effectiveness in the development process.
* **Support Implications:** Development teams should possess expertise in React.js for efficient implementation and support.

**3. Problem & Constraints**

**3.1 Problem**

The problem is to determine the technology stack for implementing the Forms Client, ensuring a seamless and user-friendly experience for census data collection.

**3.2 Context**

The system is intended for use by the national statistics department to conduct censuses efficiently and accurately.

**3.3 Scope**

The Forms Client should provide a visually appealing and responsive interface for capturing census data through web-based forms.

**3.4 Constraints**

* The Forms Client must efficiently handle a potentially high volume of form submissions.
* Budget constraints limit the use of unnecessary technologies.

**3.5 Assumptions**

* Users will interact with the system through web-based forms.
* Form data will be securely transmitted to the central database.

**4. Solution Analysis**

**4.1. Solution Architecture**

The selected solution is to use React.js for building the front-end of the Forms Client. The Forms Client will have the following functions:

* **Form Rendering:** Utilize React.js to dynamically render census forms, providing users with a visually appealing and interactive interface for data entry.
* **User Interaction:** Leverage React.js capabilities for handling user interactions, ensuring a seamless and responsive experience during form filling.
* **Data Submission:** Implement mechanisms to securely submit completed forms to the central database (e.g., CouchDB) for permanent storage.

**4.2. Solutions Comparative Analysis**

* **Alternative Frontend Technologies:** Consider other front-end technologies (e.g., Angular, Vue.js) for form rendering. React.js was chosen for its component-based architecture and widespread adoption.
* **Scalability:** React.js supports the development of scalable user interfaces, accommodating potential increases in form submissions.

**4.3. Rationale**

The decision to use React.js for the frontend of the Forms Client was made because:

* **User-Friendly Interface:** React.js allows the creation of a dynamic and user-friendly interface, enhancing the user experience during census data collection.
* **Developer Productivity:** React.js's component-based architecture facilitates modular development, improving developer productivity and code maintainability.
* **Cost-Effective:** React.js is an open-source library, aligning with budget constraints and contributing to cost-effectiveness.

**Adoption of AsyncStorage as Local Mobile Data Base**  
Date: November 19, 2023

**1. Architectural Decision**

Implement the local mobile database for the Mobile App Client using AsyncStorage in React Native, leveraging the Expo framework for streamlined development and cross-platform compatibility.

**2. Impacts and Implications**

* **Technology Choice:** The decision utilizes AsyncStorage, a local storage solution in React Native, and Expo, a framework that simplifies the development process for React Native applications.
* **Cost:** AsyncStorage is a built-in feature of React Native, and Expo is an open-source framework, contributing to cost-effectiveness.
* **Support Implications:** Development teams should have expertise in React Native and Expo for efficient implementation and support.

**3. Problem & Constraints**

**3.1 Problem**

The problem is to determine the local mobile database solution for the Mobile App Client, allowing for offline data storage and synchronization.

**3.2 Context**

The system is intended for use by the national statistics department to conduct censuses efficiently and accurately.

**3.3 Scope**

The local mobile database should enable the Mobile App Client to store census data locally on devices using React Native and Expo.

**3.4 Constraints**

* The local mobile database must efficiently handle offline data storage and synchronization.
* Budget constraints limit the use of unnecessary technologies.

**3.5 Assumptions**

* Census takers will use mobile devices to collect data in areas with limited or no internet connectivity.
* Data collected offline must be securely synchronized with the central database.

**4. Solution Analysis**

**4.1. Solution Architecture**

The selected solution is to use AsyncStorage in React Native for implementing the local mobile database in the Mobile App Client. Expo, a framework for React Native development, is leveraged for streamlined development and cross-platform compatibility. The Mobile App Client will have the following functions:

* **Offline Data Storage:** Utilize AsyncStorage to locally store census data on mobile devices, allowing census takers to collect data in areas with limited or no internet connectivity.
* **Data Synchronization:** When an internet connection is available, implement mechanisms to securely synchronize locally stored data with the central database (e.g., CouchDB) through the API Gateway.

**4.2. Solutions Comparative Analysis**

* **Alternative Local Databases:** Consider other local database solutions (e.g., SQLite) for offline data storage. AsyncStorage was chosen for its simplicity and integration with React Native.
* **Expo Framework:** Leverage the Expo framework for React Native development to simplify the development process and enhance cross-platform compatibility.

**4.3. Rationale**

The decision to use AsyncStorage in React Native and the Expo framework for the local mobile database was made because:

* **Offline Data Storage:** AsyncStorage provides a straightforward solution for local data storage in React Native, supporting offline data collection in the Mobile App Client.
* **Synchronization Support:** AsyncStorage, combined with the Expo framework, allows for efficient synchronization of locally stored data with the central database when an internet connection is available.
* **Expo's Development Benefits:** Leveraging the Expo framework simplifies the development process, enhances cross-platform compatibility, and contributes to a more streamlined and efficient development workflow.