AI4Purpose Hackathon

Natural Language Situation Room Agent

Technical Requirements & Implementation Specifications

Project Track:	Professional/Student
Challenge Theme:	Communication & Information Management
Target Context:	Lebanon Health Crisis Response
Document Date:	February 12, 2026
Version:	1.0

Executive Summary

This document outlines the comprehensive technical requirements for developing a Natural Language Situation Room Agent designed to address the critical gap in situational awareness faced by decision-makers during health emergencies in Lebanon. The system leverages advanced AI technologies, including Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), and Geographic Information Systems (GIS) to enable non-technical crisis managers to access complex data through simple natural language queries.

Problem Statement

During emergencies such as the 2020 Beirut Port Explosion, 2024-2025 Israeli War on Lebanon, and ongoing health crises, decision-makers face a critical challenge: **data is siloed across different institutions**, preventing a unified, real-time view of the situation. This fragmentation leads to:

- Delayed response times due to inability to quickly access relevant information
- Inefficient resource allocation from lack of comprehensive situational awareness
- Duplicated efforts across agencies working with incomplete information
- Communication gaps between technical data systems and non-technical decision-makers
- Inability to rapidly identify critical resources (e.g., nearest functional hospital with trauma capacity)

Proposed Solution

A mobile-first Natural Language Situation Room Agent that enables crisis managers to query complex operational data using conversational language. Example queries include:

- "Where is the nearest functional hospital with available trauma beds and a clear route for ambulances?"
- "Show me all shelters within 5km of Beirut with capacity for 100+ people"
- "Which medical facilities still have power and oxygen supply in the affected area?"
- "Map the distribution of medical personnel across the southern districts"
- "What are the current wait times at emergency departments in operational hospitals?"

Alignment with Needs Assessment

This solution directly addresses priority needs identified in the Lebanon context assessment:

Identified Gap	How Solution Addresses It	Priority
Data Fragmentation: Critical datasets siloed across institutions	RAG system integrates data from multiple sources into unified knowledge base	HIGH
Low Situational Awareness: Decision makers lack unified interface	Natural language interface provides instant access to integrated real-time data	CRITICAL
Communication Gaps: Information exchange via informal channels	Structured query system with standardized data access protocols	HIGH
Lack of Early Warning: No AI-enabled prediction capability	Foundation for future predictive analytics based on historical query patterns	HIGH
Geographic Isolation: Remote communities cannot access	Mobile-first design enables access from any location with connectivity	MEDIUM-HIGH

System Architecture Overview

High-Level Architecture

The system follows a modern three-tier architecture optimized for real-time crisis response:

Layer	Components	Technology Stack
Presentation Layer (Mobile App)	• Flutter mobile application • Voice input interface • Interactive map visualization • Real-time notifications	• Flutter 3.x • Flutter Map / Mapbox • Speech-to-Text APIs • Material Design 3
Application Layer (Backend Services)	• Natural language processing • Query understanding & routing • Response generation • Authentication & authorization	• Python FastAPI • LangChain framework • OpenAI GPT-4/Claude API • OAuth 2.0
Data Layer	• Vector database for RAG • GIS data storage • Real-time health system data • Cached responses	• Pinecone/Weaviate • PostgreSQL + PostGIS • Redis cache • AWS S3

RAG System Requirements

RAG Architecture Overview

The Retrieval-Augmented Generation (RAG) system is the core intelligence component that enables natural language queries over complex health system data. The system combines semantic search with LLM-powered response generation to provide accurate, contextual answers.

1. Data Ingestion Pipeline

- **Hospital & Health Facility Data:** Location, operational status, bed capacity (total, ICU, trauma), available specialties, equipment status (oxygen, power, medical devices), staff availability
- **Geographic Data:** Road networks, checkpoint locations, damage assessments, accessibility zones, administrative boundaries, population density
- **Resource Tracking:** Medical supply inventories, ambulance locations and availability, shelter capacities, water and food distribution points
- **Real-time Status Updates:** Emergency department wait times, power outages, communication network status, security incidents affecting access
- **Historical Crisis Data:** Past emergency response patterns, resource utilization statistics, population displacement patterns

2. Vector Database Configuration

Requirement	Specification	Implementation Notes
Embedding Model	OpenAI text-embedding-3-large or sentence-transformers/paraphrase-multilingual-mpnet-base-v2	Dimension: 1024/1536. Multilingual support for Arabic/English
Vector Database	Pinecone (managed) or Weaviate (self-hosted)	Must support metadata filtering for geographic queries
Index Structure	Hierarchical indexing by data type and geographic region	Separate namespaces for: facilities, resources, incidents, geographic
Update Frequency	Real-time for critical data, batch updates every 15 minutes	WebSocket connections for hospital status, 15-min batch for other da
Metadata Schema	Structured fields for filtering: location (lat/lon), timestamp, category, priority, source	Enable hybrid search combining semantic + metadata filters

3. Query Processing Pipeline

The query processing pipeline transforms natural language input into structured database queries and synthesizes comprehensive responses:

1. **Step 1 - Query Understanding:** LLM extracts intent, entities (locations, facilities, resources), constraints (distance, capacity, availability), and temporal context
2. **Step 2 - Query Classification:** Route to appropriate handler (geographic search, facility lookup, resource availability, routing, statistical analysis)
3. **Step 3 - Retrieval Strategy:** Construct hybrid search combining semantic similarity with metadata filters (e.g., distance < 10km AND facility_type = 'hospital' AND status = 'operational')
4. **Step 4 - Context Enrichment:** Retrieve relevant chunks from vector DB + fetch latest real-time data from operational databases
5. **Step 5 - Response Generation:** LLM synthesizes answer combining retrieved context, real-time data, and domain knowledge. Include confidence scores and data freshness timestamps
6. **Step 6 - Visualization Preparation:** Generate map coordinates, route overlays, and statistical summaries for mobile app display

4. LLM Integration Specifications

Component	Primary Option	Fallback Option	Configuration
Query Understanding	GPT-4-Turbo	Claude 3 Sonnet	Temperature: 0.1, Max tokens: 500
Response Generation	GPT-4-Turbo	Claude 3 Sonnet	Temperature: 0.3, Max tokens: 1000
Embeddings	text-embedding-3-large	sentence-transformers	Batch size: 100 documents
Function Calling	GPT-4-Turbo	Claude 3 with prompt engineering	Strict schema validation

Mobile Application Requirements

Flutter Application Specifications

The mobile application serves as the primary user interface for crisis managers and first responders. It must be intuitive, fast, and functional even under poor network conditions.

1. Core Features

- **Natural Language Query Interface:** Text input with autocomplete suggestions based on common queries. Voice input with Arabic and English speech recognition. Query history with bookmarking capability
- **Interactive Map View:** Real-time facility markers with status indicators (operational/damaged/offline). Route visualization for ambulance navigation. Heat maps for population density and resource availability. Offline map caching for areas with poor connectivity
- **Response Display:** Structured answer cards with key information highlighted. Expandable sections for detailed data. Source attribution and data freshness indicators. Quick action buttons (Call, Navigate, Share)
- **Multi-Modal Results:** Tabular data for comparisons. Charts for statistical information. Lists with filtering and sorting. Export to PDF/CSV for reporting
- **Offline Capabilities:** Cache recent queries and responses. Store critical facility data locally. Queue queries when offline for later submission. Offline map navigation

2. Flutter Package Requirements

Category	Packages	Purpose
State Management	• Riverpod 2.x • flutter_bloc	App state, API response caching, reactive updates
Networking	• dio • retrofit • connectivity_plus	HTTP client with interceptors, network status monitoring
Maps & Location	• flutter_map • latlong2 • geolocator • geocoding	Map display, coordinate handling, location services
Local Storage	• hive • shared_preferences • path_provider	Offline data caching, user preferences, file storage

UI Components	• flutter_svg • cached_network_image • shimmer	Vector graphics, image caching, loading states
Voice & Input	• speech_to_text • permission_handler	Voice recognition, microphone permissions
Analytics & Monitoring	• sentry_flutter • firebase_crashlytics	Error tracking, crash reporting

3. User Interface Design Requirements

- **Accessibility:** Minimum touch target size 48x48 dp. High contrast mode support. Screen reader compatibility. Support for Arabic RTL layout
- **Performance:** App launch time < 2 seconds. Query response rendering < 500ms. Smooth 60fps scrolling and animations. Memory usage < 200MB
- **Responsive Design:** Support phones (4.5" - 6.7") and tablets (7" - 12"). Adaptive layouts for portrait/landscape. Different information density for phone vs tablet
- **Theming:** Light and dark mode support. High-visibility emergency mode with larger fonts and buttons. Customizable color schemes for different agencies
- **Localization:** Full support for English and Arabic. Date/time formatting for Lebanon timezone. Emergency alerts in both languages

Backend Infrastructure Requirements

1. API Service Specifications

- **Framework:** FastAPI (Python 3.11+) for high-performance async operations and automatic API documentation
- **Core Endpoints:** • POST /api/v1/query - Natural language query submission • GET /api/v1/facilities - Filtered facility search • GET /api/v1/resources - Resource availability lookup • POST /api/v1/route - Calculate optimal route between points • GET /api/v1/status - System health and data freshness • WebSocket /ws/updates - Real-time status updates
- **Authentication:** OAuth 2.0 with role-based access control (RBAC). Support for emergency responder, health official, and administrator roles. API key authentication for system integrations
- **Rate Limiting:** 100 requests/minute for authenticated users. 10 requests/minute for unauthenticated (emergency access). Separate higher limits for real-time status updates
- **Response Format:** JSON with standardized error codes. GeoJSON for geographic data. Include metadata: query_id, timestamp, data_sources, confidence_score

2. Database Requirements

Database Type	Technology	Purpose	Key Features
Primary Database	PostgreSQL 15+	Operational data, user accounts, query logs	ACID compliance, JSON support, full-text search
Geospatial Extension	PostGIS 3.3+	Geographic data, spatial queries, routing	Spatial indexing, distance calculations, polygon operations
Vector Database	Pinecone / Weaviate	RAG embeddings, semantic search	Fast similarity search, metadata filtering, horizontal scaling
Cache Layer	Redis 7+	Query caching, session storage, rate limiting	In-memory speed, TTL support, pub/sub for real-time upda
Time-Series Data	TimescaleDB extension	Resource availability trends, historical analysis	Automatic partitioning, continuous aggregates

3. Infrastructure & Deployment

Component	Development	Production (MVP)	Production (Scale)
Compute	Local Docker Compose	AWS ECS Fargate / Google Cloud Run	Kubernetes (EKS/GKE)
Database Hosting	Local PostgreSQL	AWS RDS / Cloud SQL	Managed PostgreSQL with read replicas
Vector DB	Local Weaviate	Pinecone (managed)	Pinecone / self-hosted Weaviate cluster
File Storage	Local filesystem	AWS S3 / GCS	CDN + object storage
Monitoring	Local logs	CloudWatch / Stackdriver	Grafana + Prometheus + Sentry

Data Sources & Integration Requirements

Successful operation requires integration with multiple data sources. For the hackathon MVP, we will use simulated/sample data based on the Lebanon needs assessment findings.

Required Data Sources (MVP Simulation)

Data Category	Required Fields	Update Frequency	Sample Size
Health Facilities	• Facility name, type, location (lat/lon) • Operational status • Bed capacity (total, ICU, trauma) • Available specialties • Equipment status • Contact information	Real-time for status, Daily for capacity	150-200 facilities covering Beirut and surrounding
Geographic Data	• Road networks • Administrative boundaries • Checkpoint locations • Damaged infrastructure • Population centers	Weekly updates	Complete Lebanon road network, key districts
Resources	• Medical supply inventories • Ambulance fleet (location, status) • Shelter information • Distribution points	Hourly for ambulances, Daily for supplies	50-100 tracked resources
Incidents & Alerts	• Incident type, location, severity • Road closures • Security alerts • Weather warnings	Real-time	20-50 active incidents (simulated)

Data Generation Strategy

For the hackathon demonstration, we will generate realistic synthetic data based on:

- Actual facility locations from OpenStreetMap (hospitals, clinics, pharmacies)
- Road networks from OSM with simulated damage/access restrictions based on recent conflict zones
- Population distribution data from WorldPop or similar sources
- Simulated operational status reflecting realistic scenarios (30% facilities damaged, 50% at reduced capacity, 20% fully operational)
- Historical patterns from the 2020 Beirut explosion and 2024-2025 conflict for realistic resource distribution

Ethics & Responsible AI Requirements

Following the AI4Purpose hackathon's ethics-by-design approach and the European Commission framework for trustworthy AI, our system must embed ethical considerations throughout development:

1. Respect for Human Agency & Oversight

- **Human-in-the-Loop:** All critical decisions (resource allocation, evacuation orders) require human approval. System provides recommendations with confidence scores, not autonomous actions
- **Transparency:** Clearly indicate AI-generated vs. human-verified information. Show data sources and timestamps for all responses. Explain reasoning when possible
- **Override Capability:** Users can flag incorrect information. Manual data entry option for field updates. Emergency override mode to bypass normal authentication in crisis situations

2. Privacy & Data Governance

- **Data Minimization:** Collect only essential data for crisis response. No personally identifiable information (PII) stored unless absolutely necessary
- **Encryption:** All data encrypted at rest (AES-256) and in transit (TLS 1.3). Separate encryption keys for sensitive vs. public data
- **Access Control:** Role-based access control with audit logging. Query logs anonymized after 90 days. Emergency responder data access limited to active incidents
- **Data Retention:** Operational data retained for 1 year. Personal data deleted within 30 days of incident closure. Statistical aggregates retained for historical analysis

3. Fairness & Non-Discrimination

- **Equal Access:** System available in both Arabic and English. UI optimized for low-end devices (budget Android phones). Offline mode for areas with poor connectivity
- **Unbiased Recommendations:** Resource allocation recommendations based on objective criteria (distance, capacity, severity) not demographic factors. Regular audits to detect and correct geographic or demographic biases
- **Inclusive Design:** Accessibility features for users with disabilities. Voice input for users with low literacy. Visual alternatives for users with hearing impairments

4. Technical Robustness & Safety

- **Accuracy Validation:** All LLM responses validated against ground truth data where available. Confidence thresholds for automatic responses (>0.85 for display, <0.85 requires verification)
- **Failure Handling:** Graceful degradation when AI components fail (fallback to database queries). Offline mode with cached data. Clear error messages with alternative action suggestions
- **Testing:** Unit tests for all critical functions (>80% coverage). Integration tests for RAG pipeline. Load testing for 1000+ concurrent users. Adversarial testing for prompt injection and jailbreaking attempts
- **Monitoring:** Real-time monitoring of query accuracy, response times, error rates. Alerts for anomalous patterns. Regular review of flagged incorrect responses

Development Tools & Environment

Required Development Tools

Tool Category	Specific Tools	Purpose
Programming Languages	• Python 3.11+ • Dart 3.0+ (Flutter)	Backend services and mobile app development
IDEs	• VS Code with extensions • PyCharm Professional • Android Studio	Code editing, debugging, mobile development
Version Control	• Git • GitHub/GitLab	Code collaboration and version management
API Development	• FastAPI • Postman/Insomnia • Swagger UI	Backend API development and testing
Database Tools	• DBeaver / pgAdmin • Redis CLI • Pinecone Console	Database management and querying
AI/ML Libraries	• LangChain • OpenAI Python SDK • sentence-transformers • pandas, numpy	RAG implementation and data processing
Testing	• pytest (Python) • flutter_test • Mockito	Unit and integration testing
Container & Deployment	• Docker • Docker Compose • AWS CLI / gcloud	Local development and deployment

Python Dependencies (requirements.txt)

Key Python packages for backend development:

- `fastapi>=0.104.0 - Modern web framework for building APIs`
- `uvicorn[standard]>=0.24.0 - ASGI server for FastAPI`
- `langchain>=0.1.0 - Framework for LLM applications and RAG`
- `openai>=1.6.0 - OpenAI API client for GPT models`
- `pinecone-client>=3.0.0 - Vector database client`

- sentence-transformers>=2.2.0 - Embedding models
- sqlalchemy>=2.0.0 - Database ORM
- psycopg2-binary>=2.9.0 - PostgreSQL adapter
- geoalchemy2>=0.14.0 - GIS extension for SQLAlchemy
- pydantic>=2.5.0 - Data validation
- python-jose[cryptography]>=3.3.0 - JWT authentication
- redis>=5.0.0 - Redis client for caching
- httpx>=0.25.0 - Async HTTP client
- pytest>=7.4.0 - Testing framework
- python-multipart>=0.0.6 - Form data parsing

Implementation Roadmap

Hackathon Development Timeline

The following timeline is optimized for the hackathon format (30-31 January - 1 February 2026):

Phase	Duration	Key Deliverables	Team Focus
Day 1 Morning (Fri 30 Jan)	4 hours	• Environment setup • Team role assignment • Data collection/generation • Basic project structure	Full team: Setup and planning
Day 1 Afternoon	4 hours	• Backend API skeleton • Database schema • Sample data ingestion • Flutter app scaffold	Split: Backend team + Mobile team
Day 1 Evening	4 hours	• RAG pipeline basic implementation • LLM integration • Query processing logic	Backend focus
Day 2 Morning (Sat 31 Jan)	4 hours	• Complete RAG integration • API endpoints functional • Mobile UI implementation	Integration and UI development
Day 2 Afternoon	4 hours	• Map integration • Query-response flow • Error handling • Testing	Feature completion
Day 2 Evening	3 hours	• Bug fixes • Performance optimization • Demo scenario preparation	Quality assurance
Day 3 Morning (Sun 1 Feb)	3 hours	• Final testing • Demo practice • Presentation preparation • Documentation	Presentation preparation
Day 3 Afternoon	2 hours	• Final pitch • Q&A preparation	Full team: Presentation

Team Composition & Roles

Role	Responsibilities	Required Skills	Time Allocation
Tech Lead	• Architecture decisions • Code review • Integration oversight • Technical documentation	Full-stack, LLM/RAG experience, System design	100% team, oversight all components
Backend Developer (×2)	• FastAPI development • RAG pipeline • Database design • LLM integration	Python, FastAPI, LangChain, PostgreSQL	80% backend, 20% integration
Mobile Developer (×2)	• Flutter app development • UI/UX implementation • State management • Map integration	Flutter/Dart, REST APIs, Mobile UI design	80% mobile, 20% integration
Data Engineer	• Data collection/generation • Database seeding • Vector DB setup • GIS data processing	Python, SQL, GIS tools, Data modeling	60% data, 40% backend support
UX/UI Designer	• Interface design • User flow optimization • Accessibility review • Demo scenario design	Figma, Mobile UX, Accessibility standards	70% design, 30% user testing

Success Criteria & Evaluation Alignment

The following criteria map directly to the hackathon evaluation rubric (6 categories, 5 points each, 30 total):

Rubric Category	Target Score	How We Achieve It	Key Demonstrators
Relevance to Challenge & Needs Assessment	5/5	• Directly addresses "Low Situational Awareness" gap from pre-hackathon needs assessment • Targets identified priority: Communication & Information Management	• Show gap analysis mapping • Demonstrate understanding of Lebanon crisis context
AI Innovation, Responsibility & Ethics	4.5/5	• RAG system showcasing modern AI architecture • Ethics-by-design implementation • Transparency in AI responses	• Live RAG demo with source attribution • Confidence scores displayed • Ethical considerations document • Multilingual support
Feasibility & Technical Soundness	4.5/5	• Working prototype with core functionality • Clear, documented architecture • Realistic technology choices	• End-to-end demo scenario • Architecture diagram • API documentation • Code quality and organization
Sustainability Consideration	4/5	• Cost-effective cloud deployment plan • Open-source technologies • Partnership opportunities with health ministry	• Cost breakdown analysis • Scaling roadmap • Training documentation • Open-source license
Adoption, Scalability & Real-World Fit	4/5	• User-centered design for crisis managers • Offline capabilities for Lebanon context • Mobile-first approach	• User persona and journey • Scaling architecture plan • Demo with realistic Lebanon scenario • Stakeholder feedback plan
Team Qualifications & Complementarity	4.5/5	• Diverse skill set (AI, mobile, health domain) • Clear role distribution • Demonstrated execution capability	• Team bios highlighting complementary skills • Demonstrated teamwork in execution • Domain knowledge of health systems

Target Score: 25-28/30 (Outstanding to Excellent Range)

Risks & Mitigation Strategies

Risk	Impact	Probability	Mitigation Strategy
LLM API rate limits or outages	High	Medium	• Use multiple API keys • Implement caching for common queries • Prepare fallback responses • Test with rate limit buffer
Limited time for complete RAG implementation	High	High	• Use LangChain templates • Pre-prepare data ingestion scripts • Simplify to 2-3 core query types • Focus on quality over quantity
Mobile app testing on different devices	Medium	Medium	• Test on 2-3 representative devices • Use emulators for additional coverage • Focus on core user journey
Data generation delays	Medium	Low	• Pre-generate sample datasets before hackathon • Use OpenStreetMap for facility locations • Create data generation scripts in advance
Integration challenges between mobile and backend	High	Medium	• Define API contracts early • Use mock servers for parallel development • Schedule integration checkpoints • Use Postman collections for API testing
Demonstration technical failures	High	Low	• Record backup demo video • Test demo scenario 3+ times • Prepare offline fallback • Have screenshot backups

Appendices

Appendix A: Example Queries & Expected Responses

Scenario 1:

Query: "Where is the nearest functional hospital with available trauma beds and a clear route for ambulances?"

Expected Response: System retrieves hospitals within configurable radius (default 10km), filters by operational status and trauma bed availability, calculates routes avoiding checkpoints/damaged roads, returns ranked list with distances and estimated travel times. Response includes facility details, current capacity, and map visualization.

Scenario 2:

Query: "Show me all shelters in southern Beirut with capacity for at least 100 people"

Expected Response: GIS query filters shelters by location (southern Beirut districts), capacity >= 100, returns list with current occupancy, available capacity, facilities (water, medical, food), contact information, and map markers.

Scenario 3:

Query: "What is the current status of hospitals in the affected area?"

Expected Response: Aggregates hospital status data (operational, damaged, offline), provides summary statistics, lists by district, includes key capacity metrics (total beds, available beds, oxygen status, power status), highlights critical shortages.

Scenario 4:

Query: "Find medical facilities within 5km that still have oxygen supply"

Expected Response: Combines geographic filtering (5km radius from user location or specified point) with equipment status filter (oxygen available), returns prioritized list based on distance and supply levels, includes contact info and directions.

Appendix B: Key Resources & References

- **Project Documentation:** • MEDAIGENCY Project Overview: Needs assessment presentations provided • AI4Purpose Hackathon Rules: Registration pack • Evaluation Rubric: Final Pitch Evaluation Rubric PDF
- **Technical Documentation:** • LangChain RAG Tutorial: https://python.langchain.com/docs/use_cases/question_answering/ • OpenAI API Documentation: https://platform.openai.com/docs/ • Flutter Documentation: https://docs.flutter.dev/ • FastAPI Documentation: https://fastapi.tiangolo.com/ • PostGIS Tutorial: https://postgis.net/workshops/postgis-intro/
- **Ethics & Responsible AI:** • EU AI Ethics Guidelines: https://ec.europa.eu/digital-strategy/our-policies/european-approach-artificial-intelligence • Ethics By Design presentation: Provided at hackathon kickoff
- **Data Sources:** • OpenStreetMap: https://www.openstreetmap.org/ (for facility locations) • Lebanon Administrative Boundaries: HDX/OCHA data • Sample health facility data: To be generated based on needs assessment findings
- **Development Tools:** • GitHub Template Repository: [To be created for team collaboration] • Docker Compose Setup: [To be provided in project starter kit] • API Testing Collection: [Postman collection to be shared]

Appendix C: Environment Setup Checklist

- [ ] Install Python 3.11+ and verify with `python --version`
- [ ] Install Flutter SDK 3.x and verify with `flutter doctor`
- [ ] Install Docker Desktop and verify with `docker --version`
- [ ] Set up IDE (VS Code with Python, Flutter, Docker extensions)
- [ ] Create OpenAI API account and obtain API key
- [ ] Create Pinecone account and set up free tier vector database
- [ ] Clone project repository and install dependencies (`pip install -r requirements.txt`, `flutter pub get`)
- [ ] Set up local PostgreSQL database with PostGIS extension
- [ ] Configure environment variables (.env file with API keys, database URLs)
- [ ] Download sample datasets and run data ingestion scripts
- [ ] Test backend API with `uvicorn main:app --reload`
- [ ] Test Flutter app on emulator/device with `flutter run`
- [ ] Verify end-to-end flow with sample query
- [ ] Set up Git repository with proper .gitignore (exclude .env, API keys)
- [ ] Review hackathon rules and evaluation criteria one final time

Conclusion

This requirements document provides a comprehensive blueprint for developing the Natural Language Situation Room Agent for the AI4Purpose Hackathon. The proposed solution directly addresses critical gaps identified in the Lebanon health system needs assessment, particularly the low situational awareness faced by decision-makers during crises.

By combining modern AI technologies (LLMs, RAG, GIS) with ethical-by-design principles and a mobile-first approach optimized for the Lebanon context, this system has strong potential for both hackathon success and real-world impact. The clear alignment with evaluation criteria, realistic technical scope for the hackathon timeline, and focus on user-centered design position this project well for achieving an outstanding score (25-28/30).

Key success factors:

- Strong relevance to identified priority needs (Low Situational Awareness, Communication Gaps)
- Innovative AI application (RAG-powered natural language interface for crisis management)
- Technically sound and feasible within hackathon constraints (proven technologies, clear architecture)
- Ethical considerations embedded from design stage (transparency, privacy, fairness)
- Real-world fit for Lebanon context (offline mode, multilingual, mobile-first, checkpoint awareness)
- Clear path to sustainability and scalability (open-source, cloud-native, partnership opportunities)

With proper execution following this requirements specification, the team will deliver a compelling demonstration of how AI can meaningfully enhance health crisis response in resource-constrained, high-complexity environments.

—— End of Requirements Document ——