Civic Issue Management System - Backend Architecture

System Overview

The backend architecture is designed as a microservices-based system with event-driven communication, ensuring scalability, maintainability, and fault tolerance.

Core Architecture Components

1. API Gateway & Load Balancer

```
├── API Gateway (Kong/AWS API Gateway)
│   ├── Rate Limiting
│   ├── Authentication/Authorization
│   ├── Request Routing
│   ├── API Versioning
│   └── Logging & Monitoring
└── Load Balancer (AWS ALB/NGINX)
```

2. Microservices Architecture

A. User Management Service

Responsibility: Handle citizen and admin authentication, profiles, permissions

```
Database: PostgreSQL
Technologies: Node.js/Express, JWT, bcrypt
Endpoints:
- POST /auth/register
- POST /auth/login
- GET /users/profile
- PUT /users/profile
- POST /auth/refresh-token
```

B. Issue Management Service

Responsibility: Core issue lifecycle, CRUD operations, status updates

```
Database: PostgreSQL with spatial extensions (PostGIS)
Technologies: Node.js/Express, Sequelize ORM
Endpoints:
- POST /issues (create issue)
- GET /issues (list with filters)
- PUT /issues/:id (update issue)
- DELETE /issues/:id
- GET /issues/:id/history
- POST /issues/:id/vote
```

C. Media Service

Responsibility: Handle photo/video uploads, processing, storage

```
Storage: AWS S3/CloudFront or Google Cloud Storage
Technologies: Node.js, Sharp (image processing), FFmpeg (video)
Features:
- Image compression and optimization
- Thumbnail generation
- Virus scanning
- Content moderation
Endpoints:
- POST /media/upload
- GET /media/:id
- DELETE /media/:id
```

D. Geolocation Service

Responsibility: GPS processing, geocoding, ward/zone mapping

```
Database: PostgreSQL with PostGIS
External APIs: Google Maps/OpenStreetMap Nominatim
Technologies: Node.js, PostGIS functions
Endpoints:
- POST /geo/reverse-geocode
- GET /geo/wards
- GET /geo/nearby-issues
- POST /geo/boundaries
```

E. Notification Service

Responsibility: Push notifications, SMS, WhatsApp integration

Technologies: Node.js, Firebase FCM, Twilio, WhatsApp Business API
Queue: Redis Bull Queue
Endpoints:
- POST /notifications/send
- GET /notifications/preferences
- PUT /notifications/preferences

F. Analytics Service

Responsibility: Reports, dashboards, KPIs, trend analysis

Database: Time-series database (InfluxDB) + PostgreSQL
Technologies: Node.js, InfluxDB client
Features:
- Response time calculations
- Department performance metrics
- Hotspot identification
- Citizen satisfaction tracking
Endpoints:
- GET /analytics/dashboard
- GET /analytics/reports
- GET /analytics/trends

G. AI Classification Service

Responsibility: Auto-categorize issues from photos/text

Technologies: Python/Flask, TensorFlow/PyTorch, OpenAI API
Features:
- Image classification for issue types
- Text analysis for priority detection
- Deadline estimation based on issue type
Endpoints:
- POST /ai/classify
- POST /ai/estimate-priority
- GET /ai/confidence-score

H. Workflow Engine Service

Responsibility: Issue routing, escalation, deadline management

```
Database: PostgreSQL
Technologies: Node.js, node-cron
Features:
- Department routing logic
- Escalation rules
- Deadline tracking
- Cross-department collaboration
Endpoints:
- POST /workflow/route
- PUT /workflow/escalate
- GET /workflow/rules
```

I. Integration Service

Responsibility: WhatsApp/SMS interfaces, external API connections

```
Technologies: Node.js, Express
Features:
- WhatsApp webhook handling
- SMS parsing and response
- Third-party integrations
Endpoints:
- POST /integrations/whatsapp/webhook
- POST /integrations/sms/webhook
- GET /integrations/status
```

Database Design

Primary Database: PostgreSQL with PostGIS

Core Tables Structure:

```sql
```

```sql
-- Users table
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    phone_number VARCHAR(15) UNIQUE,
    email VARCHAR(255) UNIQUE,
    full_name VARCHAR(255) NOT NULL,
    user_type ENUM('citizen', 'admin', 'field_worker') DEFAULT 'citizen',
    ward_id UUID,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Issues table
CREATE TABLE issues (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    title VARCHAR(255) NOT NULL,
    description TEXT,
    category ENUM('pothole', 'streetlight', 'garbage', 'water', 'other') NOT NULL,
    priority ENUM('low', 'medium', 'high', 'urgent') DEFAULT 'medium',
    status ENUM('received', 'acknowledged', 'assigned', 'in_progress', 'resolved', 'closed') DEFAULT 'received',
    location GEOMETRY(POINT, 4326),
    address TEXT,
    ward_id UUID,
    department_id UUID,
    reporter_id UUID REFERENCES users(id),
    assigned_to UUID REFERENCES users(id),
    deadline TIMESTAMP,
    resolved_at TIMESTAMP,
    vote_count INTEGER DEFAULT 0,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Issue media table
CREATE TABLE issue_media (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    issue_id UUID REFERENCES issues(id) ON DELETE CASCADE,
    media_url VARCHAR(500) NOT NULL,
    media_type ENUM('image', 'video', 'audio') DEFAULT 'image',
    thumbnail_url VARCHAR(500),
    created_at TIMESTAMP DEFAULT NOW()
);
```

```sql
-- Issue updates/history table
CREATE TABLE issue_updates (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    issue_id UUID REFERENCES issues(id) ON DELETE CASCADE,
    updated_by UUID REFERENCES users(id),
    previous_status VARCHAR(50),
    new_status VARCHAR(50),
    comment TEXT,
    created_at TIMESTAMP DEFAULT NOW()
);

-- Departments table
CREATE TABLE departments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    code VARCHAR(50) UNIQUE,
    contact_email VARCHAR(255),
    contact_phone VARCHAR(15),
    response_sla_hours INTEGER DEFAULT 48
);

-- Wards/Zones table
CREATE TABLE wards (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    code VARCHAR(50) UNIQUE,
    boundary GEOMETRY(POLYGON, 4326),
    population INTEGER
);

-- Issue voting table
CREATE TABLE issue_votes (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    issue_id UUID REFERENCES issues(id) ON DELETE CASCADE,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    created_at TIMESTAMP DEFAULT NOW(),
    UNIQUE(issue_id, user_id)
);
```

Caching Layer: Redis

- Session storage
- Real-time issue counters

- Geospatial queries cache
- Rate limiting counters
- Queue management

Time-Series Database: InfluxDB

- Response time metrics
- Issue volume over time
- Department performance stats
- System performance metrics

Message Queue & Event System

Redis + Bull Queue

javascript

```javascript
// Queue Types
const queues = {
  issueProcessing: 'issue:processing',
  notifications: 'notifications',
  escalations: 'escalations',
  analytics: 'analytics:processing',
  aiClassification: 'ai:classification'
};

// Event Types
const events = {
  ISSUE_CREATED: 'issue.created',
  ISSUE_UPDATED: 'issue.updated',
  ISSUE_ASSIGNED: 'issue.assigned',
  DEADLINE_APPROACHING: 'deadline.approaching',
  ESCALATION_TRIGGERED: 'escalation.triggered'
};
```

Security & Privacy Implementation

1. Authentication & Authorization

- JWT-based authentication
- Role-based access control (RBAC)

- Multi-factor authentication for admins
- API key management for integrations

2. Data Privacy

- Data encryption at rest (AES-256)
- Encryption in transit (TLS 1.3)
- PII anonymization for analytics
- GDPR compliance features
- Data retention policies

3. Security Measures

- Input validation and sanitization
- SQL injection prevention (parameterized queries)
- XSS protection
- CSRF tokens
- Rate limiting
- DDoS protection
- Security headers (HSTS, CSP, etc.)

Scalability Features

1. Horizontal Scaling

- Containerized microservices (Docker/Kubernetes)
- Auto-scaling based on CPU/memory metrics
- Database read replicas
- CDN for media content

2. Performance Optimization

- Database indexing strategy
- Query optimization
- Caching layers (Redis, CDN)
- Asynchronous processing
- Connection pooling

3. Monitoring & Observability

- Application Performance Monitoring (APM)
- Distributed tracing
- Centralized logging (ELK stack)
- Health checks and alerts
- Performance metrics dashboard

API Design Patterns

RESTful API Structure

```
GET    /api/v1/issues              # List issues with filters
POST   /api/v1/issues              # Create new issue
GET    /api/v1/issues/:id          # Get specific issue
PUT    /api/v1/issues/:id          # Update issue
DELETE /api/v1/issues/:id          # Delete issue
POST   /api/v1/issues/:id/vote     # Vote on issue
GET    /api/v1/issues/:id/updates  # Get issue history
```

WebSocket Connections

- Real-time issue updates
- Live map updates
- Notification delivery
- Admin dashboard real-time stats

Deployment Architecture

Container Orchestration

```yaml
```

```yaml
# Kubernetes deployment example
apiVersion: apps/v1
kind: Deployment
metadata:
  name: issue-service
spec:
  replicas: 3
  selector:
    matchLabels:
      app: issue-service
  template:
    metadata:
      labels:
        app: issue-service
    spec:
      containers:
      - name: issue-service
        image: civic-platform/issue-service:latest
        ports:
        - containerPort: 3000
        env:
        - name: DB_HOST
          valueFrom:
            secretKeyRef:
              name: db-secret
              key: host
```

Infrastructure Components

- Container Registry (Docker Hub/AWS ECR)
- Kubernetes cluster (EKS/GKE)
- Database cluster (AWS RDS/Google Cloud SQL)
- Message Queue (AWS SQS/Redis Cloud)
- File Storage (AWS S3/Google Cloud Storage)
- CDN (CloudFlare/AWS CloudFront)

Integration Points

External Services

- Google Maps API (geocoding, routing)
- WhatsApp Business API

- Twilio (SMS)
- Firebase FCM (push notifications)
- OpenAI API (AI classification)
- Weather API (for deadline adjustments)

Webhook Support

- WhatsApp message webhooks
- SMS delivery webhooks
- Payment gateway webhooks (if premium features)
- Department system integrations

Data Flow Example

Issue Creation Flow

1. Citizen submits issue via mobile app
2. API Gateway routes to Issue Management Service
3. Media Service processes and stores photos
4. Geolocation Service determines ward/zone
5. AI Classification Service categorizes issue
6. Workflow Engine routes to appropriate department
7. Notification Service sends confirmations
8. Analytics Service updates metrics
9. Real-time updates pushed via WebSocket

This architecture ensures scalability, privacy, and maintainability while supporting all the features you've outlined. The microservices approach allows for independent scaling and deployment of components, while the event-driven design ensures loose coupling and high availability.