

# **CityAssist — Frontend, Java Backend & DevOps Detailed Task Document**

Purpose: This document provides detailed, assignable tasks for Frontend, Java Backend, and DevOps teams participating in the CityAssist hackathon. Use it as the official assignment brief to distribute to employees.

## **1. Frontend Team — Deliverable & Tasks**

Objective:

Build a mobile-first Progressive Web App (PWA) and responsive admin portal that citizens and operators use daily. The frontend must be production-looking, accessible, and integrate with backend APIs and AI services. Provide mock data for offline demo and clear integration hooks for backend endpoints.

### **1.2 Technologies (recommended)**

React (18+) with Next.js (App Router), TailwindCSS, react-leaflet (Map), react-chartjs-2 (Charts), SWR or React Query for data fetching, TypeScript (optional).

### **1.3 Folder & Component Structure (must follow)**

Suggested repository structure (give to implementers):

```
/app (Next.js App Router)
  /layout.tsx      - root layout and providers
  /page.tsx       - root (redirect to /login)
  /login/page.tsx
  /dashboard/page.tsx
  /map/page.tsx
  /incidents/page.tsx
  /incidents/[id]/page.tsx
  /sensors/page.tsx
  /cctv/page.tsx
  /analytics/page.tsx
  /settings/page.tsx
/components
  AppShell.tsx
  Topbar.tsx
  Sidebar.tsx
  KPICard.tsx
  IncidentTable.tsx
  IncidentDetail.tsx
  CityMap.tsx
  SensorCard.tsx
```

```
TimeSeriesChart.tsx  
VideoPlayer.tsx  
ModalForm.tsx  
NotificationCenter.tsx  
/data  
  users.json, incidents.json, sensors.json, videos.json (mock data)  
/public/assets  
  logo.png, placeholder images, icons  
/styles  
  globals.css, tailwind.css  
/config  
  env.example, README integration notes
```

## 1.4 Pages & Component Responsibilities

Login Page: form, client-side validation, call /api/auth/login (mock or real). Save token+user to localStorage and redirect.

Dashboard: KPIs (active incidents, avg response time, city health index), notification feed, quick actions. KPICard component to show metric + sparkline.

Map Page: interactive map with clustering for incidents & sensors, layer toggles (traffic, AQI), ability to click and view incident card. Fetch data from /api/incidents and /api/sensors.

Incidents List: table with filters (status, severity, zone, date range), server-side simulation of pagination, export CSV button. Incident detail page includes timeline, attachments, assign action (modal).

Sensors Page: list & cards showing latest metrics, health status, quick link to timeseries chart for each sensor.

CCTV Page: grid of placeholder videos with snapshot and download buttons (simulate HLS).

Analytics Page: embed Power BI or show charts; if Power BI key missing, show placeholder and sample visuals generated from mock data.

Settings: user profile, notification preferences, role management UI for admins.

## 1.5 Integration & API Contracts (for frontend)

Provide exact endpoints (backend team will implement) — frontend integrates via these

contracts:

- POST /api/auth/login -> { email, password } -> { access\_token, user }
- GET /api/incidents?status=&severity=&zone=&from=&to=&page=&size -> { items:[], total }
- GET /api/incidents/{id} -> incident\_obj
- POST /api/incidents/{id}/assign -> { assigned\_to }
- GET /api/sensors -> [sensors]
- GET /api/sensors/{id}/timeseries?from=&to= -> [{ts,metric1,...}]
- POST /api/ai/predict/route -> { origin, destination, preferences } -> { routeOptions: [...], reason }

All requests requiring auth must send Authorization: Bearer <token> header.

## 1.6 UI/UX & Accessibility Requirements

- Keyboard navigable, aria attributes on interactive elements, semantic HTML, high contrast, and readable font sizes.
- Focus management after modals, forms, and navigation.
- Prefers-reduced-motion support (CSS media query) and accessible color contrast (AA level).
- Mobile-first responsive design; PWA manifest and service-worker for offline pages (dashboard & map snapshot).

## 1.7 Mock Data & Demo Mode

- Provide /data JSON files to run UI offline. Mock login should accept demo credentials (admin@urbanops.local/admin123, demo@urbanops.city/demo123).
- Where backend calls exist, add clear TODO comments for replacement endpoints.

## 1.8 Acceptance Criteria (Frontend)

- All pages render correctly, navigation between pages works, demo login works.
- Map clusters incidents, filters work, and charts render with tooltips.
- Role-based UI (admin/operator/citizen) shows/hides actions appropriately.
- README with run steps, env placeholders, and integration notes is provided.
- Unit test coverage for major components (recommended: 60%+), and end-to-end tests for critical flows.

## 2. Java Backend Team — Deliverables & Tasks

Objective:

Build a secure, well-documented, and deployable backend consisting of microservices (or a modular monolith) that exposes APIs consumed by frontend and integrates with Python ML services and Power BI. Provide DB migrations, secrets handling, and API documentation.

## 2.2 Technologies (recommended)

Java 17+, Spring Boot 3.x, Spring Data JPA, PostgreSQL, Flyway (DB migrations), Kafka (optional), Redis (cache), Swagger/OpenAPI, JWT for auth, Docker, Helm for k8s deployment.

## 2.3 High-level Architecture & Services

Suggested service modules (can be single project with modules or separate microservices):

- auth-service: user auth, JWT issuing, refresh tokens, RBAC
- core-service: incidents, sensors, attachments, timelines
- ai-gateway: proxy to Python ML services and model cache
- ingest-service: consumers for sensor streams (Kafka) or CSV uploads
- notification-service: push/email/webhook handling

## 2.4 Database Schema & Migration

Core tables:

- users (id UUID, name, email, role, password\_hash, created\_at)
  - incidents (id UUID, title, type, severity, status, location, reported\_at, assigned\_to, data JSONB)
  - incident\_timeline (id, incident\_id, time, actor, text)
  - sensors (id, type, label, zone, lat, lon, status, last\_reported\_at)
  - sensor\_timeseries (hypertable if using TimescaleDB) or partitioned table for time series
- Use Flyway scripts for versioned migrations and provide an initial V1.sql.

## 2.5 API Endpoints (detailed)

Authentication:

- POST /api/v1/auth/login -> {email,password} -> {access\_token, refresh\_token, user}
- POST /api/v1/auth/refresh -> {refresh\_token} -> {access\_token}

Incidents:

- GET /api/v1/incidents -> filters: status,severity,zone,from,to,page,size
- POST /api/v1/incidents -> create incident (validate schema)
- GET /api/v1/incidents/{id} -> incident detail
- POST /api/v1/incidents/{id}/assign -> {assigned\_to}
- POST /api/v1/incidents/{id}/timeline -> add timeline event

#### Sensors & Timeseries:

- GET /api/v1/sensors -> list
- GET /api/v1/sensors/{id}/timeseries -> from,to,interval

#### AI Integration:

- POST /api/v1/ai/predict/flood -> features -> prediction response
- GET /api/v1/ai/models -> list models and versions

### 2.6 Security & Auth

- Use JWT signed with a strong secret or RSA keys. Short-lived access tokens + refresh tokens stored server-side (Redis) or rotated tokens.
- Enforce RBAC with roles: ADMIN, OPERATOR, RESPONDER, CITIZEN. Validate permissions server-side for every action (e.g., assign incident).
- Rate limit critical endpoints and add request validation to avoid injection or malformed payloads.
- Ensure file uploads go to S3 with presigned URLs, virus scan if possible.

### 2.7 Testing Strategy

- Unit tests for services & controllers using JUnit 5 and Mockito.
- Integration tests using Testcontainers for Postgres and Kafka where applicable.
- Contract tests (pact or OpenAPI validation) to ensure frontend-backend compatibility.
- Load tests for ingest and incident APIs (k6/JMeter).

### 2.8 CI/CD & Deliverables (Java)

- Provide Dockerfile, Helm chart, and GitHub Actions (or Jenkinsfile) for build/test/push/deploy.
- Publish OpenAPI YAML and Postman collection.
- Provide runbook: how to start locally, run migrations, seed data, and healthchecks.
- Deliverables: repo with modular services, migrations, tests, CI config, Helm charts, and docs.

## 3. DevOps Team — Deliverables & Tasks (High Effort)

#### Objective:

Design and implement infrastructure, CI/CD pipelines, monitoring, and deployment best practices to run CityAssist reliably at scale. DevOps is a major workstream and will

coordinate closely with all teams.

### 3.2 Cloud & Infra (recommended)

- Cloud Provider: AWS (preferred) or Azure/GCP.
- Core infra: VPC, subnets, EKS (Kubernetes) or ECS, RDS (Postgres), ElastiCache (Redis), S3 for objects, IAM roles for services.
- DNS & CDN: Route53 + CloudFront (or Cloudflare) for static assets and security.

### 3.3 CI/CD Pipeline Requirements

- Pipelines must include: lint -> unit tests -> integration tests (where possible) -> build -> containerize -> security scan (Trivy/Snyk) -> push image -> deploy to staging -> smoke tests -> manual or automatic promotion to production.
- Use feature branch PR gating with mandatory code reviews and pipeline success.
- Enable automated vulnerability scanning and fail build on critical vulnerabilities.

### 3.4 Kubernetes & Deployment

- Create Helm charts for frontend, backend, python services, and supporting infra (ingest, notification).
- Configure Horizontal Pod Autoscaler (HPA) and resource requests/limits for CPU/memory.
- Implement liveness/readiness probes and enable pod disruption budgets.
- Use Ingress controller (NGINX or ALB) with TLS and strict ciphers; configure WAF if available.
- Secrets management: use AWS Secrets Manager or Kubernetes ExternalSecrets to inject secrets securely.

### 3.5 Observability & Monitoring

- Metrics: instrument apps with Micrometer/OpenTelemetry and export to Prometheus. Alert on error rate, latency, and resource exhaustion.
- Traces: use OpenTelemetry and export to Jaeger or AWS X-Ray.
- Logs: centralize logs with ELK or OpenSearch/Loki; enforce structured JSON logs.
- Dashboards: Grafana dashboards for key metrics: request latency, error rate, DB connections, consumer lag, CPU/Memory, disk.
- Alerts: set alerts (PagerDuty or Slack) for P1/P2 events and define runbooks.

### **3.6 Security & Compliance**

- Enforce IAM least privilege, separate accounts for dev/staging/prod, enable MFA.
- Regular image scanning (Trivy) and IaC scanning (Checkov/Terraform-compliance).
- Network security: private DB subnets, security groups, VPC endpoints for S3/RDS.
- Backup & recovery: automated DB backups, point-in-time recovery, and tested restore process.

### **3.7 Cost & Scaling Considerations**

- Use autoscaling policies for K8s and spot instances for non-critical workloads where appropriate.
- Implement cost tagging on resources and daily cost reporting to a central dashboard.
- Optimize storage lifecycle policies for S3 and retention policies for logs.

### **3.8 Deliverables — DevOps**

- Terraform repository for core infra and EKS resources.
- CI/CD pipelines for each service and shared pipeline templates.
- Helm charts and example values for staging & production.
- Grafana dashboard JSON and Prometheus alert rules.
- Runbooks for deployments, incident response, and disaster recovery.

## **4. Cross-team Integration & Handover**

- Weekly integration checkpoints: Frontend <> Backend <> Python <> DevOps <> Power BI <> QA.
- Shared API contract document (OpenAPI) hosted in repo docs; each change must be versioned.
- Mock servers and sample data for frontend & Power BI to develop without waiting for backend.
- End-to-end test plan: include scenarios like 'user reports issue -> model classifies -> operator assigned -> resolution'.
- Handover: each team provides README, deployment steps, environment variables list, and contact points.

## **5. Project Acceptance Criteria (Master)**

- Functional: Core user flows work end-to-end (login, report issue, view incident, receive notification, view Power BI summary).

- Quality: Unit and integration tests present; critical flows covered by e2e tests.
- Performance: App handles expected throughput; backend APIs meet latency targets (e.g., <500ms for CRUD, <2s for model inference).
- Observability: Metrics & logs are available and dashboards configured; alerts configured for critical failures.
- Security: Secrets managed, TLS enforced, basic scanning performed.
- Documentation: Runbooks, API docs, README for each repo, and a final demo playbook.

## 6. Delivery Checklist (for team leads to sign off)

Frontend:

- Pages implemented and responsive.
- Mock data & API hooks present.
- PWA manifest and service worker present.
- README with run instructions and integration notes.

Java Backend:

- APIs implemented per spec and documented (OpenAPI).
- DB migrations and sample seed data included.
- Dockerfile, helm chart, CI config present.

DevOps:

- Terraform for infra and sample deploy to staging.
- CI/CD runs and promotes to staging.
- Monitoring dashboards and alerts configured.
- Runbooks and rollback procedures documented.

## 7. Roles & Communication

- Daily standups for 15 minutes during the hackathon window.
- Slack channel for each domain and a central integration channel.
- Use GitHub issues for tasks, PRs for code review, and labels for priority.
- Each team must nominate a primary contact and a backup.

## 8. Appendix — Useful Templates

- API contract template (OpenAPI)
- DB migration example (Flyway V1.sql)
- Helm values template (staging / production)
- Sample Prometheus alert rule and Grafana dashboard JSON stub