CityAssist — Full Project Document

1. Introduction

CityAssist is a citizen-centric Smart City platform designed to improve daily urban life through personalized alerts, route recommendations, utility outage predictions, and a simple civic reporting flow (image + geolocation). This document is a complete handoff for the Data Science, Power BI, and Java Backend teams. It contains datasets, detailed tasks per team, technical implementation guidance, deliverables, and a sprint roadmap.

2. Datasets (Kaggle + Other Sources)

Below are primary datasets and their relevance to CityAssist:

- **Air Quality (India) -** Station-level AQI and pollutant readings — used for personalized AQI alerts and citizen health dashboards. Link: https://www.kaggle.com/datasets/rohanrao/air-quality-data-in-india
- **Metro Interstate Traffic Volume -** Traffic volume with weather features — used for route ETA modeling and mobility dashboards. Link: https://www.kaggle.com/datasets/ulrikthygepedersen/metro-interstate-traffic-volume
- **Electric Power Outages (US) -** Historical outage events — used to train outage ETA and restoration models. Link: https://www.kaggle.com/datasets/ammaraahmad/electric-power-outages-us-2000-2016
- **Garbage / Civic Image Classification -** Image dataset for classifying civic reports (pothole, garbage, tree fall) to auto-triage reports. Link: https://www.kaggle.com/datasets/asdasdasasdas/garbage-classification
- **Historical Hourly Weather Data -** Weather features to augment outage and route prediction models. Link: https://www.kaggle.com/datasets/selfishgene/historical-hourly-weather-data
- **GTFS / City Open Data Portals -** Local transit feeds (GTFS) and municipal incident feeds — used for accurate routing and event overlays. Link: Vary by city — check municipal open-data portals or transit agencies

3. Data Science Team — Detailed Tasks & Deliverables

3.1 Objectives

Deliver ML models and analytics that directly power personalization, routing, outage prediction, and image triage. Provide model explainability and production-ready artifacts (models, notebooks, inference API specs).

3.2 Core Tasks (Detailed)

A. Personalized Alerting (AQI)

Goal: Score and classify users for AQI alert severity and send tailored alerts.

1. Detailed Steps:

- Data ingestion: ingest station-level AQI and pollutant timeseries; map stations to city zones/wards.
- User profile linking: join user profiles (age, health_flags, commute patterns) with their home/work zones.
- Feature engineering: compute rolling features (1h, 6h, 24h means), pollutant ratios (PM2.5/PM10), sudden-change flags, and temporal features (hour, weekday).
- Modeling strategy: start with rule-based thresholds for immediate safety (e.g., AQI > 300 -> HIGH alert), then train a classifier (Logistic Regression or Gradient Boosting) to predict probability of adverse impact per user profile.
- Explainability: use SHAP or rule templates to produce short human-readable reasons for alerts (e.g., 'PM2.5 rose 60% in last 6 hours near your zone').
- Evaluation: define metrics (precision/recall for high-risk alerts), and establish a monitoring dashboard for false positives/negatives.

2. Deliverables:

- Cleaned AQI zone-level dataset (parquet/CSV).
- Model notebook + trained model artifact (.pkl / saved model).
- Inference contract: JSON schema for /predict_alert (inputs: user_id or zone_id; outputs: level, probability, reason).
- Explainability samples and a monitoring plan.

B. Route Recommender & ETA

Goal: Return ranked alternate routes with ETA considering congestion, events, and preferences.

3. Detailed Steps:

- Data sources: traffic volume dataset, optional live telemetry (if available), local GTFS for transit overlays, and events calendar.
- Feature engineering: historical segment speeds, time-of-day multipliers, incident flags, weather adjustments, and user preference features (avoid highways, prefer bus lanes).
- Baseline approach: use an external routing engine (OSRM/GraphHopper) for geometry and baseline travel time; apply congestion multipliers or learned segment delay adjustments.
- ML approach: train segment-level regression models (Random Forest / XGBoost) predicting additional delay; compose route ETA by summing adjusted segment times.

- Output: ranked route options with ETA, expected variance, and short reason text (e.g., 'Delay due to accident at X').
- Evaluation: measure ETA MAE, ranking accuracy (did route chosen minimize travel time?), and user-acceptance tests.

4. Deliverables:

- Segment-level historical speed database and derived features.
- Trained delay prediction model artifacts and inference spec (/predict_route input/output schema).
- Notebook with evaluation results and baseline comparisons.

C. Outage ETA Estimation

Goal: Predict time-to-restore for utility outages to power user communications and operator prioritization.

5. Detailed Steps:

- Data ingestion: historical outage records, cause, duration, zone, and timestamps; merge with weather and grid-load proxies.
- Target definition: restore_time_hours (or minutes); consider right-censoring and truncated events.
- Feature engineering: outage cause encoding, time-of-day, day-of-week, weather at time of event, historical mean restore time for zone and similar causes.
- Modeling: regression models (Gradient Boosting, LightGBM) and survival analysis approaches if censoring is significant.
- Uncertainty: provide prediction intervals or quantiles to communicate confidence.
- Evaluation: MAE, RMSE, and calibration of prediction intervals.

6. Deliverables:

- Processed outage dataset and feature store descriptions.
- Trained model with inference spec (/predict_eta).
- Monitoring dashboard for predicted vs actual restore times.

D. Image Triage (Civic Reporting)

Goal: Auto-classify images sent by citizens to triage and prioritize responses.

7. Detailed Steps:

- Dataset curation: collect labeled images for potholes, garbage, fallen trees, streetlight issues; augment with rotations, flips, brightness/contrast variations.
- Model choice: lightweight CNN (MobileNetV2 / EfficientNet-lite) for fast inference and mobile deployment considerations.
- Confidence handling: flag low-confidence predictions for manual review and create a human-in-the-loop workflow.

- Output metadata: predicted label, confidence score, bounding box (optional), and suggested priority.
- Evaluation: per-class precision/recall and confusion matrix; monitor class imbalance.

8. Deliverables:

- Labeled dataset and augmentation recipe.
- Trained model artifacts and inference contract (/classify_image).
- Human-review workflow definition and expected SLA for low-confidence cases.

3.3 Explainability, Logging, and Monitoring

Common requirements across models:
- Include a short human-readable `reason` for each prediction.
- Log inputs, model version, prediction, and key features for drift detection.
- Set up automated retraining triggers when data drifts or performance degrades.

4. Power BI Team — Detailed Tasks & Deliverables

4.1 Objectives

Build executive and operator dashboards that provide actionable insights for city managers and public-facing summaries for citizens.

4.2 Dashboards & Tasks (Detailed)

Citizen Health Dashboard

- Visuals: zone-level AQI map, time-series of AQI, top affected demographics, alert counts.
- KPIs: current AQI averages, number of active high-risk alerts, % of population in high-risk zones.
- Data needs: aggregated AQI timeseries by zone-hour, user demographics aggregated by zone.

Service KPI Dashboard

- Visuals: tickets created vs resolved, SLA heatmap by zone, ticket lifecycle funnel.
- KPIs: tickets_open, tickets_resolved, avg_restore_time, SLA_compliance_pct.
- Data needs: reports and outages tables aggregated by status and timestamps.

Mobility Dashboard

- Visuals: peak congestion corridors (map), average commute delay over time, route success rate.
- KPIs: avg_commute_delay, % routes meeting ETA, top congested segments.
- Data needs: route ETA predictions, observed travel times, traffic counts.

Outage Prediction Monitor

- Visuals: predicted vs actual restore time scatter, prediction error distribution, outage count over time.

- KPIs: prediction_MAE, % predictions within X hours, outage_count_by_cause.
- Data needs: predictions table, actuals table, outage metadata.

Waste Management Dashboard

- Visuals: bar charts for issue types (pothole, garbage), ward-level issue hotspots, trendlines.
- KPIs: top_issue_types, avg_resolution_time_by_type.
- Data needs: classified images summary, reports table.

4.3 Data Model & DAX Examples

Data model guidance: create a star schema with fact tables (reports, outages, predictions) and dimension tables (zones, time, users). Use incremental refresh where possible.

- **TicketsCreated:** COUNTROWS(Reports)
- **TicketsResolved:** CALCULATE(COUNTROWS(Reports), Reports[status] = "resolved")
- **AvgRestoreHours:** AVERAGEX(Outages, DATEDIFF(Outages[reported_at], Outages[restored_at], HOUR))
- **HighAQIAlerts:** CALCULATE(COUNTROWS(AQI), AQI[PM2_5] > 300)

4.4 Deliverables & Deployment

Deliverables: .pbix files for each dashboard, documentation for data sources and refresh, and measure definitions (DAX).

Deployment guidance: use Power BI Gateway for scheduled refresh from PostgreSQL on-prem or configure scheduled refresh in the Power BI service for cloud databases. Store credentials in secure vaults and do not embed secrets in PBIX files.

5. Java Backend Team — Detailed Tasks & Deliverables

5.1 Objectives

Provide secure, versioned, and scalable microservices that serve the PWA and integrate ML outputs and operator tools. Ensure reliability, observability, and clear API contracts.

5.2 Microservices & Tasks (Detailed)

UserService

- Functions: user CRUD, profile preferences (age, health_flags, commute patterns), JWT authentication and token refresh.
- Tasks: design REST endpoints, validation, unit tests, and integration with identity provider (OAuth2) if required.
- Deliverables: OpenAPI spec, database migrations, integration tests.

ReportService

- Functions: accept reports (image + metadata), pre-signed URLs for S3 uploads, ticket lifecycle management, and operator assignment.

- Tasks: idempotency handling, data validation, rate limiting for abuse prevention, attachment size checks.
- Deliverables: API contract, sample Postman requests, DB schema for reports.

NotificationService

- Functions: manage subscriptions, send push notifications via FCM/Browser Push, email fallbacks, and snooze settings.
- Tasks: queueing (Redis/RabbitMQ), retry/backoff strategies, user preference handling, and opt-in/opt-out compliance.
- Deliverables: notification producer/consumer, metrics for delivery rates, and runbook for failures.

RoutingService (API Gateway)

- Functions: orchestrate calls to external routing engines (OSRM/GraphHopper) and internal ML route predictions.
- Tasks: route caching, rate limiting, assemble ranked routes and reasons, fallback strategies.
- Deliverables: documented endpoints, caching policy, and performance tests.

IntegrationService

- Functions: connectors to utility providers and third-party city systems (webhooks, polling).
- Tasks: durable connectors, data mapping, credential management, and error handling.
- Deliverables: connector templates, mapping documentation, and test harness.

5.3 Non-Functional Requirements

- Security: JWT auth, role-based access controls, input validation, secure presigned S3 uploads, and rate limiting.
- Observability: structured logging (JSON), distributed tracing (Jaeger), metrics (Prometheus/Grafana), and alerting (PagerDuty).
- Scalability: containerized services (Docker), orchestrated with Kubernetes, and autoscaling policies.
- Resilience: retries with exponential backoff, circuit breakers, and graceful degradation.
- Testing: unit tests, integration tests, contract tests, and load testing.

5.4 Deliverables

- OpenAPI/Swagger specs for all services.
- Postman collection for common flows (report creation, subscribe to alerts, route request).
- DB schema and migration scripts.
- Docker images and Helm charts for deployment.
- Runbooks for incident management.

6. Integration Overview

High-level integration points:
- Data Science -> Java: ML teams supply model artifacts and inference contracts (REST endpoints or batch output tables). Java services will call ML inference endpoints or fetch predictions from a predictions table.
- Java -> Power BI: create sanitized aggregate tables for Power BI or export scheduled CSV snapshots; alternatively Power BI connects directly to PostgreSQL (read-only user) with views designed for dashboards.
- Authentication: single-sign-on or token-based auth for PWA and operator portals.
- Storage: images stored in S3 with presigned URLs; sensitive data encrypted at rest.

7. Suggested Sprint Roadmap (Example)

Sprint 0 — Setup (3 days)

- Repo setup, CI pipeline, baseline infra skeleton (k8s/helm placeholders).
- Data download scripts and data lake folder structure.
- Basic DB schema and local dev environment.

Sprint 1 — AQI & PowerBI Prototype (7–10 days)

- Implement AQI ingestion, rule-based alerts, and Citizen Health Power BI prototype.
- Deliver: cleaned AQI dataset, initial /predict_alert contract, PBIX for Citizen Health dashboard.

Sprint 2 — Routing & Outage Baselines (10–14 days)

- Baseline route recommender (OSRM + rules) and outage ETA regression baseline.
- Deliver: route API stub, outage prediction artifact, Mobility and Outage PBIX pages.

Sprint 3 — Image Classifier & Integration (7–10 days)

- Image classification model for triage and integrate with ReportService; implement notification flows.
- Deliver: image classifier artifact, Postman collection, end-to-end test run.

Sprint 4 — Hardening & Monitoring (7 days)

- Add monitoring, runbooks, security reviews, and performance testing; prepare handoff documentation.
- Deliver: runbooks, monitoring dashboards, and final PBIX files.

8. Conclusion & Next Steps

This document is a comprehensive handoff containing datasets, in-depth tasks per team, technical guidance, and a suggested roadmap. Next steps:
1) Review responsibilities with each team lead and adjust sprint estimates.
2) Secure access to datasets and credentials (kaggle tokens, S3, DB).
3) Start Sprint 0 with repository and infra foundation.

If you want, I can now generate:
- a single Word file with this content (ready),
- a condensed one-page leader summary, or
- separate team-specific PDFs for direct distribution.