Civic Connect - Master Development Plan

Timestamp: Saturday, September 6, 2025, 11:09 PM IST

Location: Hyderabad, Telangana, India

Objective: Deliver a functional MVP in under 72 hours by eliminating all dependencies and providing a single source of truth for all teams.

1. Component Architecture & Data Flow

1. **User Interaction:** The Citizen/Admin interacts with the React/Next.js frontend.
2. **API Request:** The frontend sends REST API requests (with a JWT for authenticated routes) to the FastAPI Backend.
3. **AI Orchestration (Core Logic):** For the /smart-create route, the backend constructs a multimodal prompt (text + image) and sends it to the external Gemini/Groq API, specifying the submit_civic_issue_report tool.
4. **AI Response:** The AI service returns a structured JSON object (a tool_call).
5. **Database Transaction:** The backend validates this JSON and executes a transaction against the MongoDB Atlas database (e.g., saves the new report).
6. **API Response:** The backend returns a success/error message and any requested data to the frontend.
7. **UI Update:** The frontend updates the UI based on the API response.

2. The Data Contract: Database Schemas (MongoDB)

This is the definitive structure for our data. The Pydantic models in the backend will enforce this contract.

users Collection

```
# file: backend/models.py
from pydantic import BaseModel, Field, EmailStr
from typing import Literal, List
from datetime import datetime

class UserInDB(BaseModel):
    id: str = Field(..., alias="_id")
    email: EmailStr
    hashed_password: str
    role: Literal['citizen', 'admin'] = 'citizen'
    created_at: datetime = Field(default_factory=datetime.utcnow)
```

reports Collection

```
# file: backend/models.py
class GeoJSONLocation(BaseModel):
    type: Literal['Point'] = Field("Point", const=True)
    coordinates: List[float]  # Follows GeoJSON standard: [longitude, latitude]

class ReportInDB(BaseModel):
    id: str = Field(..., alias="_id")
    user_id: str  # Corresponds to the _id of a user in the 'users' collection

    # --- AI-Generated Fields ---
    title: str = Field(..., max_length=100)
    category: Literal['Sanitation', 'Pothole', 'Streetlight', 'Water Leakage', 'Other']
    urgency: Literal['Low', 'Medium', 'High']
    assigned_department: Literal['Sanitation', 'Public Works', 'Electrical', 'Water Board',
'General']

    # --- User-Provided Fields ---
    original_text: str | None = Field(None, max_length=500)
    image_url: str | None = None
    location: GeoJSONLocation

    # --- System Fields ---
    status: Literal['Submitted', 'In Progress', 'Resolved'] = 'Submitted'
    created_at: datetime = Field(default_factory=datetime.utcnow)
    updated_at: datetime = Field(default_factory=datetime.utcnow)
```

3. The API Contract: Detailed Routes & Signatures

This is the strict contract between the Frontend and Backend teams. All timestamps are UTC in ISO 8601 format.

Method	Endpoint	Description	Auth Req?	Request Body	Success Response (2xx)	Error Response (4xx)
POST	/api/auth/signup	Register a new citizen.	No	{ "email": "x@y.com",	201 Created: {	400 Bad Request, 409

				"password": "pw" }	"access_token": "...", "token_type": "bearer" }	Conflict (user exists)
POST	/api/auth/login	Log in a user.	No	{ "email": "x@y.com", "password": "pw" }	200 OK: { "access_token": "...", "token_type": "bearer" }	401 Unauthorized
POST	/api/reports/smart-create	Submit a new issue via the AI Brain.	Citizen	FormData: text (optional, string), image (optional, file), latitude (required, float), longitude (required, float)	201 Created: { "message": "Report submitted successfully", "report": ReportInDB }	400 Bad Request
GET	/api/reports/nearby	Get recent reports for the citizen feed.	Citizen	Query Params: lat (float), lon (float)	200 OK: [ReportInDB, ...]	400 Bad Request
GET	/api/reports/my-reports	Get reports for the logged-in	Citizen	None	200 OK: [ReportInDB, ...]	401 Unauthorized

		user.				
GET	/api/admin/reports	Get reports for the admin map.	Admin	Query Param: department (string, one of the assigned_department literals)	200 OK: [ReportInDB, ...]	401 Unauthorized, 403 Forbidden (not admin)
PATCH	/api/admin/reports/{report_id}	Update a report's status.	Admin	{ "status": "In Progress" }	200 OK: { "message": "Status updated", "report": ReportInDB }	401 Unauthorized, 404 Not Found

4. Environment & Tooling

- **Version Control:** Git. One monorepo with frontend-citizen, frontend-admin, and backend directories.
- **Package Management:** npm for frontend, pip with requirements.txt for backend.
- **Environment Variables:** All secrets (DB connection string, JWT secret, AI API key) will be managed via .env files. A .env.example file will be provided in the backend directory.

5. The Execution Plan: Per-Person Task Breakdown

Supriyo (Admin Portal Lead - Frontend)

- **Goal:** Deliver the complete, functional admin-facing dashboard.
- **Tasks:**
  1. **Project Setup:** Initialize a Next.js project in frontend-admin.
  2. **Build Admin Login:** Create the /login page. Connect it to the /api/auth/login endpoint.
  3. **Integrate Map Library:** Install and configure React Leaflet. The main dashboard page (/) will be the map.
  4. **Implement Department Filter:** Create the Department dropdown component.
  5. **Implement Map Data Layer:** Create a custom hook that fetches data from

/api/admin/reports based on the selected department filter. Handle loading and error states.
6. **Render Map Markers:** The map component will use the data from the hook to render Marker components, color-coded based on report.status.
7. **Build Detail Page:** Create the dynamic route /reports/[reportId]. This page will fetch and display report data and include the status update dropdown and PATCH request logic.

Sahiti (Citizen App Lead - Frontend)

- **Goal:** Deliver the complete, functional citizen-facing application.
- **Tasks:**
  1. **Project Setup:** Initialize a Next.js project in frontend-citizen.
  2. **Build Auth UI & Service:** Create the /login and /signup pages and the services/api.js module to handle API calls and JWT storage.
  3. **Build Submission UI:** Create the main reporting page, handling form state and the navigator.geolocation API.
  4. **Build Feed UI:** Create the main / page. On load, fetch the user's location, call the /api/reports/nearby endpoint, and render the results into styled "Card" components.
  5. **Build "My Reports" UI:** Create the /my-reports page that calls /api/reports/my-reports and renders a simple list.

Anitej (AI & Core Logic Lead - Backend)

- **Goal:** Implement the intelligent core of the application—the AI-powered submission endpoint.
- **Tasks:**
  1. **FastAPI Project Setup:** Initialize the FastAPI project in backend. Implement the Pydantic models from the Data Contract in models.py.
  2. **Define Tool Schema:** Create a Python dictionary that strictly defines the submit_civic_issue_report tool for the AI API.
  3. **Implement /smart-create Endpoint:** Implement the endpoint to accept FormData, construct the multimodal prompt, call the AI service, parse the tool_call response, and validate the data.
  4. **Implement Image Saving:** Write a utility function to save the uploaded image to a /static/images directory and return its URL.
  5. **Integrate with Database:** Use the create_report function from the database team to save the final ReportInDB object to MongoDB.

Vyaswanth (APIs & Auth Lead - Backend)

- **Goal:** Implement all supporting REST endpoints and the complete security layer.
- **Tasks:**
  1. **Implement Authentication:** Create auth.py with password hashing (passlib) and JWT (python-jose) logic. Build the /api/auth/signup and /api/auth/login endpoints.
  2. **Implement Security Dependencies:** Create FastAPI dependencies

(get_current_user, get_current_admin_user) to protect routes.
3. **Implement Citizen Endpoints:** Build /api/reports/nearby and /api/reports/my-reports, using the queries provided by the database team.
4. **Implement Admin Endpoints:** Build /api/admin/reports (with department filter) and the PATCH endpoint /api/admin/reports/{report_id}.

DSP (DB Architecture & Setup Lead)

- **Goal:** Design and provision a robust, performant database infrastructure.
- **Tasks:**
  1. **Provision MongoDB Atlas:** Create the cluster, database, and collections.
  2. **Configure Access:** Create a database user and securely provide the connection string to the backend team.
  3. **Define and Create Indexes:** Execute the createIndex commands for location, assigned_department, and user_id.
  4. **Seed Initial Data:** Write and run a seed.py script to insert the first admin user.
  5. **Create DB Connection Module:** Write the database.py file for the backend that handles the connection to MongoDB.

Lasya (DB Query & Integration Lead)

- **Goal:** Write and deliver optimized, production-ready queries for every data-fetching operation.
- **Tasks:**
  1. **Write Geospatial Query:** Provide the exact MongoDB query for the /api/reports/nearby endpoint to Vyaswanth. It must use $nearSphere.
  2. **Write Admin Filter Query:** Provide the exact query for /api/admin/reports with the department filter to Vyaswanth.
  3. **Write All CRUD Functions:** In a crud.py file for the backend, write the specific Python functions for every database operation: get_user_by_email, create_user, create_report, get_reports_by_user_id, etc.
  4. **Data Validation:** Act as Quality Assurance for the database. As Anitej tests the /smart-create endpoint, check the reports collection in Atlas to verify that every field is being inserted correctly.