**ELECTA - Architecture & Technology Specification (v1.0)**

1. **High-Level Architecture Diagram:**

[ Android App ] [ Web Dashboard ] [ Admin Panel ]

| | |

| | |

[ RESTful API Gateway - NodeJS/FastAPI ]

|

-----------------------------------------------

| | | |

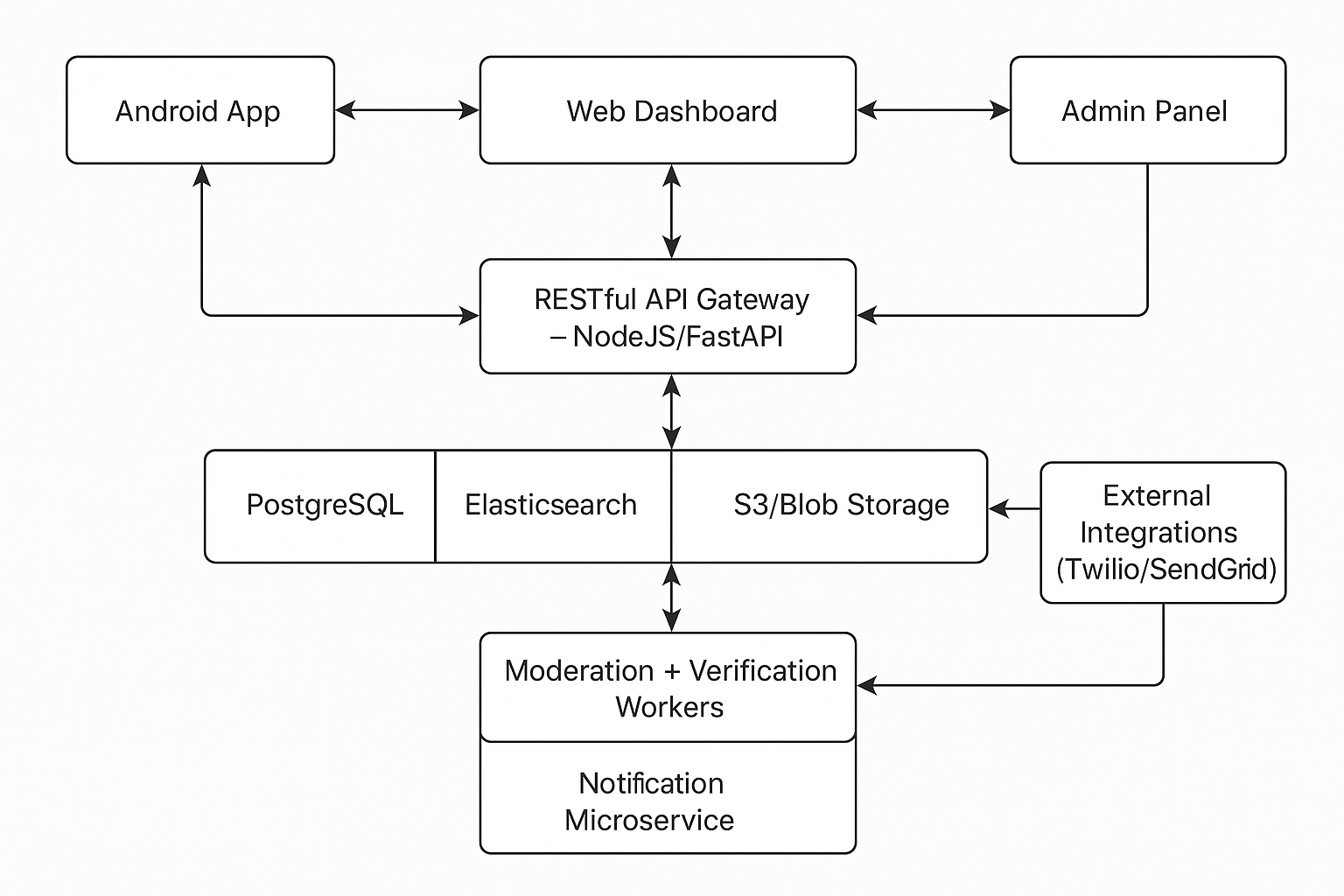
[ PostgreSQL ] [ Elasticsearch ] [ S3/Blob Storage ] [ Redis Queue ]

|

[ Moderation + Verification Workers ]

|

[ Notification Microservice ]



**Mermaid.js Code**  
The following diagram shows the high-level flow of data across ELECTA's frontend clients, backend services, and external integrations. Arrows indicate the primary direction of data flow.

***flowchart TD***

***%% Frontend Clients***

*A1[Android App] --> B[API Gateway<br/>(NodeJS / FastAPI)]*

*A2[Web Dashboard] --> B*

*A3[Admin Panel] --> B*

***%% Backend Services***

*B --> C1[PostgreSQL]*

*B --> C2[Elasticsearch]*

*B --> C3[S3 / Blob Storage]*

*B --> C4[Redis Queue]*

***%% Workers***

*C4 --> D1[Moderation + Verification Workers]*

*D1 --> D2[Notification Microservice]*

*D2 --> E[External Integrations<br/>(Twilio / SendGrid)]*

***%% Layout hints***

*classDef storage fill:#f9f,stroke:#333,stroke-width:1px;*

*classDef queue fill:#bbf,stroke:#333,stroke-width:1px;*

*class C1,C2,C3 storage;*

*class C4 queue;*

***%% Future External Partners***

*F[External Partners<br/>(EC DBs, Fact-Checkers)]*

*B-.->F*

1. **Core System Modules:**

|  |  |
| --- | --- |
| Module | Responsibilities & Justification |
| Admin Panel (Web) | The central nervous system for the platform, enabling all content and user management workflows as defined in the user stories and Deep Dive 5 . |
| Citizen Mobile App (Android) | The primary public interface, fulfilling the Mobile-First strategy. It will handle the entire citizen journey from registration to Q&A submission |
| Public Web Dashboard | Ensures public, shareable access to key information like profiles and Q&A, crucial for transparency and wider reach. |
| Backend API Layer | The core of the system, enforcing all business logic, moderation workflows, and data validation rules derived from the Deep Dives . |
| Database & Storage | A relational database is essential for data integrity, while separate object storage is critical for handling the high volume of source documents required by the Verifiability principle. |
| Search Engine | Fulfills the Intelligent Content Search requirement and is vital for the Q&A sorting and filtering user stories . |
| Queue & Worker Engine | Essential for handling asynchronous tasks like NID verification reviews and notifications, ensuring the main application remains responsive (High Performance ). |
| Notification System | Closes the loop for users on critical status changes like NID verification and question responses, as required by User Stories #9 and #13 . |

1. **Final Technology Stack:**

This stack is chosen to meet the explicit principles of using Proven Technology , ensuring Robust Security , and building for Scalability & Peak Load Management.

* **Backend (API Layer):** **Python with FastAPI.**
  + **Justification:** FastAPI's asynchronous nature provides exceptional performance, directly addressing the High Performance and Peak Load Management principles. Its automatic generation of interactive API documentation is a massive accelerator for an API-First project.
* **Database (Primary DB & Audit Logs):** **PostgreSQL.**
  + **Justification:** PostgreSQL's robustness, transactional integrity (ACID compliance), and strong support for structured data like JSON make it the ideal choice for storing the Factual Records and auditable logs required by the Rigorous Data Management policy.
* **Search:** **Elasticsearch.**
  + **Justification:** It is the industry standard for building the fast, typo-tolerant, and filterable search experiences required by the Q&A module and the Intelligent Content Search principle.
* **Frontend (Web):** **React.js (Admin Panel) & Next.js (Public Dashboard).**
  + **Justification:** React is a mature and powerful library for building the complex, role-based UI of the Admin Panel. Next.js provides server-side rendering for the Public Dashboard, which is critical for performance and SEO.
* **Frontend (Mobile):** **Kotlin (Native Android).**
  + **Justification:** As the officially recommended language for Android development, Kotlin provides the best performance, security, and access to native device features, fulfilling the Mobile-First vision. Its Room DB is perfect for enabling future offline "Lite Mode" capabilities. Its local database, Room DB, will be utilized for caching core profile and Q&A data to enable the offline Lite Mode functionality.
* **Queue/Worker:** **Celery with Redis/RabbitMQ.**
  + **Justification:** Celery is the standard, battle-tested task queue for Python applications. It will reliably manage the asynchronous moderation and verification queues we have designed.
* **Infrastructure & DevOps:** **AWS/GCP, Docker, and GitHub Actions.**
  + **Justification:** A major cloud provider is essential for the cloud-native, auto-scaling architecture. Docker containers will ensure consistency across environments, and GitHub Actions will automate our CI/CD pipeline for rapid, reliable deployments.

1. **Key Design Principles & Implementation:**

* **Verifiability:** All Factual Record fields in the backend schema must enforce non-null constraints for source links. File uploads will be handled via signed URLs to a secure S3/Blob storage bucket.
* **Auditability:** A dedicated audit log table will be created in PostgreSQL. Every mutating action in the API (Create, Update, Delete) performed by an admin will generate an immutable, append-only log entry containing the admin's ID, the action performed, the affected entity's ID, and a timestamp. Read events (e.g., viewing NID data or question submissions) will not be logged in the MVP for performance reasons but may be added later under privileged access contexts.
* **Scalability:** The backend will be designed as stateless services within Docker containers. This allows for horizontal scaling by simply adding more containers to handle peak election loads. Caching layers (Redis) will be used for high-read operations like upvote counts.
* **Security:** Authentication will be managed via JWTs. All communication will be encrypted via HTTPS. The API gateway will enforce rate-limiting. The RBAC logic defined in Deep Dive 5 will be implemented both in the frontend UI and enforced at the API endpoint level. The API Gateway will be configured with specific rate-limiting rules to prevent abuse. Initial limits will be set (e.g., 1 question submission per minute per NID-verified user; 10 upvotes per minute per user) and can be adjusted.

1. **Identified Risks & Mitigation (MVP):**

* **Risk:** Manual NID Review Bottleneck.
* **Mitigation:** The "Verification Officer" role must be scalable with temporary staff during peak election periods. The Admin Panel UI must be highly optimized for rapid side-by-side comparison to minimize review time per submission.

1. **Data Compliance & Disaster Recovery**

* **Data** Localization**:** "All user data and platform databases will be hosted in a cloud region (e.g., AWS ap-south-1 or a relevant GCP region) that complies with the data sovereignty and localization laws of Bangladesh. This decision ensures full legal and regulatory compliance from launch."
* Disaster **Recovery:** "The PostgreSQL database will be configured for automated daily backups. These backups will be stored in a geo-redundant storage location (e.g., a separate AWS S3 bucket in a different region) to meet the platform's Robustness & Reliability standards."