CivicConnect: Complete Working Documentation

Generated from repository implementation

March 31, 2026

Contents

1 Document Scope

This document describes the full implementation flow of CivicConnect from source code and infrastructure files in the workspace. It covers:

- system architecture and inter-service communication,
- admin panel and citizen Flutter app behavior,
- backend microservices, data stores, and async pipelines,
- RAG chatbot workflow,
- Kubernetes management/deployment model,
- core entity-relationship model,
- screenshot walkthroughs (admin web and mobile-form-factor Flutter flow).

2 Repository Architecture Summary

CivicConnect is built as a microservices platform with two primary user surfaces:

- Government management interface: Vue 3 admin panel.
- Citizen interface: Flutter application with multiple feature modules.

Gateway and service topology are centered around NGINX reverse proxying to domain services.

2.1 Primary Service Matrix

Service	Stack	Port	Responsibility
admin-service	Go + Gin + GORM	8081	Authentication, admin hierarchy, governments, departments, follows, admin dashboard aggregates
content-service	Node.js + Express + pg	8082 (HTTP), 50051 (gRPC)	Posts, articles, comments, likes/bookmarks, media upload, AI summary trigger
complaint-service	Go + Gin + GORM + PostGIS	8083	Complaint lifecycle, votes, actions, nearby geo queries, complaint media
chatbot-service	Python FastAPI + WebSocket	8084	PulseBot chat API/WS, RAG retrieval, vector ingest/update
ai-worker	Python + RabbitMQ + gRPC	8085, 50052	Async summarization and complaint analysis consumers, assistant/chat worker duties
admin-panel	Vue 3 + Vite	3000 dev / 80 runtime	Government operations UI

2.2 Core Infra Components

- PostgreSQL with multiple logical databases: `admin_db`, `content_db`, `complaint_db`.
- PostGIS enabled in `complaint_db` for geospatial queries.
- RabbitMQ for async job messaging.
- Redis for cache/session/counters/chat history.
- MinIO for object storage (post/complaint uploads).
- ChromaDB as vector store for RAG retrieval.

3 End-to-End Request Flow

3.1 Gateway Routes

NGINX routes client traffic by prefix:

- `/api/v1/auth/*` and `/api/v1/admin/*` → admin-service
- `/api/v1/content/*` → content-service
- `/api/v1/complaints*` → complaint-service
- `/api/v1/chatbot/*` and `/ws/*` → chatbot-service
- `/uploads/*` proxied to MinIO

3.2 Control and Data Planes

1. User interfaces call gateway HTTP APIs.
2. Services persist domain state in service-owned tables/databases.
3. Event-producing actions publish to RabbitMQ queues.
4. AI worker consumes queues and sends back summaries/analysis via gRPC or downstream updates.
5. Chatbot service ingests knowledge documents into ChromaDB and serves contextual responses.

4 Vector Architecture Diagram

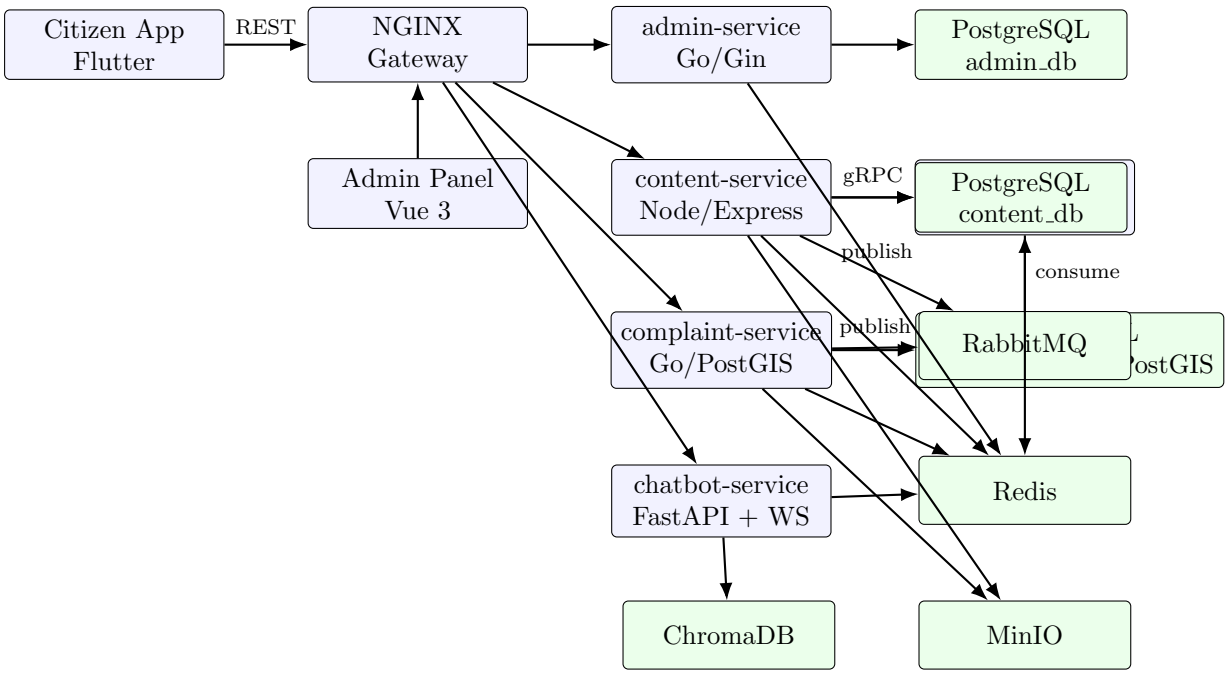


Figure 1: CivicConnect runtime architecture (vector diagram generated in LaTeX).

5 Management Portal Working (Vue Admin Panel)

5.1 Auth and Role Model

The admin panel uses Pinia state with token/user persisted in browser storage. Route guards enforce role-scoped access:

- `super_admin`: municipalities and manager administration.
- `manager`: complaints, departments, dept manager setup, articles.
- `dept_manager`: operational complaint/article/community actions within department scope.

5.2 Operational Screens

- Dashboard: complaint counters, quick actions, recent complaint feed.
- Complaints: status transitions, action timeline, reassignment for category correction.
- Articles: create/edit announcements, category filtering, AI summary visibility.
- Community Posts: official responses to citizen posts and comment auditing.
- Departments, Admins, Municipalities: hierarchical entity management.
- Users: searchable citizen registry with pagination and sort.

6 Citizen App Working (Flutter)

Flutter uses `go_router` and feature modules for Home, Articles, Posts, Complaints, Communities, Chatbot Assistant, and Profile.

6.1 Route and Navigation Pattern

- Top-level auth routes: login/register.
- Full-screen detail routes: article, post, complaint, assistant, government detail.
- Indexed shell route for bottom navigation tabs.

6.2 Client Service Layer

The app calls gateway endpoints through dedicated service classes:

- Auth service for citizen login/register/session restore.
- Content service for articles/posts/comment interactions.
- Complaint service for complaint CRUD and progress states.
- Government service for follow/discovery flows.
- Chatbot service for REST and WebSocket conversations.

7 Backend Services: Working Details

7.1 admin-service (Go)

- Citizen auth: register/login/profile/password/session verification.
- Admin auth: admin login, role-based middleware, dashboard metrics.
- Government structure: municipalities, departments, officials, admins.
- Follow graph: user-official and user-government follow tables.

7.2 content-service (Node.js)

- Tables include posts, comments, likes, bookmarks, multimedia, and articles.
- Supports media upload via MinIO and article management with optional AI summaries.
- Pushes content events to chatbot ingest endpoint for near real-time RAG refresh.
- Exposes gRPC endpoint used by AI worker for enrichment tasks.

7.3 complaint-service (Go + PostGIS)

- Complaint lifecycle with status transitions and completion percentages.
- Complaint comments, action history, image uploads.
- Nearby complaints query using geospatial functions in PostGIS.
- Publishes complaint analysis events for asynchronous AI processing.

7.4 ai-worker (Python)

- Consumes `ai_summarize` and `complaint_analysis` queues from RabbitMQ.
- Provides gRPC-driven AI tasks and stores thread context in Redis.
- Uses Groq/Gemini strategy with fallback behavior.

7.5 chatbot-service (FastAPI + RAG)

- REST endpoint `/ask` and WebSocket endpoint `/ws/client_id`.
- Ingests policy JSON corpus and dynamic content from articles/posts/complaints.
- Stores embeddings in ChromaDB and filters retrieval using metadata (government/department/category).
- Maintains chat history in Redis and exports Prometheus metrics.

8 RAG Chatbot Workflow

8.1 Ingestion Pipeline

1. Startup: policy corpora are ingested into ChromaDB as baseline knowledge.
2. Dynamic refresh: service fetches content and complaint data periodically.
3. Event sync: content-service can call `/ingest/event` so newly created/updated content becomes searchable quickly.

8.2 Query Pipeline

1. User message arrives with optional government context.
2. Relevant vectors retrieved from ChromaDB by semantic similarity + metadata filters.
3. Prompt assembled with conversation context and retrieved snippets.
4. LLM response generated using Groq or Gemini (with fallback templates).
5. Reply and history persisted in Redis.

9 Kubernetes Management Flow

Kubernetes manifests under `infrastructure/k8s` define namespace-scoped deployment for all services and infra.

9.1 Deployment Design

- Namespace: `civic-connect`.
- Workloads: Deployments for services, StatefulSet for PostgreSQL.
- Config and secrets: centralized via ConfigMaps and Secret refs.
- Access: ingress + optional NodePorts for local development.
- Health: readiness/liveness probes on service health endpoints.

9.2 Operational Lifecycle

1. Build images with Skaffold profile.
2. Apply namespace, secrets/config, stateful infra, then app services.
3. Validate probes and ingress routing.
4. Monitor logs and metrics from chatbot and service health endpoints.

10 ER Diagram and Data Model

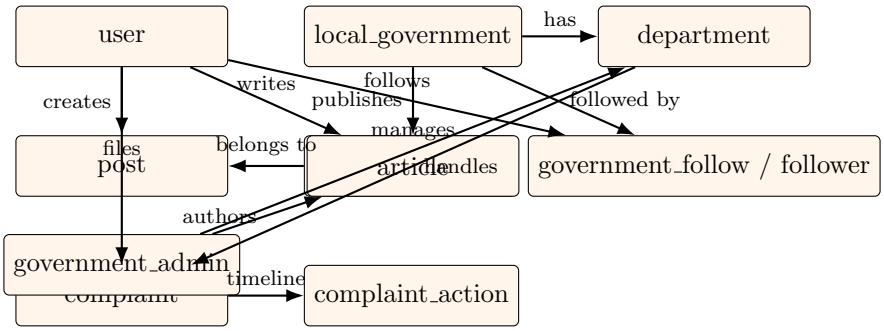


Figure 2: Core ER view across admin/content/complaint domains.

10.1 Schema Notes

- Databases are service-scoped; cross-service joins are avoided at runtime.
- Relationships are maintained through IDs and enforced in service logic.
- Full-text vectors are used in article search.
- Geospatial fields and functions are used in complaint discovery by proximity.

11 YouTube API Integration Status

YouTube API integration is **not implemented** in the current CivicConnect codebase. This section is intentionally retained only as implementation status clarification:

- no runtime routes/services call YouTube Data API,
- no YouTube ingestion or playback business flow is wired in backend services,
- documentation focus remains on the implemented CivicConnect modules.

12 Screenshot Walkthrough

12.1 Admin Panel Screens

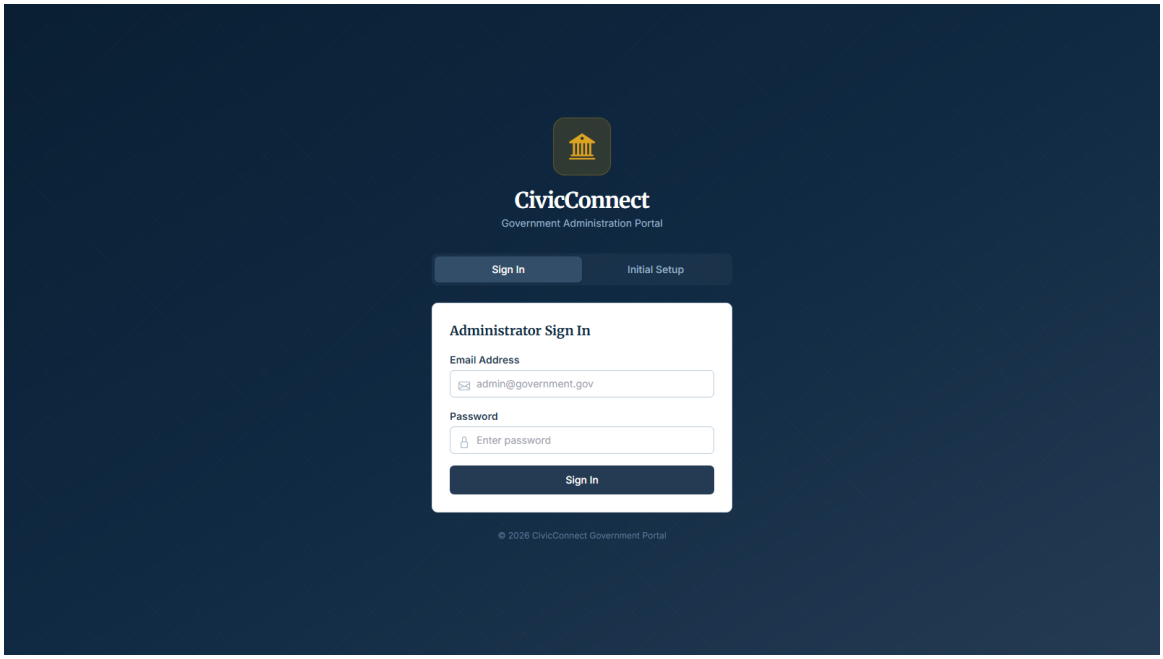


Figure 3: Admin panel login screen.

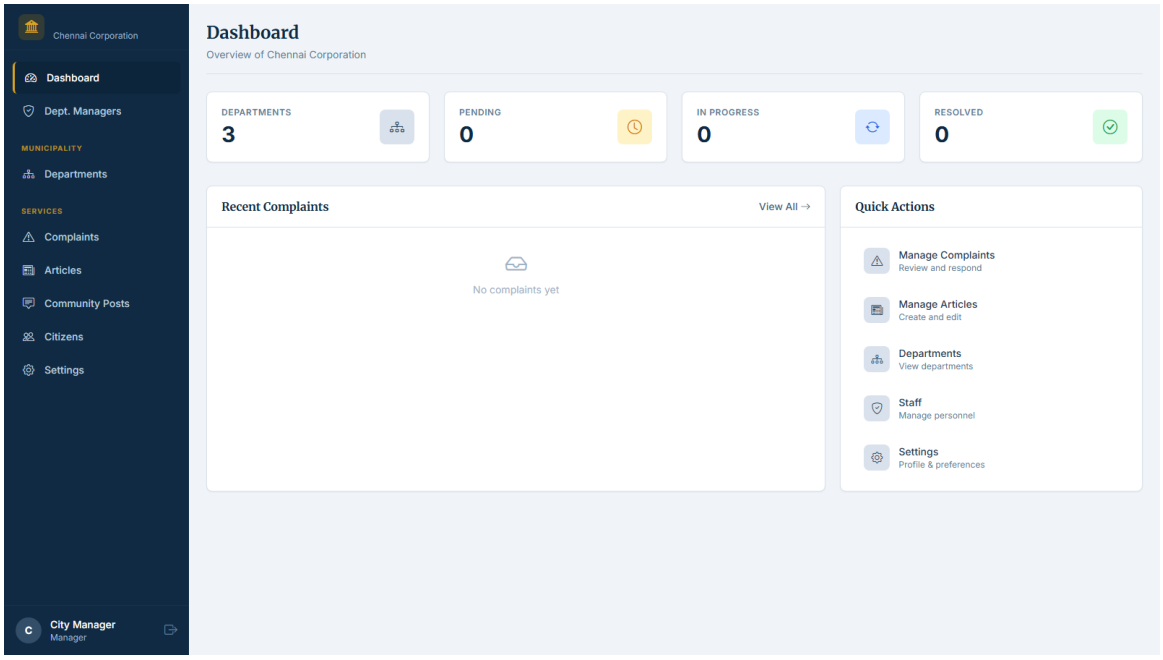


Figure 4: Manager dashboard with complaint metrics and quick actions.

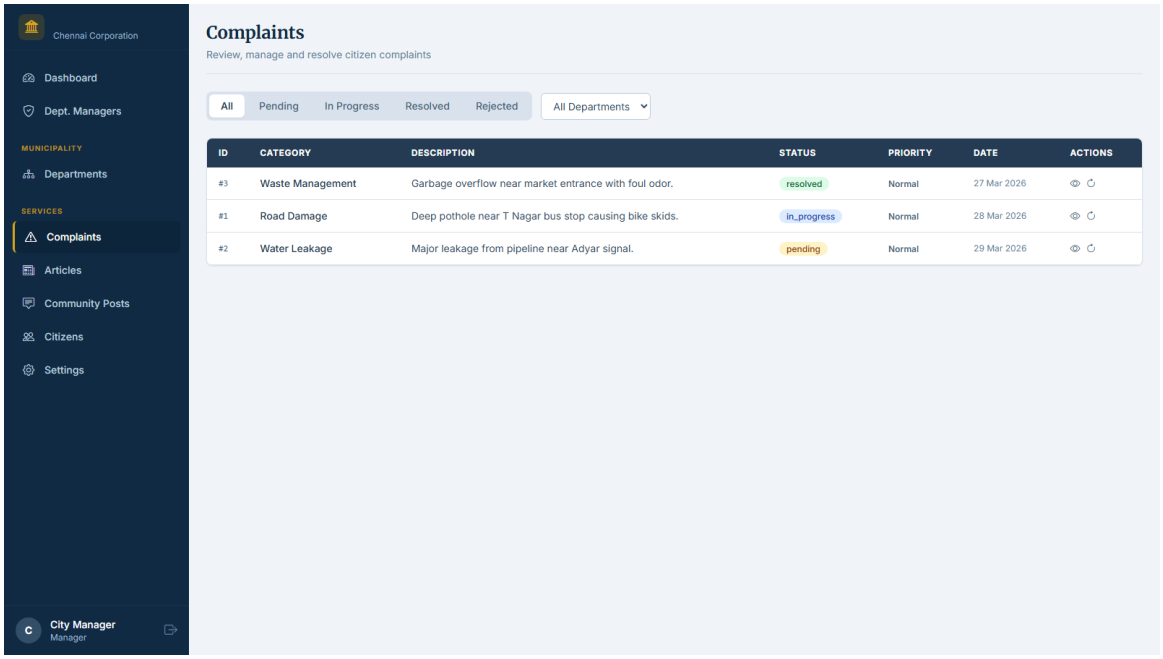


Figure 5: Complaints operations view with status and action handling.

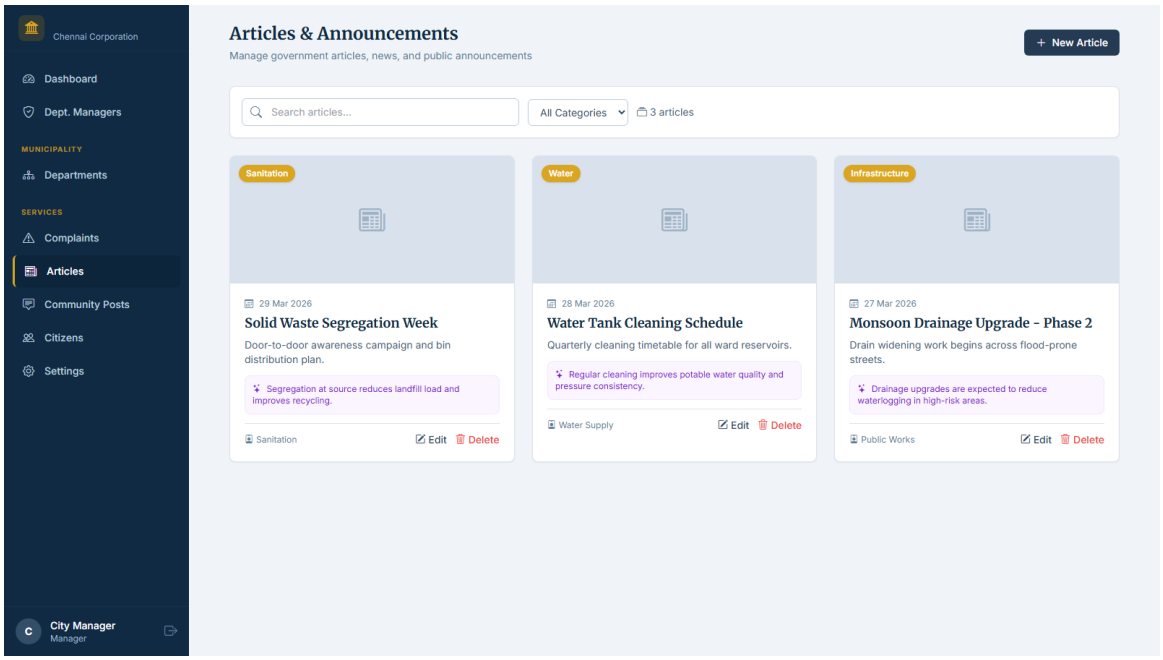


Figure 6: Articles and announcements management.

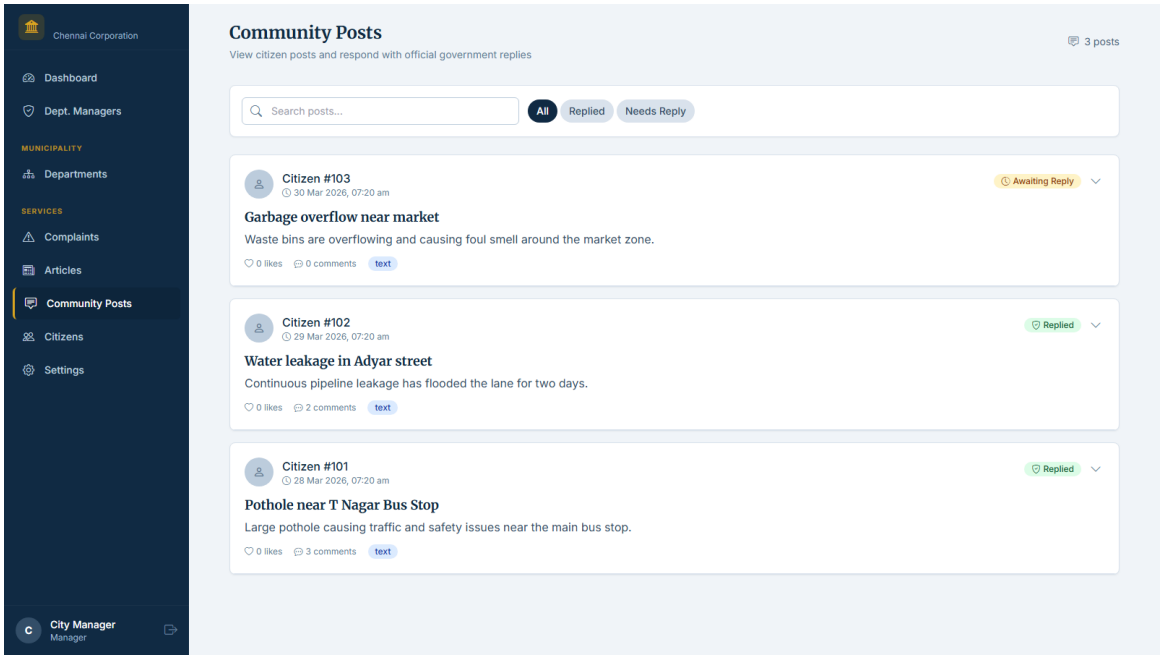


Figure 7: Community posts moderation with official response flow.

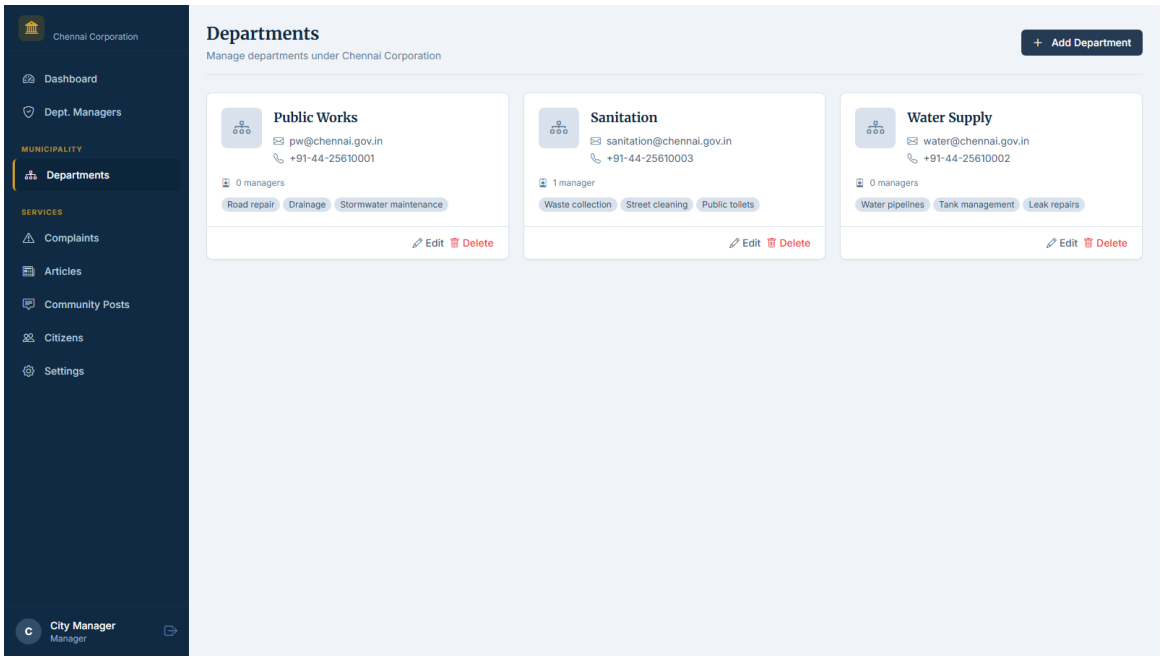


Figure 8: Department management for municipality managers.

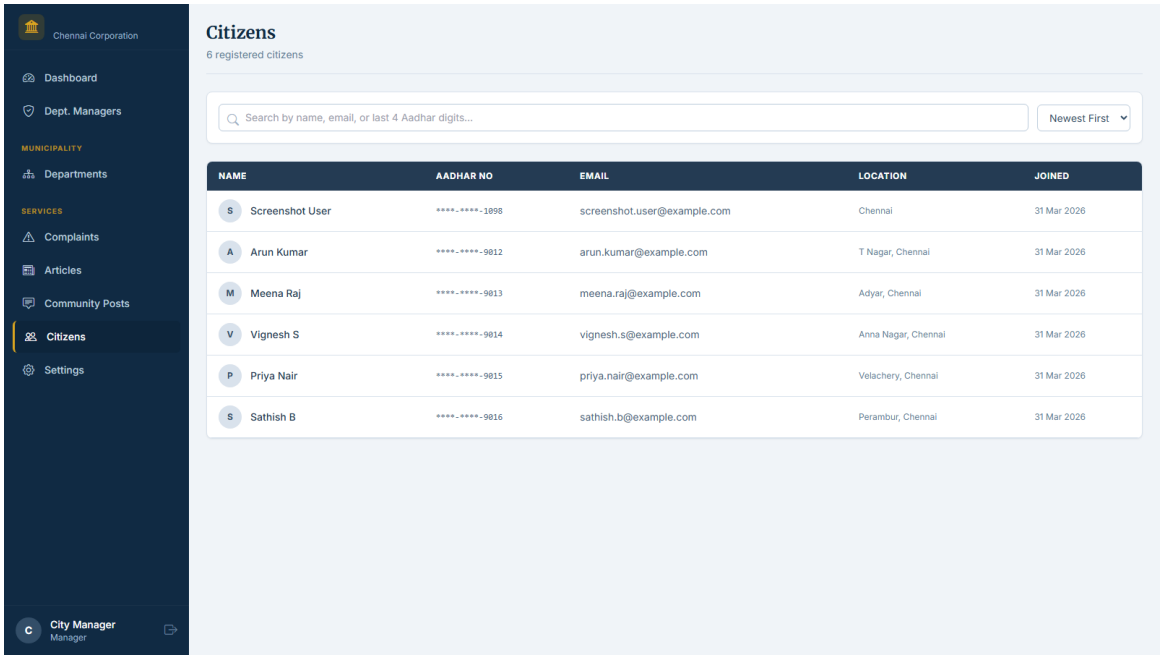


Figure 9: Citizen listing with search and pagination.

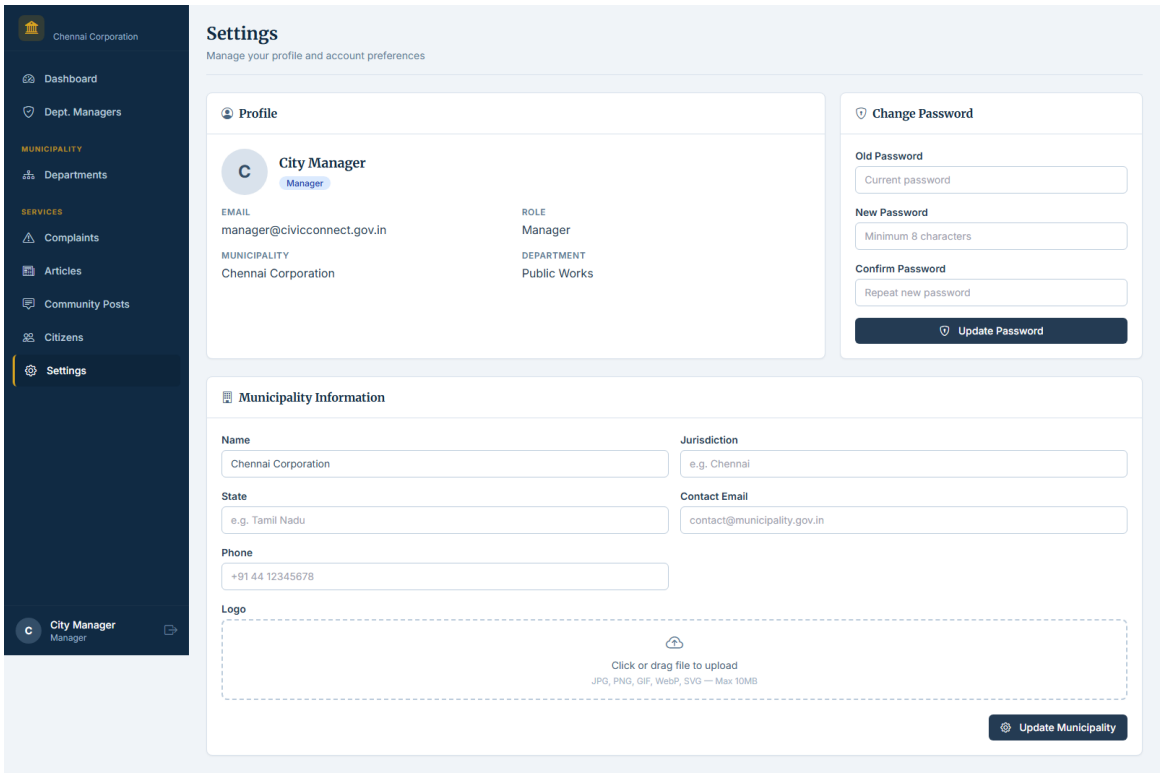


Figure 10: Settings view: account and municipality details.

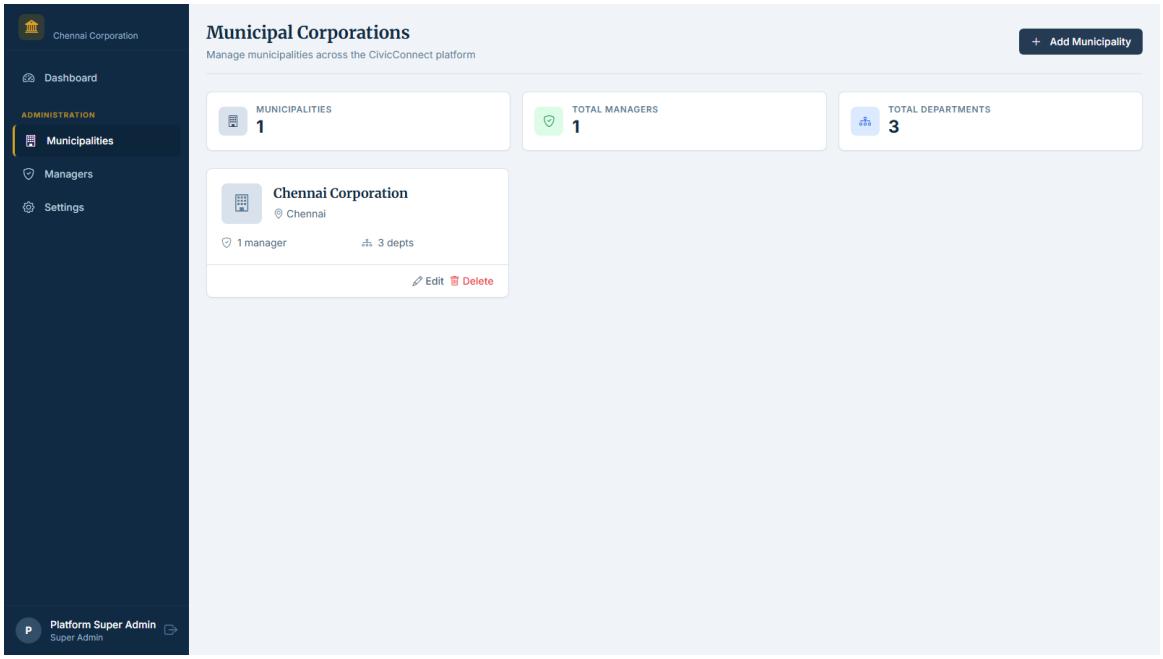


Figure 11: Super admin municipality management.

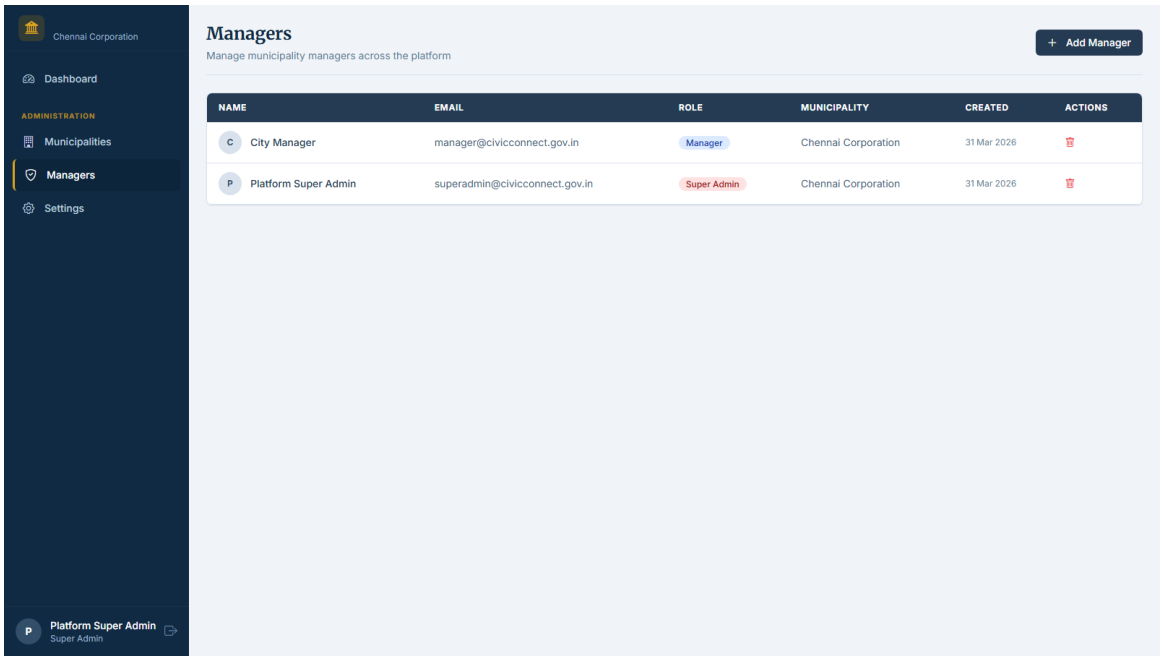


Figure 12: Super admin manager administration table.

12.2 Flutter Mobile-Form-Factor Screens (390x844)

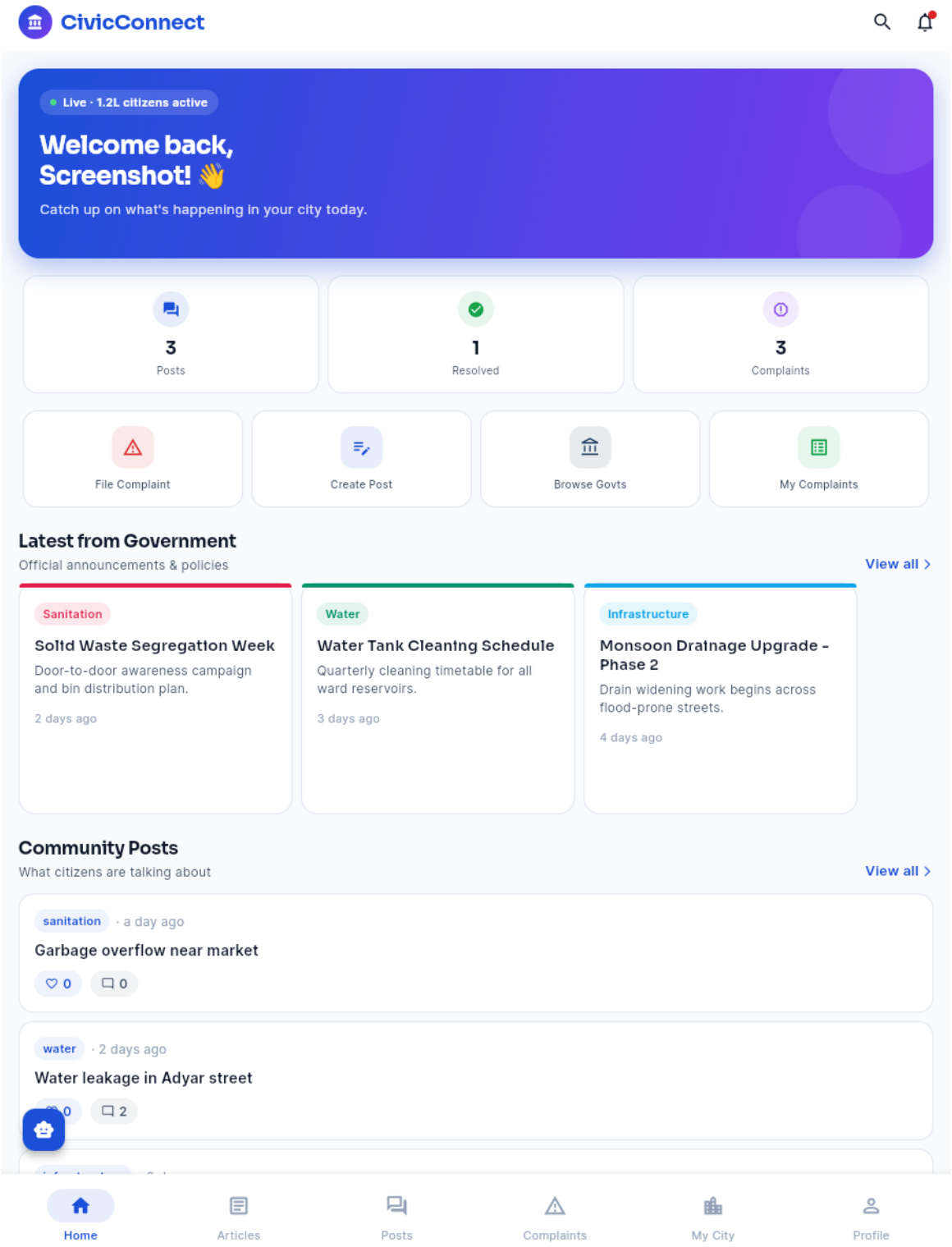
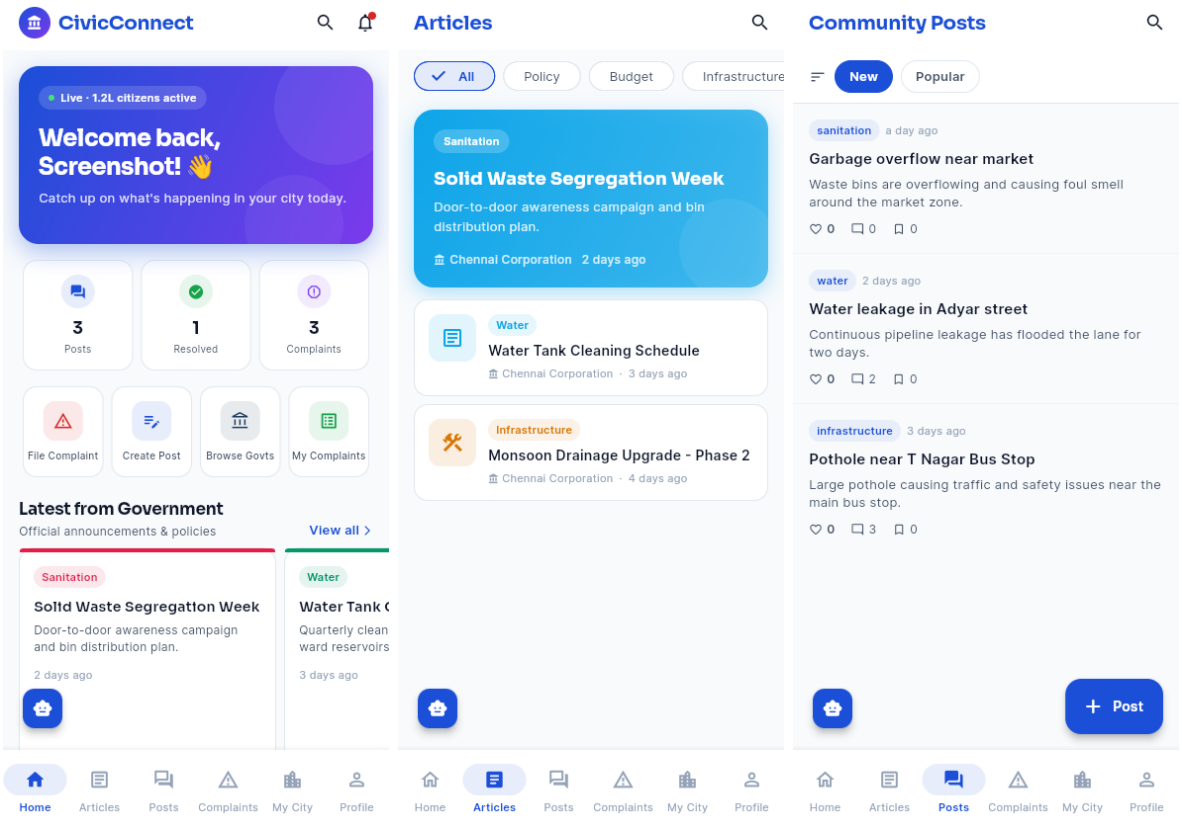
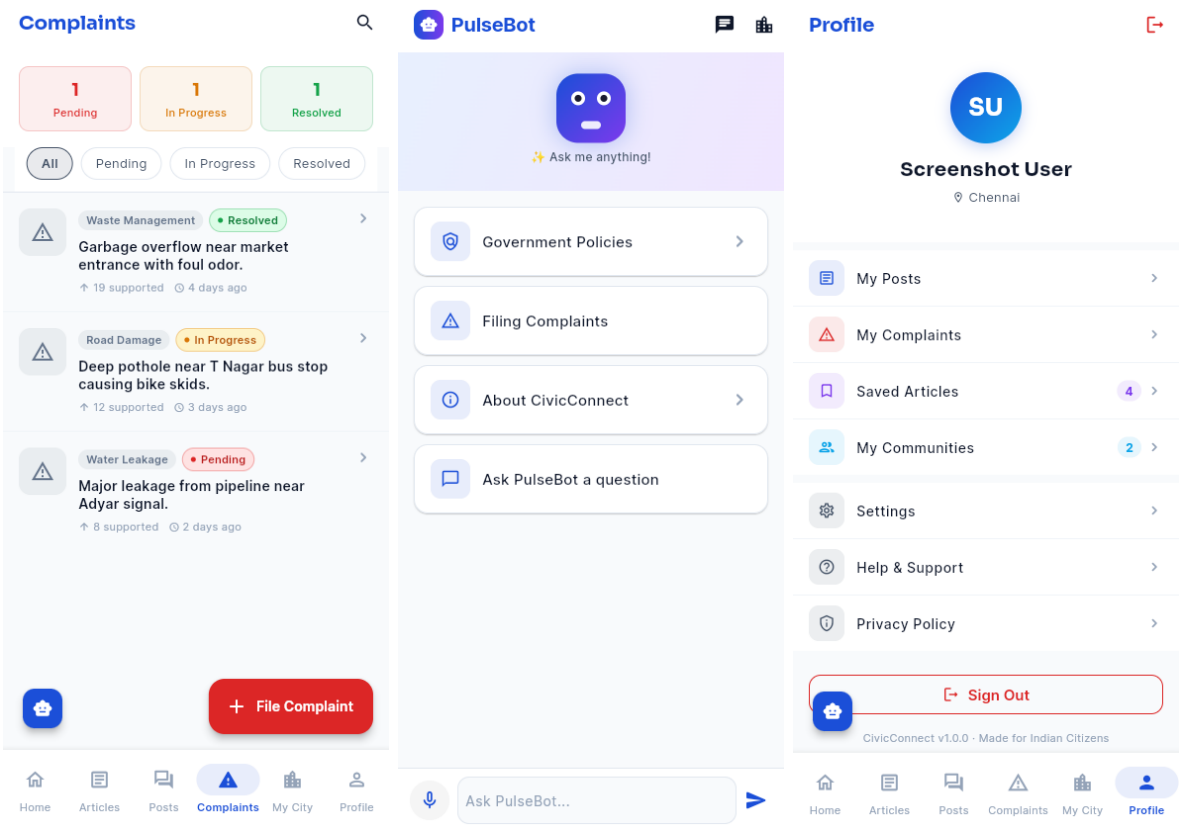


Figure 13: Mobile flow overview generated at phone form factor.



Home Articles Posts

Figure 14: Mobile screens group A.



Complaints PulseBot Profile

Figure 15: Mobile screens group B.

13 Build and Operations Commands

13.1 Docker Compose

```
cd civic-connect/infrastructure/docker
docker compose up --build -d
```

13.2 Kubernetes + Skaffold

```
minikube start
minikube addons enable ingress
kubectl apply -f civic-connect/infrastructure/k8s/
skaffold dev --port-forward
```

13.3 Service Health Validation

- Admin: `/health`
- Content: `/health`
- Complaint: `/health`
- Chatbot: `/health`, `/metrics`

14 Closing Notes

The platform is production-oriented in architecture (service isolation, async processing, infra probes, ingress control) and supports both governance workflows and citizen-facing engagement with AI-assisted support. This document package can be extended with release-specific API appendices and deployment runbooks per environment.