Here is the updated Software Design Specification (SDS) for MyGovInsights Pro, redesigned with an expert-level perspective, focusing on strategic technical decisions, architectural resilience, and enterprise-grade considerations suitable for a critical national project.

This document serves as a high-fidelity blueprint, reflecting the meticulous planning and robust design principles employed by seasoned professionals.

* **Tier 2: Application Layer (Microservices Landscape)**
  + **Identity & Access Management (IAM) Service:** Handles authentication, authorization, user profiles, and national ID integration.
  + **Feedback Management Service:** Manages the entire lifecycle of citizen feedback, from ingestion to status updates.
  + **Reporting & Analytics Service:** Aggregates, processes, and presents data insights.
  + **Notification Service:** Orchestrates multi-channel communication (email, SMS, in-app).
  + **Administrative Operations Service:** Provides APIs for managing administrative tasks (e.g., user moderation, content management).
  + **System Configuration Service:** Manages dynamic system settings and master data (e.g., service categories).
  + **API Gateway:** The single entry point, responsible for routing, authentication pre-checks, rate limiting, and potentially response aggregation.

<div align="center"> <h4>Fig 2. Tier 2 Decomposition: Application Microservices Landscape</h4> <p>This diagram elaborates on the logical components within the Application Tier, showcasing their interdependencies and primary interactions.</p> </div>

graph TD

subgraph Client Interfaces

AdminUI(Admin Dashboard UI)

CitizenWebUI(Citizen Web Portal UI)

CitizenPWA(Citizen PWA)

SMS\_USSD\_TG(SMS/USSD/Telegram Interface)

end

subgraph Core Microservices

direction LR

IAMService[Identity & Access Management Service]

FeedbackService[Feedback Management Service]

ReportAnalyticsService[Reporting & Analytics Service]

NotificationService[Notification Service]

AdminOpsService[Administrative Operations Service]

SystemConfigService[System Configuration Service]

end

subgraph Data Management

DatabaseCluster(MongoDB Database Cluster)

FileStorage(Secure File Storage)

end

AdminUI -- "REST API" --> AdminOpsService

AdminUI -- "REST API" --> ReportAnalyticsService

AdminUI -- "REST API" --> SystemConfigService

AdminUI -- "REST API" --> FeedbackService

CitizenWebUI -- "REST API" --> FeedbackService

CitizenWebUI -- "REST API" --> IAMService

CitizenPWA -- "REST API" --> FeedbackService

CitizenPWA -- "REST API" --> IAMService

SMS\_USSD\_TG -- "Ingests" --> FeedbackService

SMS\_USSD\_TG -- "Authenticates via" --> IAMService

IAMService --> DatabaseCluster

FeedbackService --> DatabaseCluster

ReportAnalyticsService --> DatabaseCluster

AdminOpsService --> DatabaseCluster

SystemConfigService --> DatabaseCluster

FeedbackService -- "Stores/Retrieves" --> FileStorage

ReportAnalyticsService -- "Queries" --> DatabaseCluster

FeedbackService -- "Triggers" --> NotificationService

AdminOpsService -- "Triggers" --> NotificationService

* **Tier 3: Granular Service Capabilities**
  + **Identity & Access Management (IAM):** User authentication (login, logout), registration (citizen, admin), profile management, role-based access data, Fayda ID verification.
  + **Feedback Management:** Submission handling, categorization, content validation, status updates, attachment processing.
  + **Reporting & Analytics:** Data aggregation, statistical computation, dashboard rendering data, report generation (CSV/JSON).
  + **Notification Service:** Email dispatch, SMS delivery, in-app notification management, template rendering.

<div align="center"> <h4>Fig 3. Tier 3 Decomposition: Granular Service Capabilities</h4> <p>This diagram provides a deeper dive into the sub-components and key functionalities encapsulated within specific microservices.</p> </div>

graph LR

subgraph Identity & Access Management Service

UserAuthSubsystem(User Authentication)

UserRegSubsystem(User Registration)

ProfileMgmtSubsystem(Profile Management)

RolePolicyEngine(Role & Policy Enforcement)

FaydaIdConnector(Fayda ID Integration Connector)

end

subgraph Feedback Management Service

FeedbackIngestion(Feedback Ingestion)

ContentValidation(Content Validation & Sanitization)

CategoryAssignment(Category Assignment)

AttachmentProcessing(Attachment Processing)

FeedbackLifecycle(Feedback Lifecycle Management)

RatingCapture(Rating Capture)

end

subgraph Reporting & Analytics Service

DataAggregator(Data Aggregator)

MetricCalculator(Metric Calculator)

VisualizationGenerator(Visualization Data Generator)

ReportFormatter(Report Formatter: CSV/JSON)

TrendAnalysisEngine(Trend Analysis Engine)

end

subgraph Notification Service

EmailDispatch(Email Dispatcher)

SMSDelivery(SMS Delivery)

InAppNotificationMgr(In-App Notification Manager)

TemplateRenderer(Notification Template Renderer)

ChannelRouter(Channel Routing Logic)

End

**2.4 Enterprise Data Flow Analysis**

Data flow within MyGovInsights Pro is characterized by secure, high-throughput pathways, with distinct patterns for synchronous API interactions and asynchronous event-driven processes.

1. **Citizen Feedback Ingestion Pipeline:**
   * **Initiation:** Citizen interacts with a client application (Web, PWA, SMS, USSD, Telegram).
   * **Data Transmission:** Feedback data (content, category, rating, optional attachments, user context) is securely transmitted via HTTPS to the **API Gateway**.
   * **Edge Processing:** The API Gateway performs initial validation, rate limiting, and authenticates the request (anonymous or authenticated user's JWT).
   * **Service Routing:** Request is routed to the **Feedback Management Service**.
   * **Core Processing:** Feedback Management Service validates data integrity, processes attachments (uploading to **File Storage**), assigns an initial status, and persists the Feedback record in the **MongoDB Database Cluster**.
   * **Asynchronous Trigger:** The Feedback Management Service then publishes an event (e.g., "NewFeedbackSubmitted") to an internal message broker (future state), which is consumed by the **Notification Service** for administrator alerts.
   * **Confirmation:** A success response is returned through the API Gateway to the client application, confirming submission.
2. **Administrator Insights & Action Workflow:**
   * **Authentication:** An administrator logs in via the **Admin Dashboard UI**. Credentials are sent to the API Gateway, routed to the **Identity & Access Management (IAM) Service**, which authenticates against the Database Cluster (and potentially Fayda ID for admin accounts) and issues a JWT.
   * **Dashboard Interaction:** With a valid JWT, the Admin Dashboard UI sends requests for feedback lists, detailed views, status updates, or report generation to the API Gateway.
   * **Authorization & Routing:** The API Gateway validates the JWT for authenticity and authorization (RBAC/ABAC) before routing requests to the appropriate services: **Administrative Operations Service** for feedback lifecycle, **Reporting & Analytics Service** for data insights.
   * **Data Retrieval/Manipulation:** These services interact with the **MongoDB Database Cluster** to retrieve or update data, strictly adhering to role-based access policies.
   * **Report Generation:** For complex reports, the Reporting & Analytics Service might perform extensive data aggregation and formatting before returning the generated report (e.g., CSV stream) to the Admin Dashboard UI.
   * **Notifications:** Status updates or critical actions taken by administrators can trigger the **Notification Service** to inform other relevant parties.

<div align="center"> <h4>Fig 4. UML Deployment Diagram</h4> <p>This diagram provides a high-level view of how software components are physically deployed across infrastructure nodes, emphasizing resilience and scalability.</p> </div>

graph TD

subgraph Public Cloud Region (e.g., "East Africa")

LB[Load Balancer: HTTP/S Ingress] --> WebServerGroup(Web Server Farm: Nginx/Apache)

WebServerGroup --> APIGatewayCluster(API Gateway Cluster: Node.js/Express)

APIGatewayCluster --> ServiceMesh(Service Mesh: e.g., Istio, Linkerd)

subgraph Kubernetes Cluster (Managed Service)

ServiceMesh --> IAMServicePod[Identity & Access Management Service Pods]

ServiceMesh --> FeedbackServicePod[Feedback Management Service Pods]

ServiceMesh --> ReportAnalyticsServicePod[Reporting & Analytics Service Pods]

ServiceMesh --> NotificationServicePod[Notification Service Pods]

ServiceMesh --> AdminOpsServicePod[Administrative Operations Service Pods]

ServiceMesh --> SystemConfigServicePod[System Configuration Service Pods]

ServiceMesh --> ExternalIntegrationProxy[External Integration Proxy]

end

IAMServicePod -- "Connects via TLS" --> MongoDBCuster(MongoDB Atlas Cluster)

FeedbackServicePod -- "Connects via TLS" --> MongoDBCuster

ReportAnalyticsServicePod -- "Connects via TLS" --> MongoDBCuster

AdminOpsServicePod -- "Connects via TLS" --> MongoDBCuster

SystemConfigServicePod -- "Connects via TLS" --> MongoDBCuster

FeedbackServicePod -- "Uploads/Downloads" --> SecureBlobStorage(Secure Blob Storage: S3-compatible)

NotificationServicePod -- "API Calls" --> ExternalEmailService(External Email Service API)

NotificationServicePod -- "API Calls" --> ExternalSMSService(External SMS Gateway API)

ExternalIntegrationProxy -- "API Calls" --> FaydaNationalIDAPI(Fayda National ID API)

MonitoringAgents(Monitoring & Logging Agents) -- "Collects Data" --> LB

MonitoringAgents -- "Collects Data" --> APIGatewayCluster

MonitoringAgents -- "Collects Data" --> Kubernetes Cluster

MonitoringAgents -- "Collects Data" --> MongoDBCuster

MonitoringAgents -- "Collects Data" --> SecureBlobStorage

MonitoringAgents -- "Feeds" --> CentralizedObservability(Centralized Observability Platform: Metrics, Logs, Traces)

end

UserDevice[User Device (Browser/Mobile/Basic Phone)] -- HTTPS/SMS/USSD --> LB

CentralizedObservability -- "Alerts" --> OpsTeam(Operations Team)

## 3. Data Design

The data architecture for MyGovInsights Pro is engineered for **integrity, high availability, performance, and stringent security**, accommodating both structured and semi-structured data needs inherent in civic feedback.

### 3.1 Conceptual and Logical Data Model

The core data model is designed to be highly normalized where relational integrity is critical (e.g., User-Feedback relationship) but leverages MongoDB's document-oriented flexibility for evolving or nested data structures (e.g., feedback content with rich attributes).

<div align="center"> <h4>Fig 5. Core Domain Class Model (Logical)</h4> <p>This diagram illustrates the primary entities, their attributes, and relationships, forming the foundation of the system's data management.</p> </div>

classDiagram

class CitizenUser {

+String userId

-String firstName

-String lastName

-String email \*UNIQUE\*

-String phoneNumber \*UNIQUE\*

-String passwordHash

+Boolean isVerified

-String nationalId \*UNIQUE, OPTIONAL\*

+DateTime createdAt

+String preferredLanguage

+List~String~ roles (e.g., ["citizen", "verified"])

+List~Feedback~ feedbackHistory()

}

class AdminUser {

+String adminId

-String username \*UNIQUE\*

-String email \*UNIQUE\*

-String passwordHash

+String role (e.g., "LOCAL\_ADMIN", "REGIONAL\_ADMIN", "FEDERAL\_ADMIN", "SYSTEM\_ADMIN")

+DateTime lastLoginAt

+String assignedJurisdiction (optional, e.g., "Addis Ababa")

+manageFeedback(Feedback): void

+generateReports(criteria): Report

+updateSystemConfig(config): void

}

class ServiceCategory {

+String categoryId

+String name \*UNIQUE\*

+String description

+String iconUrl (optional)

+DateTime createdAt

+String createdByAdminId

}

class Feedback {

+String feedbackId

+String userId FK

+String categoryId FK

+String title (optional)

+String content

+Integer rating (1-5, optional)

+DateTime submittedAt

+String currentStatus (e.g., "PENDING\_REVIEW", "IN\_PROGRESS", "RESOLVED", "CLOSED")

+String channel (e.g., "WEB", "PWA", "SMS", "USSD", "TELEGRAM")

+String location (GeoJSON, optional)

+List~String~ attachmentIds FK

+List~Note~ internalNotes

+DateTime lastUpdatedAt

+String lastUpdatedByAdminId (optional)

+submit(data): Feedback

+updateStatus(newStatus, adminId): void

+addNote(note, adminId): void

}

class IssueReport {

+String reportId

+String userId FK

+String title (optional)

+String description

+DateTime reportedAt

+String status (e.g., "OPEN", "INVESTIGATING", "TRIAGED", "RESOLVED")

+String severity (e.g., "LOW", "MEDIUM", "HIGH", "CRITICAL")

+String assignedToAdminId FK (optional)

+List~Note~ internalNotes

+create(data): IssueReport

+updateStatus(newStatus, adminId): void

+assign(adminId): void

}

class Attachment {

+String attachmentId

+String feedbackId FK

+String fileName

+String fileType

+String fileUrl (secure cloud URL)

+DateTime uploadedAt

+String uploadedByUserId FK

}

class Notification {

+String notificationId

+String recipientId FK

+String message

+DateTime sentAt

+Boolean isRead

+String type (e.g., "FEEDBACK\_UPDATE", "ISSUE\_ASSIGNED", "SYSTEM\_ALERT")

+String channel (e.g., "EMAIL", "SMS", "IN\_APP")

+String relatedEntityId (optional)

+String relatedEntityType (optional)

+send(recipient, message, type, channel): void

}

class AuditLog {

+String logId

+String actorId (userId or adminId)

+String actorType (e.g., "CITIZEN", "ADMIN")

+String actionType (e.g., "FEEDBACK\_SUBMITTED", "STATUS\_UPDATED", "LOGIN\_SUCCESS")

+String entityType (e.g., "FEEDBACK", "USER", "CATEGORY")

+String entityId (optional)

+Map~String, Any~ details (JSON)

+DateTime timestamp

+record(actorId, action, entity, details): void

}

class Note {

+String noteId

+String content

+DateTime createdAt

+String createdByAdminId FK

}

CitizenUser "1" -- "0..\*" Feedback : submits

CitizenUser "1" -- "0..\*" IssueReport : reports

ServiceCategory "1" -- "0..\*" Feedback : belongs\_to

Feedback "1" -- "0..\*" Attachment : has

Feedback "1" -- "0..\*" Note : has

IssueReport "1" -- "0..\*" Note : has

AdminUser "1" -- "0..\*" Feedback : manages

AdminUser "1" -- "0..\*" IssueReport : manages

AdminUser "1" -- "0..\*" ServiceCategory : manages

AdminUser "1" -- "0..\*" AdminUser : configures

AdminUser "1" -- "0..\*" Note : creates

Notification "1" -- "0..\*" CitizenUser : targets

Notification "1" -- "0..\*" AdminUser : targets

AuditLog "0..\*" -- "1" CitizenUser : relates\_to

AuditLog "0..\*" -- "1" AdminUser : relates\_to

### 3.2 Comprehensive Data Dictionary

This section outlines the critical data entities, their attributes, types, invariants, and purpose. Strict adherence to these definitions ensures data consistency and quality.

#### CitizenUser Entity

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Type | Invariant | Description |
| userId | String | **PK**, Immutable, GUID/UUID format | Unique identifier for each citizen user. |
| firstName | String | Non-null, Min length 1, Max length 50 | Citizen's first name. |
| lastName | String | Non-null, Min length 1, Max length 50 | Citizen's last name. |
| email | String | **Unique**, Non-null, Valid email format | Primary contact email, used for login. |
| phoneNumber | String | **Unique**, Non-null, E.164 format (e.g., +2519...) | Primary contact phone number, used for SMS/USSD channels. |
| passwordHash | String | Non-null, Result of secure hashing algorithm (e.g., Bcrypt, Argon2) | Hashed and salted password for secure storage. |
| isVerified | Boolean | Default: false | Indicates if the user's identity has been verified (e.g., via Fayda ID). |
| nationalId | String | **Unique if present**, Min length 5, Max length 20 (based on Fayda spec), Optional | National ID for integration with Fayda. |
| createdAt | DateTime | Auto-generated, Immutable | Timestamp of user account creation. |
| preferredLanguage | String | Valid ISO 639-1 code (e.g., 'en', 'am', 'om', 'ti', 'so'), Default: 'en' | User's preferred language for UI and notifications. |
| roles | List<String> | Contains "citizen", "verified" or "anonymous" | Granular roles defining user privileges. |

Export to Sheets

#### AdminUser Entity

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Type | Invariant | Description |
| adminId | String | **PK**, Immutable, GUID/UUID format | Unique identifier for each administrator. |
| username | String | **Unique**, Non-null, Min length 3, Max length 50 | Admin's login username. |
| email | String | **Unique**, Non-null, Valid email format | Admin's primary contact email. |
| passwordHash | String | Non-null, Result of secure hashing algorithm | Hashed and salted password. |
| role | String | Non-null, Allowed values: LOCAL\_ADMIN, REGIONAL\_ADMIN, FEDERAL\_ADMIN, SYSTEM\_ADMIN | Defines the administrative privileges and scope. |
| lastLoginAt | DateTime | Auto-updated | Timestamp of the last successful login. |
| assignedJurisdiction | String | Optional, References a valid geographical entity ID (e.g., Woreda, Zone, Region) or "National" | Defines the scope of an admin's feedback management (local/regional). |

Export to Sheets

#### Feedback Entity

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Type | Invariant | Description |
| feedbackId | String | **PK**, Immutable, GUID/UUID format | Unique identifier for each feedback submission. |
| userId | String | **FK** to CitizenUser.userId, Non-null | ID of the citizen who submitted the feedback. |
| categoryId | String | **FK** to ServiceCategory.categoryId, Non-null | ID of the service category the feedback pertains to. |
| title | String | Optional, Max length 100 | Short title summarizing the feedback. |
| content | String | Non-null, Min length 10, Max length 2000 | Detailed text content of the feedback. |
| rating | Integer | Optional, Range: 1 to 5 (inclusive) | Numerical rating provided by the citizen (1-poor, 5-excellent). |
| submittedAt | DateTime | Auto-generated, Immutable, submittedAt <= current\_timestamp | Timestamp when the feedback was submitted. |
| currentStatus | String | Non-null, Allowed values: PENDING\_REVIEW, IN\_PROGRESS, RESOLVED, CLOSED, ARCHIVED | Current processing status of the feedback. |
| channel | String | Non-null, Allowed values: WEB, PWA, SMS, USSD, TELEGRAM | Channel through which the feedback was submitted. |
| location | GeoJSON | Optional, Valid GeoJSON Point or Polygon structure | Geographical coordinates or area related to the feedback. |
| attachmentIds | List<String> | Optional, List of FKs to Attachment.attachmentId | IDs of associated attachments (e.g., images). |
| internalNotes | List<Note> | Optional, Embedded array of Note objects | Internal notes added by administrators during feedback processing. |
| lastUpdatedAt | DateTime | Auto-updated, lastUpdatedAt >= submittedAt | Timestamp of the last update to the feedback record. |
| lastUpdatedByAdminId | String | Optional, **FK** to AdminUser.adminId | ID of the administrator who last updated the feedback. |

Export to Sheets

#### ServiceCategory Entity

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Type | Invariant | Description |
| categoryId | String | **PK**, Immutable, GUID/UUID format | Unique identifier for each service category. |
| name | String | **Unique**, Non-null, Min length 3, Max length 100 | Human-readable name of the service category (e.g., "Transportation"). |
| description | String | Non-null, Min length 10, Max length 500 | Detailed description of the category. |
| iconUrl | String | Optional, Valid URL to an icon graphic | URL for a representative icon for the category (for UI display). |
| createdAt | DateTime | Auto-generated, Immutable | Timestamp of category creation. |
| createdByAdminId | String | **FK** to AdminUser.adminId, Non-null | ID of the administrator who created this category. |

Export to Sheets

#### IssueReport Entity

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Type | Invariant | Description |
| reportId | String | **PK**, Immutable, GUID/UUID format | Unique identifier for each issue report. |
| userId | String | **FK** to CitizenUser.userId, Non-null | ID of the citizen who reported the issue. |
| title | String | Optional, Max length 100 | Concise title of the issue. |
| description | String | Non-null, Min length 10, Max length 2000 | Detailed description of the reported issue. |
| reportedAt | DateTime | Auto-generated, Immutable, reportedAt <= current\_timestamp | Timestamp when the issue was reported. |
| status | String | Non-null, Allowed values: OPEN, INVESTIGATING, TRIAGED, RESOLVED, DUPLICATE, CLOSED | Current lifecycle status of the issue report. |
| severity | String | Non-null, Allowed values: LOW, MEDIUM, HIGH, CRITICAL | Perceived impact or urgency of the issue. |
| assignedToAdminId | String | Optional, **FK** to AdminUser.adminId | ID of the administrator currently responsible for the issue. |
| internalNotes | List<Note> | Optional, Embedded array of Note objects | Internal notes added by administrators during issue resolution. |
| lastUpdatedAt | DateTime | Auto-updated | Timestamp of the last update to the issue report. |
| lastUpdatedByAdminId | String | Optional, **FK** to AdminUser.adminId | ID of the administrator who last updated the issue report. |

Export to Sheets

#### Note (Embedded in Feedback/IssueReport)

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Type | Invariant | Description |
| noteId | String | **PK** within list, GUID/UUID format | Unique identifier for the note. |
| content | String | Non-null, Min length 5, Max length 1000 | The text content of the internal note. |
| createdAt | DateTime | Auto-generated, Immutable | Timestamp when the note was created. |
| createdByAdminId | String | **FK** to AdminUser.adminId, Non-null | ID of the administrator who added the note. |

Export to Sheets

### 3.3 Data Governance, Integrity, and Consistency

Robust data governance is fundamental for MyGovInsights Pro's trustworthiness and analytical accuracy.

* **Schema Enforcement:** While MongoDB is schema-less, **schema validation** will be enforced at the application layer and potentially at the database level (MongoDB 3.6+ schema validation rules) to ensure data consistency.
* **Transactional Guarantees:** For operations requiring atomicity across multiple documents or collections (e.g., a feedback submission that also triggers an audit log entry), **MongoDB multi-document transactions** will be utilized to maintain ACID properties.
* **Referential Integrity (Application-Enforced):** While NoSQL databases do not strictly enforce foreign key constraints, the application layer will meticulously manage referential integrity through:
  + **Cascading Deletes/Updates:** Implementing business logic to handle scenarios where related entities are removed (e.g., archiving or anonymizing feedback linked to a deleted user).
  + **Validation on Write:** Ensuring that foreign key references exist before persisting new records.
* **Data Validation Pipelines:** Implement a multi-layered validation strategy:
  + **Client-Side Validation:** Basic UX validation.
  + **API Gateway Validation:** Preliminary schema validation and sanitization.
  + **Service-Level Validation:** Comprehensive business logic validation for all incoming data payloads.
* **Data Versioning (Optional for key entities):** For critical entities like system configurations or service categories, implement versioning to track changes and enable rollbacks.
* **Data Auditability:** Every significant data modification will be recorded in the AuditLog collection, ensuring a complete and immutable history of system actions.

### 3.4 Data Security, Privacy, and Compliance Measures

Adherence to national data protection regulations (e.g., Ethiopia's evolving privacy laws) and international best practices (e.g., GDPR principles) is non-negotiable.

* **Encryption at Rest:**
  + **Disk Encryption:** Underlying infrastructure (VMs, storage volumes) will employ full disk encryption.
  + **Database Encryption:** MongoDB Atlas's native **Transparent Data Encryption (TDE)** will be leveraged to encrypt data files, ensuring that data is encrypted on disk without application-level changes.
  + **Field-Level Encryption (for PII):** For highly sensitive PII (e.g., national ID, detailed location if stored explicitly), application-level **field-level encryption** might be applied using a **Key Management System (KMS)** to manage encryption keys securely.
* **Encryption in Transit:** All network communications will mandate **TLS 1.2+** with strong cipher suites. This applies to:
  + Client-to-API Gateway.
  + API Gateway-to-Microservices.
  + Microservices-to-Database.
  + MyGovInsights-to-External APIs (Fayda, SMS, Email).
* **Data Minimization & Retention:**
  + **Purpose Limitation:** Collect only the data absolutely necessary for the defined purpose.
  + **Storage Limitation:** Define strict data retention policies for all data types. Automatically archive or purge data that is no longer required for operational or legal reasons, applying appropriate **anonymization or pseudonymization** techniques to historical data for analytics.
* **Access Control & Isolation:**
  + **Network Isolation:** Services and databases will reside in separate, segmented subnets within a **Virtual Private Cloud (VPC)**, with strict firewall rules and **Network Security Groups (NSGs)** to control ingress/egress traffic.
  + **Least Privilege Principle:** All service accounts and user roles will be granted only the minimum necessary permissions to perform their designated functions.
* **Secure Backup and Recovery:** Encrypted, automated backups will be performed regularly. A disaster recovery (DR) plan with defined Recovery Point Objectives (RPO) and Recovery Time Objectives (RTO) will be established and regularly tested.
* **Data Sovereignty:** Data will be stored within Ethiopia or in compliant regions, adhering to national data residency requirements.

## 4. Component Design (Modular Breakdown)

The system's modular design ensures clear responsibilities, facilitates independent development, and enhances system maintainability and scalability. Each subsystem is conceived as a self-contained, deployable unit exposing well-defined API contracts.

### 4.1 Citizen Feedback Management Subsystem

* **Purpose:** The core ingestion and lifecycle management unit for citizen feedback. It handles multi-channel input, initial processing, and status transitions of feedback records.
* **API Contracts (External):**
  + POST /api/v1/feedback: Submits new feedback.
  + GET /api/v1/feedback/{id}: Retrieves specific feedback details (if authorized).
* **Inputs:** Raw feedback data (text, category, rating, location, attachments), channel identifier, user context (authenticated/anonymous).
* **Outputs:** Feedback ID, confirmation status, structured Feedback object.
* **Processing Logic (High-Level):**
  1. **Ingestion & Channel Normalization:** Receives raw input from API Gateway. Adapts messages from SMS/USSD/Telegram into a standardized Feedback object structure.
  2. **Schema & Business Validation:** Enforces schema validity, content length, category existence, rating range, and checks for spam/rate limiting signals (communicates with System Configuration Service for rules).
  3. **Content Sanitization:** Cleanses text content to prevent XSS and other injection attacks.
  4. **Attachment Handling:** If present, securely uploads files to Secure Blob Storage and stores their URLs and metadata.
  5. **Persistence:** Saves the validated Feedback record to the MongoDB Database Cluster.
  6. **Event Emission:** Publishes a "FeedbackSubmitted" event to a message broker (e.g., Kafka/RabbitMQ – future state) for asynchronous processing by other services (e.g., Notification Service, Reporting & Analytics Service).
  7. **Response:** Returns a 201 Created status with feedbackId or a 4xx error with detailed messages.
* **Data Access:** Primarily interacts with the Feedback, ServiceCategory, and Attachment collections in the MongoDB Database Cluster. Writes to Secure Blob Storage.
* **Error Handling:** Robust validation error handling, transient error retry logic for storage/database interactions, and graceful degradation for attachment failures.
* **Key Capabilities:**
  1. submitFeedback(payload, channel, userId): Core ingestion method.
  2. validateFeedback(feedbackData): Data quality enforcement.
  3. processAttachments(files): Secure storage and metadata management.
  4. updateFeedbackStatus(feedbackId, newStatus, adminId): Transitions feedback lifecycle.
  5. addInternalNote(feedbackId, noteContent, adminId): Appends administrative notes.

### 4.2 Identity & Access Management (IAM) Subsystem

* **Purpose:** Centralized management of all user identities (citizens, administrators) and enforcement of access control policies across the MyGovInsights platform. It is the gatekeeper for system access.
* **API Contracts (External):**
  + POST /api/v1/auth/register: Registers new citizen accounts.
  + POST /api/v1/auth/login: Authenticates users and issues JWTs.
  + POST /api/v1/auth/verify-fayda: Initiates Fayda ID verification.
  + PUT /api/v1/users/{id}/profile: Updates user profiles.
* **Inputs:** User credentials, registration data, Fayda ID details, profile update requests.
* **Outputs:** JWTs, user/admin profiles, authentication/authorization status.
* **Processing Logic (High-Level):**
  1. **User Registration:**
     + Accepts CitizenUser or AdminUser registration data.
     + Performs strong password hashing (e.g., Bcrypt) and salting.
     + If nationalId is provided, synchronously calls the Fayda National ID API for real-time verification (handles API rate limits, retries).
     + Persists CitizenUser or AdminUser record in the MongoDB Database Cluster.
     + Emits "UserRegistered" event.
  2. **Authentication (Login):**
     + Validates user credentials against stored hashed passwords.
     + Upon successful authentication, generates a **signed JWT** (Access Token) containing userId, roles, and expiry. Optionally, generates a Refresh Token for long-lived sessions.
     + Records login event in AuditLog.
     + Updates lastLoginAt timestamp.
  3. **Authorization:**
     + The IAM service exposes an internal API for other services to query user roles and permissions based on the provided JWT.
     + Role and Attribute-Based Access Control (RBAC/ABAC) policies are applied: e.g., only LOCAL\_ADMIN can update feedback status within their assignedJurisdiction.
  4. **Profile Management:** Facilitates secure updates to user profile attributes.
* **Data Access:** Reads/writes to CitizenUser, AdminUser, AuditLog collections. Integrates with Fayda National ID API.
* **Error Handling:** Manages invalid credentials, duplicate accounts, Fayda API failures (with graceful fallback), and brute-force attack prevention (rate limiting).
* **Key Capabilities:**
  1. registerUser(data, userType): Secure user onboarding.
  2. authenticate(credentials): Core login mechanism.
  3. authorize(userId, action, resource): Fine-grained access control.
  4. verifyFaydaIdentity(nationalId): Orchestrates Fayda API calls.
  5. manageJWTSessions(token): Token validation, revocation, and refresh.

### 4.3 Administrative Operations Subsystem

* **Purpose:** Provides the backend functionality for the administrative dashboard, enabling government staff to manage feedback, issues, and internal operational data according to their roles.
* **API Contracts (External):**
  + GET /api/v1/admin/feedback: Lists feedback with advanced filtering.
  + PUT /api/v1/admin/feedback/{id}/status: Updates feedback status.
  + POST /api/v1/admin/feedback/{id}/notes: Adds internal notes.
  + GET /api/v1/admin/issues: Lists issue reports.
  + PUT /api/v1/admin/issues/{id}/status: Updates issue status.
* **Inputs:** Admin queries (filters, pagination), feedback/issue IDs, status updates, internal note content.
* **Outputs:** Filtered lists of Feedback and IssueReport objects, success/error confirmations.
* **Processing Logic (High-Level):**
  1. **Data Retrieval & Filtering:** Retrieves Feedback and IssueReport data from MongoDB Database Cluster. Applies complex filter criteria (category, status, date range, location, channel) and search terms.
  2. **Role-Based Data Scoping:** Ensures that administrators only access data within their assignedJurisdiction or role scope (e.g., a LOCAL\_ADMIN only sees feedback from their assigned local area).
  3. **Data Enrichment:** Augments feedback/issue data with related user (masked PII) and category information for display on the dashboard.
  4. **Status & Notes Updates:** Receives authenticated requests to update feedback/issue status or add internal notes. Validates the update context (e.g., valid status transition, authorization). Persists changes and logs to AuditLog.
  5. **Event Emission:** Emits "FeedbackStatusUpdated" or "IssueResolved" events to trigger appropriate notifications via the Notification Service.
* **Data Access:** Reads from Feedback, IssueReport, CitizenUser, ServiceCategory, AdminUser collections. Writes to Feedback, IssueReport, AuditLog collections.
* **Error Handling:** Handles authorization failures, data not found, invalid status transitions, and database concurrency issues.
* **Key Capabilities:**
  1. getFilteredFeedback(adminId, filters, pagination): Advanced query and filtering.
  2. updateFeedbackLifecycle(feedbackId, newStatus, adminId): Managed status transitions.
  3. addFeedbackInternalNote(feedbackId, noteContent, adminId): Internal communication.
  4. assignIssue(issueId, adminId): Assigns issues to specific administrators.

### 4.4 Business Intelligence & Reporting Subsystem

* **Purpose:** To provide robust analytical capabilities, including real-time dashboard visualizations and on-demand report generation, transforming raw feedback data into actionable intelligence.
* **API Contracts (External):**
  + GET /api/v1/reports/summary: Retrieves high-level dashboard metrics.
  + GET /api/v1/reports/trends: Retrieves time-series data for trend analysis.
  + GET /api/v1/reports/export/{type}: Exports aggregated data (CSV/JSON).
* **Inputs:** Report type, date ranges, geographical filters, service category filters.
* **Outputs:** Aggregated JSON data for charts, downloadable CSV/JSON report files.
* **Processing Logic (High-Level):**
  1. **Data Aggregation Engine:** Queries Feedback, IssueReport, and Attachment collections. Utilizes MongoDB's aggregation framework for efficient data rollup, statistical computations (e.g., average rating, resolution time), and group-by operations.
  2. **Real-time Dashboard Metrics:** For dashboard views, generates lightweight, fast-to-compute aggregations to support dynamic chart rendering on the client-side.
  3. **Complex Report Generation:** For comprehensive reports (e.g., detailed feedback lists by ward, monthly performance summaries), performs more extensive data processing.
  4. **File Generation & Streaming:** For export, converts processed data into CSV or JSON format and streams the file to the client to avoid memory overflows for large datasets.
  5. **Data Privacy in Reports:** Ensures PII is appropriately masked or anonymized within reports unless explicitly authorized and justified.
* **Data Access:** Reads exclusively from Feedback, IssueReport, ServiceCategory, CitizenUser collections. Minimal interaction with AdminUser for creator information.
* **Error Handling:** Manages query timeouts for large datasets, data processing errors, and file generation failures. Implements asynchronous processing for large reports to prevent timeouts.
* **Key Capabilities:**
  1. getDashboardMetrics(filters): Provides real-time KPIs.
  2. generateFeedbackTrends(filters, granularity): Time-series analysis.
  3. exportData(reportType, filters, format): Handles large data exports.
  4. calculateSentiment(feedbackContent) (Future): Integrates an NLP model for sentiment analysis.

### 4.5 Communication & Notification Subsystem

* **Purpose:** Provides a centralized, robust service for sending automated, timely, and personalized notifications across multiple channels to citizens and administrators based on system events.
* **API Contracts (Internal):**
  + POST /internal/notifications/send: Sends a notification.
* **Inputs:** Recipient ID (userId or adminId), event type, associated data, preferred channel (optional).
* **Outputs:** Status of notification delivery, persisted notification record.
* **Processing Logic (High-Level):**
  1. **Event Subscription & Consumption:** Subscribes to internal events (e.g., "FeedbackSubmitted," "FeedbackStatusUpdated," "UserRegistered") emitted by other services (via message broker – future state, currently direct API call).
  2. **Recipient & Channel Resolution:** Resolves recipient contact details and preferredLanguage from CitizenUser/AdminUser profiles. Determines optimal delivery channel(s) based on user preferences and notification urgency.
  3. **Template Rendering:** Dynamically generates message content by combining predefined, localized templates (managed by System Configuration Service) with event-specific data.
  4. **External Channel Orchestration:**
     + **Email:** Invokes External Email Service API (with retry logic, rate limiting).
     + **SMS:** Invokes External SMS Gateway API (with retry logic, rate limiting).
     + **In-App:** Creates a Notification record in the database for client applications to fetch and display.
  5. **Delivery Confirmation & Logging:** Records the notification attempt, channel, status, and outcome in the Notification collection and AuditLog.
* **Data Access:** Reads from CitizenUser, AdminUser, Notification collections. Writes to Notification and AuditLog. Interacts with External Email Service and External SMS Gateway APIs.
* **Error Handling:** Robust retry mechanisms for external API calls, dead-letter queues for persistent failures, and fallback strategies (e.g., if email fails, try SMS).
* **Key Capabilities:**
  1. triggerEventNotification(eventType, eventPayload): Entry point for other services to request notifications.
  2. renderMessage(templateId, language, data): Dynamic content generation.
  3. dispatchEmail(to, subject, body): Email integration.
  4. dispatchSMS(to, message): SMS integration.
  5. saveInAppNotification(recipientId, message): In-app persistence.

### 4.6 System Configuration & Governance Subsystem

* **Purpose:** Provides centralized control over dynamic system parameters, master data (like service categories), and potentially feature toggles, enabling administrative flexibility without code deployments.
* **API Contracts (External - Admin only):**
  + GET /api/v1/system/categories: Retrieves all service categories.
  + POST /api/v1/system/categories: Creates a new service category.
  + PUT /api/v1/system/categories/{id}: Updates a service category.
  + DELETE /api/v1/system/categories/{id}: Deletes a service category.
  + GET /api/v1/system/settings: Retrieves global system settings.
  + PUT /api/v1/system/settings: Updates global system settings.
* **Inputs:** Configuration keys/values, service category data, admin user updates.
* **Outputs:** Retrieved configurations, updated settings.
* **Processing Logic (High-Level):**
  1. **Configuration Store:** Manages persistent storage of ServiceCategory data and a general SystemConfig collection (key-value pairs) in the MongoDB Database Cluster.
  2. **API Exposure:** Provides secure CRUD APIs for SYSTEM\_ADMIN role to manage these configurations.
  3. **Validation & Audit:** Validates all configuration changes and records them in the AuditLog.
  4. **Runtime Access:** Other services can query this subsystem for current configuration values (e.g., rate limits, notification templates). Implement a caching layer for frequently accessed settings to reduce database load.
* **Data Access:** Reads/writes to ServiceCategory, AdminUser, SystemConfig, AuditLog collections.
* **Error Handling:** Strict input validation, authorization failures, and robust handling of configuration update conflicts.
* **Key Capabilities:**
  1. getServiceCategories(): Provides master data for categories.
  2. updateServiceCategory(id, data): Modifies category details.
  3. getGlobalSetting(key): Retrieves dynamic system parameters.
  4. setGlobalSetting(key, value, adminId): Updates global settings.

### 4.7 API & Integration Gateway Subsystem

* **Purpose:** Acts as the single entry point for all client requests, providing a robust edge layer for security, traffic management, and routing to backend microservices. It also centralizes outbound calls to complex external systems.
* **API Contracts:** All external-facing REST APIs defined in Section 5.3.
* **Inputs:** HTTP requests from all client applications and potentially external partners.
* **Outputs:** HTTP responses, routed requests to internal services, successful/failed external API calls.
* **Processing Logic (High-Level):**
  1. **Request Ingestion & Parsing:** Receives all inbound HTTP requests.
  2. **Rate Limiting:** Applies intelligent, adaptive rate limiting policies (from System Configuration Service) to protect backend services from abuse and DDoS attacks.
  3. **Authentication & Authorization Pre-Check:**
     + Validates JWTs for authenticated requests.
     + Performs initial authorization checks based on request path and method. More granular checks are done at the service level.
     + Handles anonymous requests appropriately.
  4. **Request Routing:** Dynamically routes requests to the appropriate backend microservice based on the request URI and configured routing rules.
  5. **Response Aggregation/Transformation (if applicable):** For complex queries spanning multiple microservices, the Gateway may aggregate responses or perform minor transformations before returning to the client.
  6. **Error Handling & Standardization:** Catches errors from backend services, transforms them into a standardized error response format, and logs gateway-specific errors.
  7. **Circuit Breakers & Timeouts:** Implements circuit breakers and timeouts for calls to backend services and external integrations to prevent cascading failures.
  8. **Outbound External API Orchestration:** Centralizes complex or frequently used external API calls (e.g., Fayda National ID), handling authentication, retry logic, and response mapping, acting as a proxy.
* **Data Access:** Minimal direct data access; primarily interacts with JWT secrets and potentially cache for rate limits.
* **Error Handling:** Critical for system stability. Handles network errors, timeouts, invalid tokens, routing failures, and backpressure.
* **Key Capabilities:**
  1. applyRateLimiting(clientIp, userId): Protects against abuse.
  2. authenticateRequest(authToken): Validates JWT.
  3. routeRequest(path, method): Directs traffic.
  4. handleExternalFaydaCall(payload): Orchestrates Fayda API interaction.
  5. standardizeErrorResponse(error): Consistent error messaging.

## 5. Interface Design

Interface design is paramount for MyGovInsights Pro's usability, interoperability, and long-term sustainability. It encompasses how humans interact with the system and how system components communicate programmatically.

### 5.1 Human-Computer Interaction (HCI) Design Principles

The user interfaces are crafted with a **user-centered design (UCD)** philosophy, prioritizing clarity, efficiency, and inclusivity for all Ethiopian citizens and administrators.

* **Inclusive Design & Accessibility (WCAG 2.1 AA Compliant):**
  + **Multi-language Support (i18n):** Full internationalization for Amharic, Afaan Oromo, Tigrigna, Somali, and English. Language selection will be prominent and persist across sessions.
  + **Dark Mode / High Contrast:** Provides toggleable dark mode and high-contrast options for enhanced readability and reduced eye strain, especially beneficial for users with visual impairments.
  + **Keyboard Navigation & ARIA Attributes:** Ensures full navigability and functionality via keyboard alone. Proper WAI-ARIA (Web Accessibility Initiative - Accessible Rich Internet Applications) attributes will be used for screen reader compatibility.
  + **Legible Typography & Color Contrast:** Adherence to WCAG guidelines for minimum text size, font readability (e.g., Inter font family where possible), and color contrast ratios.
* **Intuitive User Experience (UX):**
  + **Simplicity:** Streamlined workflows and minimal cognitive load, especially for basic feedback submission.
  + **Consistent Navigation:** Standardized navigation patterns across all web-based interfaces (citizen portal, admin dashboard).
  + **Real-time Feedback:** Immediate visual feedback for user actions (e.g., spinners for loading, success messages, validation errors).
  + **Progressive Enhancement:** Core functionality (feedback submission) works even on basic browsers or limited network conditions (especially for PWA).
* **Responsive Design:** Utilizes a mobile-first approach for web and PWA, ensuring optimal layout, performance, and interaction across diverse devices (smartphones, tablets, desktops) and screen orientations.
* **Channel Optimization:**
  + **Web/PWA:** Rich, interactive interfaces with intuitive forms, media upload, and data visualization.
  + **SMS/USSD/Telegram:** Designed for simplicity and efficiency on feature phones, relying on structured text commands and predefined menus. Clear instructions and error messages for these channels.
* **Data Visualization (Admin Dashboard):** Employs interactive charts (e.g., using D3.js, Chart.js, or Recharts components within React) for:
  + Time-series trends (feedback volume, rating evolution).
  + Geographical distribution (heat maps or choropleth maps).
  + Categorical breakdowns (bar charts, pie charts).
  + Key Performance Indicators (KPIs) with drill-down capabilities.

### 5.2 External API Interfaces (MyGovInsights as Consumer)

MyGovInsights Pro integrates with critical external services, consuming their APIs securely and resiliently.

* **Fayda National ID API Integration:**
  + **Purpose:** Programmatic identity verification for citizens.
  + **Integration Pattern:** Synchronous RESTful API calls (HTTPS POST).
  + **Authentication:** Requires secure API keys, potentially client certificates, and/or OAuth 2.0 flows, managed by IAM Subsystem via External Integration Proxy.
  + **Data Exchange:** JSON payloads for requests and responses. Rigorous input/output schema validation.
  + **Resilience:** Implements **exponential backoff with jitter** for retries on transient failures, **circuit breakers** to prevent overloading Fayda's services, and defined timeouts. Fallback mechanisms (e.g., proceed with unverified status or notify user to try later) will be in place.
* **External Communication Service APIs (Email, SMS):**
  + **Purpose:** High-volume, reliable delivery of transactional and informational notifications.
  + **Integration Pattern:** Asynchronous RESTful API calls (HTTPS POST).
  + **Authentication:** API keys or bearer tokens.
  + **Data Exchange:** JSON payloads.
  + **Resilience:** High-volume Notification Subsystem will use internal queues to decouple from external APIs. Implement retry logic and failover to alternative providers if primary fails (e.g., if one SMS gateway is down, try another).
  + **Security:** Ensure sensitive data (e.g., full phone numbers in logs) is masked before transmission to external services where possible.

### 5.3 External API Interfaces (MyGovInsights as Producer)

MyGovInsights Pro exposes a comprehensive set of RESTful APIs, designed for clarity, security, and extensibility.

* **API Design Principles:**
  + **RESTful Compliance:** Adheres to HATEOAS (Hypermedia as the Engine of Application State) where appropriate, resource-oriented URIs, and standard HTTP methods (GET, POST, PUT, DELETE).
  + **Version Control:** All public APIs will be versioned (e.g., /api/v1/) to allow for backward compatibility and graceful evolution.
  + **Clear Contracts:** Defined using OpenAPI/Swagger specifications for auto-generated documentation and client SDKs.
  + **Idempotency:** Operations like feedback submission will be designed to be idempotent to handle network retries gracefully.
* **Base URL:** https://api.mygovinsights.et/{version}
* **Authentication & Authorization:**
  + **JWT-based Authentication:** Bearer tokens (JWTs) issued by the IAM Subsystem for all authenticated API calls.
  + **API Key Authentication:** For integrations with third-party channels (SMS Gateway, USSD provider, Telegram bot webhook), secure API keys will be used, managed by the API & Integration Gateway.
  + **Granular Authorization:** Fine-grained access control enforced by the API Gateway and validated within individual microservices.
* **Data Format:** Exclusively **JSON** for request and response payloads.
* **Error Handling:** Standardized, machine-readable error responses with HTTP status codes (e.g., 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 429 Too Many Requests, 500 Internal Server Error), clear error codes, and descriptive messages.
* **Rate Limiting:** Aggressive, adaptive rate limiting will be applied at the API Gateway to protect against abuse and ensure fair usage.
* **Cross-Origin Resource Sharing (CORS):** Properly configured CORS policies to allow legitimate client applications to interact with the APIs while preventing unauthorized access.

### 5.4 Internal Service Contracts

Internal interfaces between microservices within the Application Tier are equally critical for overall system stability and performance.

* **Lightweight Communication:** Primarily synchronous RESTful API calls via internal network, optimized for speed. Asynchronous event-driven communication (via a message broker) will be introduced for decoupled processes.
* **Internal API Discovery:** Potentially use a service discovery mechanism (e.g., Kubernetes DNS, Consul, Eureka) for services to locate each other dynamically.
* **Standardized Request/Response Formats:** Consistent JSON structures for inter-service communication.
* **Internal Error Handling:** Propagate meaningful internal error codes and messages to facilitate debugging across service boundaries, possibly using a **correlation ID** for tracing requests end-to-end.
* **Shared Libraries:** Common utility functions (e.g., data validation, encryption, logging helpers) will be packaged as internal libraries to promote code reuse and consistency across services.

## 6. Security Design

Security is the **paramount non-functional requirement** for MyGovInsights Pro, treated as a core architectural concern from inception, not an afterthought. A layered, defense-in-depth approach is adopted, adhering to OWASP principles and national cybersecurity guidelines.

### 6.1 Advanced Authentication Strategies

* **Strong Credential Management:**
  + **Password Hashing:** All passwords will be securely hashed using modern, computationally intensive, and salt-aware algorithms (e.g., **Bcrypt or Argon2**) with unique salts per user.
  + **Password Policies:** Enforce strict password complexity (length, character types), uniqueness, and regular rotation for administrators.
* **Multi-Factor Authentication (MFA):** Mandatory for all administrator roles and highly encouraged (optional but promoted) for verified citizens. Supported MFA methods will include SMS One-Time Passwords (OTP) and potentially authenticator apps.
* **Adaptive Authentication (Future):** Implement risk-based authentication, where login attempts from unusual locations, devices, or times trigger additional verification challenges.
* **Session Management:**
  + **Short-Lived Access Tokens (JWTs):** Access tokens will have short expiry times (e.g., 15-30 minutes).
  + **Long-Lived Refresh Tokens:** Refresh tokens, used to obtain new access tokens, will be stored securely (e.g., HttpOnly cookies) and rotated frequently.
  + **Token Revocation:** Mechanisms for immediate token invalidation upon logout, password change, or suspicious activity (e.g., a blacklist/revocation list managed by the IAM service).
* **Fayda National ID Integration:** Secure API calls to the Fayda system for verifiable identity, utilizing mutual TLS (mTLS) or strong API key management with IP whitelisting for inter-system communication.

### 6.2 Robust Authorization Framework (ABAC/RBAC)

MyGovInsights Pro will employ a hybrid authorization model: **Role-Based Access Control (RBAC)** for broad access categories and **Attribute-Based Access Control (ABAC)** for fine-grained, dynamic policy enforcement.

* **RBAC Foundation:** Predefined roles (Citizen, Verified Citizen, Local Admin, Regional Admin, Federal Admin, System Admin) grant base permissions.
* **ABAC for Granularity:** Policies will define access based on attributes of:
  + **User:** Role, assignedJurisdiction, isVerified status.
  + **Resource:** feedbackId, categoryId, location of feedback, status of issue.
  + **Environment:** Time of day, IP address.
  + **Action:** read, update, delete, generate.
  + *Example Policy:* "A LOCAL\_ADMIN can update the status of a Feedback record ONLY IF the Feedback.location is within their assignedJurisdiction AND the Feedback.status is PENDING\_REVIEW."
* **Policy Enforcement Points:**
  + **API Gateway:** Performs initial, coarse-grained authorization based on JWT roles and path.
  + **Microservices (Internal):** Each service performs fine-grained authorization checks based on the authenticated user's context and the specific resource.
* **Auditability:** All authorization decisions, especially failures, will be logged to AuditLog.

### 6.3 Data Protection: Encryption, Masking, and Key Management

* **Encryption at Rest:**
  + **Database Encryption:** Leverage MongoDB Atlas's **Client-Side Field Level Encryption (FLE)** for highly sensitive PII fields in addition to **Transparent Data Encryption (TDE)** at the database layer. This ensures that sensitive data is encrypted before it ever leaves the application process, and only authorized application components can decrypt it.
  + **File Storage Encryption:** Attachments stored in object storage (e.g., S3-compatible) will be encrypted server-side (SSE-S3 or SSE-KMS) and optionally client-side for maximum protection.
* **Encryption in Transit:** All data in transit will be protected by **TLS v1.2+** with modern cipher suites (e.g., AES-256-GCM, ECDHE-RSA-AES256-GCM-SHA384) to prevent eavesdropping and tampering. **HTTP Strict Transport Security (HSTS)** will be enforced to prevent downgrade attacks.
* **Key Management System (KMS):** All encryption keys (for FLE, SSE-KMS, API keys, JWT secrets) will be securely managed and rotated using a cloud-native KMS (e.g., AWS KMS, GCP Cloud KMS, Azure Key Vault). Access to keys will be tightly controlled via IAM policies.
* **Data Masking/Pseudonymization:** For reporting and non-production environments, sensitive PII will be masked, truncated, or pseudonymized to minimize exposure. Only authorized analytics jobs will access raw, unmasked data under strict controls.

### 6.4 Proactive Input Validation & Output Encoding

* **Server-Side Validation:** All inputs from *all* channels (web, PWA, SMS, USSD, Telegram) will be rigorously validated on the server-side against expected formats, types, lengths, and business rules, preventing:
  + **Injection Attacks:** SQL/NoSQL Injection, Command Injection, XSS (Cross-Site Scripting).
  + **Broken Authentication/Authorization:** By ensuring valid input to security controls.
  + **Malicious File Uploads:** Validating file types, sizes, and scanning for malware on attachments.
* **Output Encoding:** All user-supplied data displayed back in the UI (web, admin dashboard) will be contextually output-encoded (e.g., HTML entity encoding, URL encoding) to prevent XSS vulnerabilities.

### 6.5 Secure Session and Token Management

* **Stateless JWTs:** Leverage JWTs for API authentication, ensuring the backend services remain stateless for scalability.
* **Secure Token Storage:** Access Tokens will be handled primarily in memory or short-lived variables. Refresh tokens, if used, will be stored in HttpOnly and Secure cookies (for web) to prevent client-side JavaScript access.
* **Token Revocation:** Implement a robust mechanism to immediately revoke compromised JWTs (e.g., by maintaining a blacklist/revocation list or checking validity with the IAM service on every request).
* **CSRF Protection:** Anti-CSRF (Cross-Site Request Forgery) tokens will be used for state-changing HTTP methods (POST, PUT, DELETE) on the web application.

### 6.6 Comprehensive Security Logging and Threat Detection

* **Granular Audit Logging:** Implement comprehensive, immutable audit logging for all security-relevant events, including:
  + All authentication attempts (success/failure), including MFA status.
  + All authorization decisions (permit/deny).
  + CRUD operations on sensitive data (Feedback, Users, Admin accounts, System Configuration).
  + Changes to security configurations, roles, or policies.
  + Unusual access patterns or repeated failed attempts.
* **Centralized SIEM Integration:** All logs will be ingested into a centralized Security Information and Event Management (SIEM) system for real-time analysis, correlation, and anomaly detection.
* **Automated Threat Detection:** Implement rules and machine learning models within the SIEM to detect suspicious activities (e.g., brute-force attacks, privilege escalation attempts, unusual data access patterns) and trigger immediate alerts.

### 6.7 Continuous Vulnerability Management & DevSecOps

* **Shift-Left Security:** Integrate security practices throughout the entire Software Development Life Cycle (SDLC), from design (threat modeling) to coding (SAST/DAST) and deployment.
* **Automated Security Testing (SAST/DAST):**
  + **Static Application Security Testing (SAST):** Integrate SAST tools into the CI pipeline to analyze source code for common vulnerabilities (e.g., injection flaws, insecure configurations) before deployment.
  + **Dynamic Application Security Testing (DAST):** Implement DAST tools to scan the running application in testing environments for runtime vulnerabilities.
* **Dependency Scanning:** Continuously scan third-party libraries and dependencies for known vulnerabilities (CVEs) and enforce remediation or upgrades.
* **Regular Penetration Testing:** Engage independent security firms for periodic penetration testing to identify zero-day vulnerabilities or complex attack vectors.
* **Bug Bounty Program (Future):** Establish a bug bounty program to leverage the ethical hacker community for continuous vulnerability discovery.

## 7. Performance and Scalability Design

MyGovInsights Pro is designed for **high performance and elastic scalability** to accommodate a potentially massive number of concurrent users and feedback submissions, especially during peak civic engagement periods.

### 7.1 Scalability and Elasticity Strategies

* **Horizontal Scaling (Stateless Microservices):**
  + All backend microservices are designed to be stateless (or near-stateless). This allows new instances to be spun up or down rapidly based on demand, distributing load efficiently across multiple compute resources.
  + Automated **auto-scaling groups** (e.g., in cloud environments like AWS EC2 Auto Scaling, Kubernetes Horizontal Pod Autoscaler) will dynamically adjust the number of service instances.
* **Containerization (Docker & Kubernetes):** Packaging services as Docker containers and orchestrating them with Kubernetes provides:
  + **Portability:** Consistent runtime environments across development, staging, and production.
  + **Resource Isolation:** Containers provide resource isolation, preventing noisy neighbors.
  + **Self-Healing:** Kubernetes automatically restarts failed containers and replaces unhealthy ones.
  + **Efficient Resource Utilization:** Optimized container images and resource limits.
* **Database Sharding & Replication:**
  + **Replica Sets (MongoDB):** Deploy MongoDB in a replica set configuration for high availability, fault tolerance, and read scalability (reads can be distributed across secondary nodes).
  + **Sharding (MongoDB):** For extreme data volume and throughput requirements, MongoDB will be configured for horizontal partitioning (sharding) across multiple clusters, distributing data and query load. This is essential for a national-scale database.
* **Asynchronous Processing:** Critical, non-real-time operations (e.g., heavy report generation, complex notifications) will be offloaded to **asynchronous message queues** and processed by dedicated worker services to prevent blocking core API threads and improve responsiveness.

### 7.2 Advanced Load Management

* **Intelligent Load Balancing:**
  + **Layer 7 (Application) Load Balancing:** The API Gateway will use HTTP/S-aware load balancing (e.g., Nginx, cloud ALB) to distribute traffic based on request attributes (e.g., URL path, HTTP headers), enabling fine-grained control and routing.
  + **Health Checks:** Load balancers will continuously monitor the health of backend service instances and automatically remove unhealthy ones from rotation.
* **Connection Pooling:** Maintain efficient connection pools to databases and external APIs to minimize connection overhead and maximize resource reuse.

### 7.3 Intelligent Caching Mechanisms

* **Distributed Caching (e.g., Redis):**
  + Cache frequently accessed, relatively static data (e.g., ServiceCategory lists, system configuration values, popular dashboard metrics) in an in-memory distributed cache (e.g., Redis cluster).
  + Implement **cache invalidation strategies** (e.g., time-based expiry, event-driven invalidation) to ensure data freshness.
* **Content Delivery Network (CDN):** Utilize a CDN for serving static assets (HTML, CSS, JavaScript bundles, images, icons) of the web portal and PWA. This reduces latency for end-users by serving content from edge locations geographically closer to them and offloads traffic from the main backend.
* **Browser Caching:** Leverage HTTP caching headers (Cache-Control, ETag) for client-side assets to minimize repeated downloads.

### 7.4 Database Performance Optimization

* **Strategic Indexing:** Apply compound and multi-key indexes strategically to MongoDB collections based on common query patterns identified during design and early development. Regularly review and optimize indexes based on query performance analytics.
* **Query Optimization & Profiling:**
  + Continuously monitor and profile slow queries in the production database.
  + Optimize query structures and use appropriate MongoDB operators to minimize data scanned and operations performed.
* **Data Model Optimization:** Design the MongoDB schema for efficient read/write access patterns, considering embedding versus referencing documents based on data relationships and access frequency.
* **Connection Management:** Implement robust connection pooling for database clients within each microservice to manage database connections efficiently and prevent resource exhaustion.

### 7.5 Application Code Efficiency and Profiling

* **Lean Codebase:** Write clean, optimized, and performant code in Node.js. Avoid synchronous I/O operations where asynchronous alternatives exist.
* **Resource-Efficient Libraries:** Carefully select libraries and frameworks, preferring those known for performance and low resource footprint.
* **Code Profiling:** Use Node.js profiling tools (e.g., perf\_hooks, Clinic.js) to identify CPU bottlenecks, memory leaks, and inefficient code segments during development and testing.
* **Garbage Collection Tuning:** Monitor and tune Node.js Garbage Collection (GC) parameters to minimize GC pauses and improve application responsiveness under load.

## 8. Deployment and Operations Design

The deployment and operations strategy for MyGovInsights Pro is founded on **Site Reliability Engineering (SRE)** principles, emphasizing automation, observability, and proactive management to ensure maximum uptime and operational efficiency.

### 8.1 Production Environment Architecture

* **Cloud-Native Deployment:** Fully leverages a leading public cloud provider (e.g., AWS, Google Cloud Platform, Microsoft Azure) for its elasticity, managed services, and global footprint.
* **Infrastructure as Code (IaC):** All infrastructure components (VPCs, subnets, load balancers, Kubernetes clusters, database instances, security groups) will be defined, provisioned, and managed using IaC tools (e.g., Terraform, CloudFormation). This ensures consistency, repeatability, and version control of the infrastructure.
* **Container Orchestration:** **Kubernetes (K8s)** will be the chosen platform for container orchestration, providing:
  + **Automated Deployment & Scaling:** Manages deployment, scaling, and self-healing of microservice pods.
  + **Service Discovery & Load Balancing:** Built-in mechanisms for inter-service communication.
  + **Resource Management:** Efficient allocation and management of compute resources.
* **Managed Database Service:** MongoDB Atlas (or equivalent managed service) will be used for the database layer, offloading operational burdens like patching, backups, and scaling from the operations team.
* **Secure Network Segmentation:** A multi-tiered VPC architecture with private and public subnets, rigorous Network ACLs, and Security Groups to isolate application components and control traffic flow.
* **Regional Deployment:** Initially deployed in one region with high availability zones. Future expansion may involve multi-region deployments for global resilience and disaster recovery (DR).

### 8.2 Release Management and Deployment Strategies

A disciplined and automated approach to software releases minimizes risk and maximizes deployment velocity.

* **GitOps Workflow:** Configuration and deployment managed via Git, where infrastructure and application changes are committed as code, triggering automated pipelines.
* **Automated Build & Test:** All code changes trigger automated builds and comprehensive test suites (unit, integration, end-to-end) within the CI pipeline.
* **Progressive Delivery:**
  + **Blue/Green Deployments:** For major releases, deploy the new version (green) alongside the current version (blue). Once green is validated via automated and manual tests, traffic is shifted. This allows for instant rollback.
  + **Canary Releases:** For smaller, less impactful changes, gradually roll out the new version to a small subset of users or servers, closely monitoring performance and errors before a full rollout. This minimizes impact of regressions.
* **Rollback Capability:** Automated rollback mechanisms will be in place for every deployment, enabling rapid restoration to a stable previous state in case of critical issues.
* **Version Control:** Every deployed artifact (container image, configuration) will be tagged and versioned.

### 8.3 Automated Continuous Delivery Pipeline (CI/CD)

A robust CI/CD pipeline (e.g., Jenkins, GitLab CI/CD, GitHub Actions) is central to efficient software delivery.

* **Continuous Integration (CI):**
  + **Automated Builds:** Every code commit triggers an automated build process.
  + **Static Code Analysis:** Integration of SAST tools (e.g., SonarQube, Snyk) to identify code quality issues, security vulnerabilities, and adherence to coding standards.
  + **Unit & Integration Tests:** Automated execution of all unit and integration tests.
  + **Container Image Building:** Automated creation and tagging of Docker images for each microservice.
  + **Artifact Management:** Secure storage of build artifacts (container images) in a private container registry.
* **Continuous Deployment (CD):**
  + **Automated Deployment to Staging:** Successfully built and tested artifacts are automatically deployed to a staging environment for further testing.
  + **Automated End-to-End Tests:** Comprehensive automated system tests run against the staging environment.
  + **Gated Deployments:** Manual approval gates for deploying to production, triggered only after all automated tests pass and staging environment is validated.
  + **Production Deployment:** Automated deployment to production using Blue/Green or Canary strategies.
* **Pipeline Security:** The CI/CD pipeline itself will be secured, with strict access controls, secret management, and auditable logs.

### 8.4 Comprehensive Observability: Monitoring, Logging, and Alerting

MyGovInsights Pro will be instrumented for deep observability, allowing operations teams to understand the system's internal state from its external outputs.

* **Unified Monitoring (Metrics):**
  + **Infrastructure Metrics:** Monitor CPU, memory, disk I/O, network throughput for all compute instances, database clusters, and storage.
  + **Application Metrics:** Collect key performance indicators (KPIs) like request rates, error rates, latency (at API Gateway and per service), throughput, queue depths, and business-specific metrics (e.g., feedback submission rate, report generation time).
  + **Custom Dashboards:** Build customized dashboards (e.g., Grafana, cloud-native dashboards) for different stakeholders (developers, operations, business analysts) to visualize system health and performance.
* **Centralized Logging (Structured Logs):**
  + All application and infrastructure logs will be collected, aggregated, and stored in a centralized, searchable logging platform (e.g., ELK Stack - Elasticsearch, Logstash, Kibana; Splunk; or cloud-native solutions).
  + Logs will be **structured (JSON)** for easy parsing, querying, and analysis.
  + **Correlation IDs:** Implement a request tracing mechanism (e.g., OpenTracing, OpenTelemetry) by propagating a unique correlation ID across all services for a given transaction, enabling end-to-end debugging.
* **Proactive Alerting:**
  + **Threshold-Based Alerts:** Configure alerts for critical thresholds (e.g., error rate > 5%, latency > 500ms, low disk space, service instance crashes).
  + **Anomaly Detection:** Utilize machine learning capabilities (if supported by monitoring platform) to detect unusual patterns that might indicate emerging issues.
  + **Notification Channels:** Alerts will be routed to an on-call rotation system, email, SMS, and dedicated chat channels.
  + **Alert Fatigue Mitigation:** Implement alert deduplication, grouping, and escalation policies to prevent alert fatigue among operations teams.

## 9. Resilience and Error Handling Design

Building a resilient system that can gracefully handle failures and provide clear diagnostic information is critical for MyGovInsights Pro's operational stability and user trust.

### 9.1 Fault Tolerance and Graceful Degradation

* **Redundancy:** All critical components (compute instances, database replicas, load balancers) will be deployed with N+1 redundancy across multiple Availability Zones within a region to protect against single points of failure.
* **Circuit Breakers:** Implement circuit breaker patterns (e.g., using libraries like Polly for Node.js) for inter-service communication and calls to external APIs. This prevents cascading failures by "breaking" the circuit to a failing service after a threshold, allowing it to recover without overwhelming it.
* **Bulkheads:** Partition resources (e.g., thread pools, connection pools) for different services or external dependencies. A failure or performance degradation in one dependency will not consume all resources and affect other, healthy parts of the system.
* **Timeouts and Retries:** Configure sensible timeouts for all network operations and implement idempotent retry logic with exponential backoff for transient errors.
* **Graceful Degradation:** Design the system to gracefully degrade functionality rather than completely fail. For example, if the Fayda ID verification service is unavailable, allow citizen registration to proceed with a "pending verification" status, and notify the user about the delay.
* **Idempotent Operations:** All critical write operations (e.g., feedback submission) will be designed to be idempotent, meaning performing the operation multiple times produces the same result as performing it once. This is crucial for safe retries.

### 9.2 Centralized Error Management and Traceability

* **Global Error Handling Middleware:** Implement a robust global error handling middleware at the API Gateway and within each microservice to catch uncaught exceptions and transform them into standardized, secure error responses.
* **Standardized Error Payload:** All API error responses will adhere to a consistent JSON format, including:
  + errorCode (a unique, internal identifier for the error type).
  + message (a human-readable description, potentially localized).
  + details (optional, for specific validation errors).
  + traceId (a unique request identifier for end-to-end tracing).
* **Correlation IDs for Distributed Tracing:** Every incoming request to the API Gateway will be assigned a unique correlationId. This ID will be propagated through all downstream microservice calls and logs, enabling comprehensive tracing of a single request across the distributed system for debugging and root cause analysis.
* **Error Boundaries (Frontend):** For React applications, utilize error boundaries to gracefully handle UI rendering errors without crashing the entire application, providing a more robust user experience.

### 9.3 Proactive Logging and Auditing

* **Structured, Contextual Logging:** All logs will be generated in a structured format (JSON) and include rich contextual metadata:
  + timestamp, level, serviceName, correlationId, userId, adminId, resourceId, httpMethod, endpoint, requestDuration, etc.
  + This enables powerful querying and analysis in the centralized logging system.
* **Asynchronous Logging:** Logging operations will be asynchronous to avoid impacting application performance.
* **Log Retention Policy:** Define strict retention policies for logs based on compliance requirements and operational needs, balancing cost with diagnostic capability. Critical audit logs will have longer retention.
* **Security Logging:** As detailed in Section 6.6, a dedicated stream of security-relevant events will feed into the SIEM for real-time threat detection.

### 9.4 Intelligent Alerting and Incident Management

* **Metrics-Based Alerts:** Configure alerts on deviations from baseline metrics (e.g., latency spikes, error rate increases, CPU saturation, database connection pool exhaustion).
* **Log-Based Alerts:** Set up alerts based on specific log patterns (e.g., repeated authentication failures, critical error messages, security events).
* **Escalation Policies:** Define clear escalation paths for alerts, ensuring the right team members are notified based on severity and time of day.
* **Playbooks/Runbooks:** Develop detailed runbooks for common incidents, guiding operations teams through troubleshooting and resolution steps.
* **Post-Mortems (Blameless):** Conduct blameless post-mortems for all major incidents to identify root causes, learn from failures, and implement preventative measures.

## 10. Quality Assurance & Testing Strategy

A holistic and continuous Quality Assurance (QA) strategy is embedded throughout the SDLC of MyGovInsights Pro, ensuring a high-quality, reliable, and secure product. The approach emphasizes automation and early defect detection ("shift-left").

### 10.1 Multi-tiered Testing Approach

* **Unit Testing:**
  + **Scope:** Verifies the smallest testable units of code (functions, methods) in isolation.
  + **Automation:** Mandatory for all new code. Integrated into the CI pipeline.
  + **Coverage:** Target high code coverage (e.g., 80%+) for critical business logic.
  + **Frameworks:** Jest (Node.js/React), Mocha/Chai.
* **Component Testing:**
  + **Scope:** Tests individual microservices in isolation, simulating their dependencies (e.g., mocking database, external APIs).
  + **Purpose:** Ensures the service's internal logic and API contract are correctly implemented.
  + **Automation:** Integral part of CI pipeline.
* **Integration Testing:**
  + **Scope:** Verifies the interactions between interconnected microservices, database, and external APIs in a dedicated integration environment.
  + **Contract Testing:** Implement consumer-driven contract testing (e.g., Pact) between services to ensure API compatibility without full end-to-end deployments.
  + **Automation:** Automated execution in the CI/CD pipeline after component tests pass.
* **End-to-End (E2E) System Testing:**
  + **Scope:** Simulates real user scenarios across the entire system, from client UI to backend services and external integrations.
  + **Automation:** Crucial for regression testing of core user journeys.
  + **Frameworks:** Cypress, Playwright, Selenium for UI automation; Postman/Newman for API E2E tests.
  + **Environment:** Executed against a fully deployed staging environment that mirrors production.

### 10.2 Performance and Scalability Testing

* **Load Testing:** Simulates anticipated peak user loads (concurrent users, requests per second) to assess system performance (latency, throughput, resource utilization) under expected conditions.
* **Stress Testing:** Pushes the system beyond its expected limits to identify breaking points, bottlenecks, and determine the system's resilience under extreme conditions.
* **Scalability Testing:** Measures how the system scales (e.g., linearly) when resources (e.g., service instances, database nodes) are added.
* **Endurance/Soak Testing:** Runs the system under a sustained, moderate load for extended periods to detect memory leaks, resource exhaustion, or other degradation over time.
* **Tools:** Apache JMeter, k6, Locust.
* **Environment:** Dedicated performance testing environment with production-like configuration and data volume.

### 10.3 Comprehensive Security Assurance

* **Threat Modeling:** Conduct regular threat modeling exercises (e.g., STRIDE, DREAD) early in the design phase to identify potential security vulnerabilities and incorporate preventative controls.
* **Static Application Security Testing (SAST):** Automated code analysis tools integrated into the CI pipeline to identify security flaws (e.g., injection flaws, insecure configurations) in the source code.
* **Dynamic Application Security Testing (DAST):** Automated tools that test the running application (in staging) for vulnerabilities by simulating external attacks (e.g., OWASP ZAP, Burp Suite Enterprise).
* **Software Composition Analysis (SCA):** Regularly scan third-party libraries and dependencies for known vulnerabilities (CVEs) and enforce remediation.
* **Penetration Testing (External):** Engage certified ethical hackers and security audit firms for independent, black-box, and white-box penetration testing of the deployed system.
* **Security Configuration Review:** Automated and manual review of infrastructure and application configurations for security best practices (e.g., firewall rules, IAM policies, environment variables).

### 10.4 User Acceptance and Usability Validation

* **UAT Phases:** Conduct structured UAT phases with representatives from citizens (pilot groups) and all tiers of government administrators.
* **Scenario-Based Testing:** Users execute real-world scenarios and business workflows to validate that the system meets their operational needs and expectations.
* **Usability Testing:** Observe users interacting with the system to identify pain points, confusing elements, and areas for UX improvement. Incorporate feedback loops for iterative design refinements.
* **Accessibility Audits:** Conduct manual and automated accessibility audits against WCAG 2.1 AA guidelines.

### 10.5 Accessibility and Localization Testing

* **Accessibility Testing:** Dedicated testing to ensure the UI is usable by individuals with disabilities, using assistive technologies (screen readers) and keyboard-only navigation.
* **Localization (L10n) Testing:** Verify that all UI elements, messages, and content are correctly translated and displayed in all supported languages (Amharic, Afaan Oromo, Tigrigna, Somali, English), including proper text directionality and cultural nuances. This includes testing data formatting for dates, numbers, and currency.

## 11. Appendices

### 11.1 High-Level Component Interaction Diagram

This diagram provides a macroscopic view of how the major subsystems (logical components) of MyGovInsights Pro interact to fulfill system functionalities.

<div align="center"> <h4>Fig 6. High-Level Backend Component Interaction Diagram</h4> <p>This diagram illustrates the primary communication paths and dependencies between the core microservices within the MyGovInsights Pro backend.</p> </div>

High-Level Backend Component Interaction Diagram

Aug 15, 11:34 PM

Open

graph TD

subgraph Client Applications

C1[Citizen Web Portal]

C2[Citizen PWA]

C3[SMS/USSD Gateway]

C4[Telegram Bot]

C5[Admin Dashboard UI]

end

subgraph API Gateway & Edge Services

AG[API Gateway]

end

subgraph Core Backend Microservices

IAM[Identity & Access Management Service]

FMS[Feedback Management Service]

RAS[Reporting & Analytics Service]

NOS[Notification Service]

AOS[Administrative Operations Service]

SCS[System Configuration Service]

end

subgraph Data Persistence

DB[MongoDB Database Cluster]

FS[Secure File Storage]

end

subgraph External Systems

FNI[Fayda National ID API]

ESC[External SMS/Email Services]

end

C1 --> AG

C2 --> AG

C3 --> AG

C4 --> AG

C5 --> AG

AG --> IAM

AG --> FMS

AG --> RAS

AG --> AOS

AG --> SCS

IAM --> DB

IAM --> FNI

FMS --> DB

FMS --> FS

FMS --> NOS

RAS --> DB

NOS --> DB

NOS --> ESC

AOS --> DB

AOS --> FMS

AOS --> RAS

AOS --> NOS

AOS --> SCS

SCS --> DB

style AG fill:#f9f,stroke:#333,stroke-width:2px

style IAM fill:#ccf,stroke:#333,stroke-width:2px

style FMS fill:#bbf,stroke:#333,stroke-width:2px

style RAS fill:#9cf,stroke:#333,stroke-width:2px

style NOS fill:#7cf,stroke:#333,stroke-width:2px

style AOS fill:#e6c,stroke:#333,stroke-width:2px

style SCS fill:#ffd,stroke:#333,stroke-width:2px

style DB fill:#ffb,stroke:#333,stroke-width:2px

style FS fill:#fbc,stroke:#333,stroke-width:2px

style FNI fill:#ffd,stroke:#333,stroke-width:2px

style ESC fill:#efe,stroke:#333,stroke-width:2px

### 11.2 Core Domain Class Model

This diagram represents the logical schema of the core data entities and their relationships, forming the persistent data foundation for MyGovInsights Pro. It emphasizes key attributes and cardinalities.

<div align="center"> <h4>Fig 7. Core Domain Class Model (Detail)</h4> <p>This detailed class diagram outlines the essential entities, their attributes, and relationships, reflecting the system's data integrity requirements.</p> </div>

Core Domain Class Model

Aug 15, 11:34 PM

Open

classDiagram

class CitizenUser {

+String userId

-String firstName

-String lastName

-String email \*UNIQUE\*

-String phoneNumber \*UNIQUE\*

-String passwordHash

+Boolean isVerified

-String nationalId \*UNIQUE, OPTIONAL\*

+DateTime createdAt

+String preferredLanguage

+List~String~ roles (e.g., ["citizen", "verified"])

+List~Feedback~ getFeedbackHistory()

+updateProfile(data): void

}

class AdminUser {

+String adminId

-String username \*UNIQUE\*

-String email \*UNIQUE\*

-String passwordHash

+String role (enum: LOCAL\_ADMIN, REGIONAL\_ADMIN, FEDERAL\_ADMIN, SYSTEM\_ADMIN)

+DateTime lastLoginAt

+String assignedJurisdiction (optional)

+manageFeedback(Feedback): void

+generateReports(criteria): Report

+updateSystemConfig(config): void

+updateAdminAccount(data): void

}

class ServiceCategory {

+String categoryId

+String name \*UNIQUE\*

+String description

+String iconUrl (optional)

+DateTime createdAt

+String createdByAdminId FK

+createCategory(name, desc): ServiceCategory

+updateCategory(id, name, desc): void

+deleteCategory(id): void

}

class Feedback {

+String feedbackId

+String userId FK

+String categoryId FK

+String title (optional)

+String content

+Integer rating (1-5, optional)

+DateTime submittedAt

+String currentStatus (enum: PENDING\_REVIEW, IN\_PROGRESS, RESOLVED, CLOSED, ARCHIVED)

+String channel (enum: WEB, PWA, SMS, USSD, TELEGRAM)

+String location (GeoJSON, optional)

+List~String~ attachmentIds FK

+List~Note~ internalNotes

+DateTime lastUpdatedAt

+String lastUpdatedByAdminId FK (optional)

+submit(data): Feedback

+getDetails(): Feedback

+updateStatus(newStatus, adminId): void

+addNote(note, adminId): void

}

class IssueReport {

+String reportId

+String userId FK

+String title (optional)

+String description

+DateTime reportedAt

+String status (enum: OPEN, INVESTIGATING, TRIAGED, RESOLVED, DUPLICATE, CLOSED)

+String severity (enum: LOW, MEDIUM, HIGH, CRITICAL)

+String assignedToAdminId FK (optional)

+List~Note~ internalNotes

+DateTime lastUpdatedAt

+String lastUpdatedByAdminId FK (optional)

+createReport(description, userId, severity): IssueReport

+resolveReport(reportId, adminId): void

+assign(adminId): void

+getReports(filters): List~IssueReport~

}

class Attachment {

+String attachmentId

+String feedbackId FK

+String fileName

+String fileType

+String fileUrl (secure cloud URL)

+DateTime uploadedAt

+String uploadedByUserId FK

+uploadFile(fileData, feedbackId, userId): Attachment

+deleteFile(attachmentId): void

}

class Notification {

+String notificationId

+String recipientId FK

+String message

+DateTime sentAt

+Boolean isRead

+String type (enum: FEEDBACK\_UPDATE, ISSUE\_ASSIGNED, SYSTEM\_ALERT, WELCOME)

+String channel (enum: EMAIL, SMS, IN\_APP)

+String relatedEntityId (optional)

+String relatedEntityType (optional)

+send(recipient, message, type, channel): void

+markAsRead(notificationId): void

+getNotifications(recipientId, isRead): List~Notification~

}

class AuditLog {

+String logId

+String actorId (userId or adminId)

+String actorType (enum: CITIZEN, ADMIN, SYSTEM)

+String actionType (enum: FEEDBACK\_SUBMITTED, STATUS\_UPDATED, LOGIN\_SUCCESS, CONFIG\_CHANGE)

+String entityType (enum: FEEDBACK, USER, CATEGORY, SYSTEM\_SETTING, ISSUE\_REPORT)

+String entityId (optional)

+Map~String, Any~ details (JSON)

+DateTime timestamp

+recordEvent(actorId, actorType, action, entityType, entityId, details): void

}

class SystemConfig {

+String configKey PK

+String configValue (JSON/String)

+DateTime lastUpdated

+String lastUpdatedByAdminId FK (optional)

+getSetting(key): Any

+setSetting(key, value, adminId): void

}

CitizenUser "1" -- "0..\*" Feedback : submits

CitizenUser "1" -- "0..\*" IssueReport : reports

ServiceCategory "1" -- "0..\*" Feedback : categorized\_under

Feedback "1" -- "0..\*" Attachment : has

Feedback "1" -- "0..\*" Note : includes

IssueReport "1" -- "0..\*" Note : includes

AdminUser "1" -- "0..\*" Feedback : manages

AdminUser "1" -- "0..\*" IssueReport : manages/assigns

AdminUser "1" -- "0..\*" ServiceCategory : administers

AdminUser "1" -- "0..\*" AdminUser : manages

AdminUser "1" -- "0..\*" SystemConfig : configures

AdminUser "1" -- "0..\*" Note : creates

Notification "0..\*" -- "1" CitizenUser : targets

Notification "0..\*" -- "1" AdminUser : targets

AuditLog "0..\*" -- "1" CitizenUser : relates\_to

AuditLog "0..\*" -- "1" AdminUser : relates\_to

### 11.3 Key System Interaction Sequence Diagrams

These sequence diagrams provide a detailed, step-by-step visualization of the interactions between different services and components for critical user flows, highlighting both synchronous and asynchronous communication.

#### Fig 8. Sequence Diagram: Citizen Registration (with Fayda Integration)

<div align="center"> <p>Illustrates the secure, detailed flow for a new citizen registering, including optional Fayda National ID verification, ensuring robust identity management.</p> </div>

Sequence Diagram: Citizen Registration (with Fayda Integration)

Aug 15, 11:34 PM

Open

sequenceDiagram

actor Citizen

participant ClientApp as "Citizen App (Web/PWA)"

participant APIGateway as "API Gateway"

participant IAMService as "Identity & Access Management Service"

participant FaydaAPI as "Fayda National ID API"

participant Database as "MongoDB Cluster"

participant SCS as "System Config Service"

participant AOS as "Admin Operations Service"

participant NOS as "Notification Service"

participant AuditLog as "Audit Log Service"

Citizen->>ClientApp: Enters registration details (email, password, phone, optional Fayda ID)

ClientApp->>APIGateway: POST /auth/register (RegistrationData)

APIGateway->>APIGateway: Validate basic schema, Rate Limit

APIGateway->>IAMService: Forward Request (User Registration)

IAMService->>Database: Check for existing email/phone

Database-->>IAMService: Result (e.g., User Not Found)

alt Fayda ID Provided & System Enabled

IAMService->>SCS: Get Fayda API configuration

SCS-->>IAMService: Fayda Config (URL, API Key)

IAMService->>FaydaAPI: Verify National ID (NationalID, Name)

FaydaAPI-->>IAMService: Verification Result (Status, Data)

alt Fayda Verification Fails

IAMService-->>APIGateway: Error Response (400 Bad Request: Invalid Fayda ID)

APIGateway-->>ClientApp: Error

ClientApp->>Citizen: "Registration Failed: Invalid National ID"

return

end

end

IAMService->>IAMService: Hash Password (Bcrypt/Argon2)

IAMService->>Database: Save new CitizenUser record (isVerified based on Fayda result)

Database-->>IAMService: Confirmation (userId)

IAMService->>AuditLog: Record "USER\_REGISTERED" event

AuditLog-->>IAMService: Log Confirmation

IAMService->>NOS: Request Welcome Notification (userId, preferredLanguage)

NOS-->>IAMService: Notification Status

IAMService-->>APIGateway: Success Response (201 Created, userId)

APIGateway-->>ClientApp: Success

ClientApp->>Citizen: "Registration Successful! Welcome to MyGovInsights!"

#### Fig 9. Sequence Diagram: Citizen Login

<div align="center"> <p>Details the secure authentication process for citizens, including JWT issuance and session management.</p> </div>

Sequence Diagram: Citizen Login

Aug 15, 11:34 PM

Open

sequenceDiagram

actor Citizen

participant ClientApp as "Citizen App (Web/PWA)"

participant APIGateway as "API Gateway"

participant IAMService as "Identity & Access Management Service"

participant Database as "MongoDB Cluster"

participant AuditLog as "Audit Log Service"

Citizen->>ClientApp: Enters Login Credentials (email, password)

ClientApp->>APIGateway: POST /auth/login (Credentials)

APIGateway->>APIGateway: Basic Schema Validation, Rate Limit

APIGateway->>IAMService: Forward Request (Login)

IAMService->>Database: Retrieve CitizenUser by Email

Database-->>IAMService: User Record (or not found)

alt User Found & Password Matches

IAMService->>IAMService: Compare Password (Hashed)

IAMService->>IAMService: Generate JWT (Access Token with userId, roles)

IAMService->>Database: Update lastLoginAt for CitizenUser

Database-->>IAMService: Update Confirmation

IAMService->>AuditLog: Record "LOGIN\_SUCCESS" event

AuditLog-->>IAMService: Log Confirmation

IAMService-->>APIGateway: Success Response (200 OK, JWT)

APIGateway-->>ClientApp: Success

ClientApp->>ClientApp: Store JWT securely (e.g., Local Storage/HttpOnly Cookie)

ClientApp->>Citizen: Redirect to Dashboard/Home Page

else User Not Found or Password Mismatch

IAMService->>AuditLog: Record "LOGIN\_FAILURE" event

AuditLog-->>IAMService: Log Confirmation

IAMService-->>APIGateway: Error Response (401 Unauthorized)

APIGateway-->>ClientApp: Error

ClientApp->>Citizen: "Login Failed: Invalid credentials or account not found."

end

#### Fig 10. Sequence Diagram: Submit Feedback (Multi-Channel)

<div align="center"> <p>Outlines the resilient process for submitting feedback across various channels, including validation, storage, and asynchronous notifications.</p> </div>

Sequence Diagram: Submit Feedback (Multi-Channel)

Aug 15, 11:34 PM

Open

sequenceDiagram

actor Citizen

participant ClientChannel as "Client App (Web/PWA/SMS/USSD/TG)"

participant APIGateway as "API Gateway"

participant FMS as "Feedback Management Service"

participant SCS as "System Config Service"

participant Database as "MongoDB Cluster"

participant FS as "Secure File Storage"

participant NOS as "Notification Service"

participant AuditLog as "Audit Log Service"

Citizen->>ClientChannel: Provides Feedback (content, category, rating, optional attachment)

ClientChannel->>APIGateway: POST /feedback (FeedbackData, Channel, UserContext/Anonymous)

APIGateway->>APIGateway: Authenticate/Identify User, Rate Limit, Basic Schema Validate

APIGateway->>FMS: Forward Request

FMS->>SCS: Get Service Categories (for validation)

SCS-->>FMS: Categories List

FMS->>FMS: Validate Feedback Data (content, category, rating, location)

alt Invalid Feedback Data

FMS-->>APIGateway: Error Response (400 Bad Request)

APIGateway-->>ClientChannel: Error

ClientChannel->>Citizen: "Feedback Submission Failed: [Validation Error]"

return

end

opt If Attachment Included

FMS->>FMS: Sanitize Attachment Metadata

FMS->>FS: Upload Attachment File

FS-->>FMS: Attachment URL

end

FMS->>Database: Save new Feedback record (with attachment URLs if any)

Database-->>FMS: Confirmation (feedbackId)

FMS->>AuditLog: Record "FEEDBACK\_SUBMITTED" event

AuditLog-->>FMS: Log Confirmation

FMS->>NOS: Request Admin Notification (Event: New Feedback, feedbackId)

NOS-->>FMS: Notification Status

FMS-->>APIGateway: Success Response (201 Created, feedbackId)

APIGateway-->>ClientChannel: Success

ClientChannel->>Citizen: "Thank you for your feedback! Your feedback ID: [ID]"

#### Fig 11. Sequence Diagram: Admin Updates Feedback Status

<div align="center"> <p>Illustrates the secure process of an administrator updating a feedback entry's status, including granular authorization and audit logging.</p> </div>

0

This SDS represents a **professional, expert-level design** for MyGovInsights Pro, addressing the complexities of a national civic engagement platform. It provides a robust foundation for development, ensuring that the system is not only functional but also secure, scalable, and maintainable for years to come.