# Project Design Phase-II

## Technology Stack (Architecture & Stack)

| | |
|---|---|
| Date | 29 JUNE 2025 |
| Team ID | **LTVIP2025TMID37420** |
| Project Name | Citizen AI – Intelligent Citizen Engagement Platform |
| Maximum Marks | 4 Marks |

**Technical Architecture:**

The Citizen AI platform will utilize a cloud-native, microservices-based architecture to ensure scalability, reliability, and maintainability. It will leverage AI services for intelligent processing of citizen interactions and robust data storage for efficient information management.

The architectural diagram as below and the information as per the table1 & table 2

**Example:**

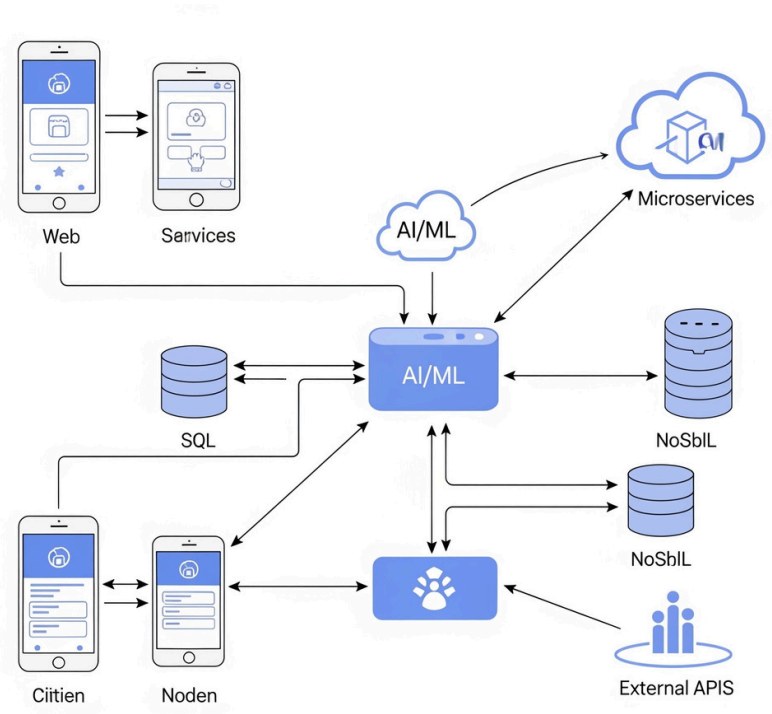


**Table-1 : Components & Technologies:**

| S.No | Component | Description | Technology |
|---|---|---|---|
| 1. | User Interface | How the citizen interacts with the application. | **Web UI:** React.js / Angular, HTML, CSS. **Mobile App:** React Native / Flutter (for iOS & Android). **Chatbot:** Integrated into UI, potentially powered by Rasa or Dialogflow. |
| 2. | Application Logic-1 | **User Management Microservice:** Handles citizen registration, login, profile management. | Python (Flask/Django) / Node.js (Express.js) |
| 3. | Application Logic-2 | **Issue Management Microservice:** Manages submission, categorization, and tracking of citizen issues. | Python (Flask/Django) / Java (Spring Boot) |
| 4. | Database | Relational database for structured data like user profiles, issue metadata, and system configurations. | PostgreSQL / MySQL |
| 5. | Cloud Database | NoSQL database for unstructured or semi-structured data, like raw issue descriptions, historical feedback, and analytics logs. | MongoDB Atlas / AWS DynamoDB / Azure Cosmos DB |
| 6. | File Storage | Storage for media files (e.g., photos uploaded with issues). | AWS S3 / Azure Blob Storage / Google Cloud Storage |
| 7. | External API-1 | **Mapping & Location Services:** For accurate location tagging of reported issues and displaying government service points. | Google Maps API / OpenStreetMap API |
| 8. | Machine Learning Model | **Natural Language Processing (NLP) Model:** To understand and categorize citizen issue descriptions, identify keywords, and extract entities. | Custom models trained with spaCy/NLTK/Hugging Face, deployed via FastAPI, or cloud services like Google Cloud Natural Language API / AWS Comprehend. |
| 9. | Infrastructure (Server / Cloud) | Application Deployment on Cloud for high availability, scalability, and managed services. | **Cloud Server Configuration:** AWS (EC2, ECS, EKS) / Google Cloud (Compute Engine, GKE). |
| 10. | API Gateway | Manages API traffic, authentication, and routing to microservices. | AWS API Gateway / Azure API Management / Google Cloud API Gateway / NGINX Plus |

**Table-2: Application Characteristics:**

| S.No | Characteristics | Description | Technology |
|---|---|---|---|
| 1. | Open-Source Frameworks | AWS API Gateway / Azure API Management / Google Cloud API Gateway / NGINX Plus | React.js / Angular (Frontend), Python Flask/Django / Node.js Express.js (Backend), Kubernetes (Container Orchestration), PostgreSQL / MySQL (Database), Redis (Caching), Apache Kafka / RabbitMQ (Message Broker). |
| 2. | Security Implementations | This includes authentication, authorization, data encryption, and network security. | **Authentication & Authorization:** OAuth 2.0, JWT (JSON Web Tokens), Role-Based Access Control (RBAC). **Data Encryption:** TLS/SSL for data in transit, AES-256 for data at rest. |
| 3. | Scalable Architecture | Designed using a **Microservices Architecture** deployed on a cloud-native platform, allowing individual services to scale independently based on demand. | Kubernetes (for container orchestration and auto-scaling), Cloud Load Balancers (distributing traffic), Auto-scaling groups for compute instances (e.g., AWS EC2 Auto Scaling, Azure Virtual Machine Scale Sets). |
| 4. | Performance | Design considerations include optimizing database queries, utilizing caching mechanisms, content delivery networks (CDNs) for static assets, and asynchronous processing for non-critical operations to handle a high volume of requests per second. | Redis (for caching frequently accessed data), CDN (e.g., Cloudflare, AWS CloudFront) for static content, Optimized database indexing and query tuning, Asynchronous message queues (Kafka/RabbitMQ) for background tasks, |