AuraX Project Documentation

1 Overview

Aura-AI is a Real-Time City Vibe Tracker that decodes the emotional heartbeat of cities worldwide using AI, social media, news, and weather signals. It provides insights into how different cities are “feeling” at any given time and helps users explore moods, trends, and vibes around the globe. The project consists of a FastAPI backend for data fetching, analysis, and API services, and a React-based frontend for user interaction and visualization.

The system collects data on weather, news, tweets, and trending topics for major cities (e.g., Chennai, Mumbai, Bangalore, Delhi, etc.), computes mood summaries using sentiment analysis, generates headlines, and stores data in a database. It supports features like live city vibes with mood scores, emojis, weather details, and trending topics.

Git Repository Link: https://github.com/Umang7198/Aura-Ai

2 Team Members

- **Umang Chaudhary** (Backend Developer) - 21f3001035
- **Sanjay B** (Backend Developer) - 22f3001023
- **Pragati Sethi** (Machine Learning Developer) - 21f2001021
- **G Raghul** (Machine Learning Developer) - 21f2001093
- **Sahil Raj** (Frontend Developer) - 23f3001764
- **Tarunpreet Singh** - 23f1000113

3 Technologies Used

3.1 Backend

- **Framework**: FastAPI (for building APIs)
- **Programming Language**: Python
- **Database**: SQLite (via `aura_data.db` for storing city data)
- **Data Processing & AI**:
  - Services for data collection (news, tweets, weather via APIs like Open- WeatherMap, NewsAPI, or custom scrapers)
  - LLM integration (e.g., for sentiment analysis and headline generation, possibly using libraries like Hugging Face Transformers or OpenAI API)
  - RAG (Retrieval-Augmented Generation) service for enhanced AI responses
- **Dependencies**: Listed in `requirements.txt` (e.g., FastAPI, Uvicorn, SQLAlchemy for DB, requests for API calls, NLTK/VADER for sentiment)
- **Other**: `.env` for environment variables (API keys), Jupyter Notebook (`notebook.ipynb`) for experimentation

3.2 Frontend

- **Framework**: React (with React Router for routing)
- **Styling**: Tailwind CSS (for responsive, glassmorphism UI with gradient hover effects)
- **Storage**: LocalStorage (for persisting vibe and mood data)
- **Other**: npm for package management

4 Backend

4.1 Description

The backend service for Aura.AI, built with FastAPI, provides APIs for fetch- ing, analyzing, and serving city vibes data, including weather, news sentiment, tweets, trending topics, and mood summaries.

4.2 Tech Stack

- FastAPI for API development
- Python for core logic
- SQLite database for storage
    - External APIs/Services: Weather (e.g., OpenWeatherMap), News (e.g., News- API), Social Media (e.g., Twitter/X API for tweets)
    - AI/ML: Sentiment analysis (VADER), LLM for headline generation (via `llm_service.py`), RAG for advanced querying (`rag_service.py`)

Data is collected from

- Weather data from WeatherAPI
- News data from newsdata.io
- Twitter tweets from RapidAPI

4.3 Flow

1. **Data Collection**: Use `data_collector.py` to fetch weather, news, and tweets for predefined cities. Coordinates are hardcoded for accuracy.
2. **Sentiment Analysis**: Compute compound sentiment scores for news and tweets using libraries like VADER.
3. **Mood Summary**: Aggregate average sentiment, assign mood labels (e.g., Positive, Negative), and confidence scores.
4. **Storage**: Save raw data and summaries to SQLite DB (`aura_data.db` via

`database.py`).

5. **Headline Generation**: Use LLM to create engaging, Gen-Z style headlines based on mood and trends.
6. **API Serving**: Expose endpoints for fetching data, generating headlines, and mood forecasts.

4.4 APIs

- **POST /api/cities/fetch-and-save**: Fetch, save, and return raw data for all cities.
  - **Description**: Collects data (weather, news, tweets), saves to DB, and returns raw content for headline generation.
  - **Parameters**: None
  - **Response**: JSON with status, message, and array of city data objects (including coordinates, weather, news, tweets, trending topics, mood summary).
  - **Example Curl**:

```
curl -X 'POST' \
  'http://127.0.0.1:8000/api/cities/fetch-and
  -save' \
-H 'accept: application/json' \
-d ''
```

  - **Sample Response**: (Truncated) Includes city-specific data like weather metrics, news articles with sentiments, tweets, etc.
- **POST /api/headlines/generate-batch**: Generate headlines for multiple cities from fetched data.
- **POST /api/headlines/future_mood_forecast_batch**: Predict future moods based on trends.
- **GET /api/cities/mood**: Retrieve current mood data for all cities.

Interactive docs available at `/docs` (Swagger) or `/redoc` when server is running.

4.5 Getting Started (Backend)

1. **Create a Virtual Environment** (using uv):

   ```
   uv venv venv
   ```

   Activate:

   - Linux/macOS: `source venv/bin/activate`
   - Windows (PowerShell): `.\venv\Scripts\activate`

2. **Install Dependencies**:

   ```
   uv pip install -r requirements.txt
   ```

3. **Run the Server**:

   ```
   uvicorn main:app --reload --port 8000
   ```

   Access API docs at http://127.0.0.1:8000/docs.

4.6 Project Structure (Backend)

```
AURA DATA [WSL: UBUN...]
├── __pycache__
├── models
│   ├── __pycache__
│   ├── schemas.py % Data models/schemas
├── services
│   ├── __pycache__
```

```
│   ├── data_collector.py % Main data fetching logic
│   ├── database.py % DB connections and operations
│   ├── llm_service.py % LLM integration for analysis/headlines
│   ├── rag_service.py % Retrieval-Augmented Generation
├── venv % Virtual environment
├── .env % Environment variables (API keys)
├── .gitignore
├── requirement.txt.swp
├── aura_data_20250813_234214.json % Sample data files
├── aura_data_20250813_234826.json
├── aura_data.db % SQLite DB
├── main.py % FastAPI app entry point
├── notebook.ipynb % Jupyter for testing/exploration
├── requirements.txt % Dependencies
```

5 Frontend

5.1 Description

Aura-AI is the user-facing web app that displays city vibes, moods, and trends in an interactive, responsive interface.

5.2 Features

- **City Vibes**: Live emotional sentiment with mood score & emoji.
- **Weather Data**: Temperature, humidity, wind speed, and condition.
- **Trending Topics**: What's buzzing in each city.

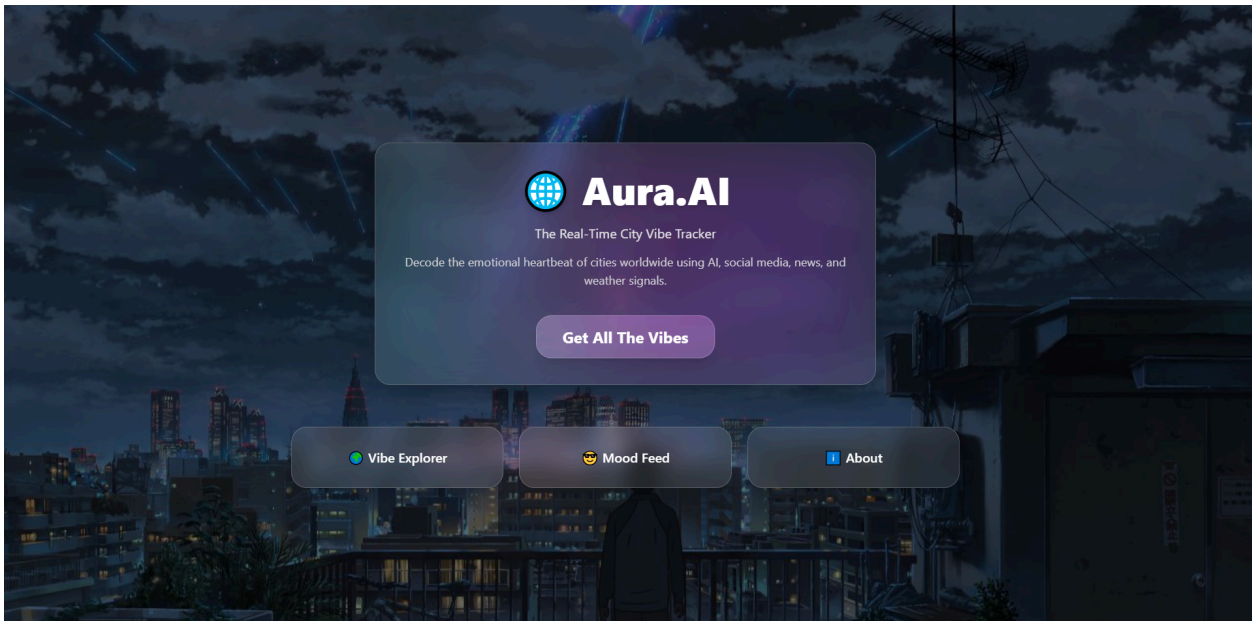
• **Vibe Explorer**: Detailed metrics for individual cities.

• **Mood Feed**: Quick overview of vibes across multiple cities.

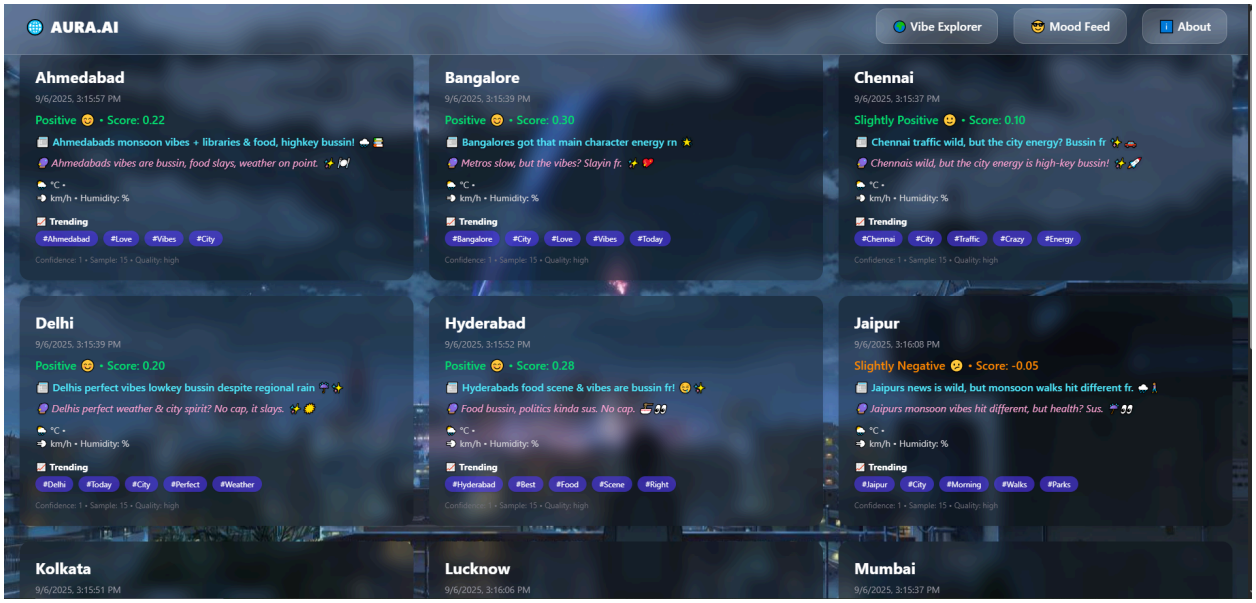
• **About**: Project info and mission.

5.3 Screenshots

• **Homepage**: Fetch and save vibes (button toggles if data exists).



• **Mood Feed**: Grid of mood cards with labels, scores, headlines, weather, and topics.



5.4 Tech Stack

- React + Tailwind CSS
- Routing: React Router
- Storage: LocalStorage (for persisting vibe and mood data)
- UI/UX: Glassmorphism, gradient hover effects, responsive design

5.5 Getting Started (Frontend)

1. **Install Dependencies**:

```
cd frontend

npm install
```

2. **Run the App**:

```
npm run dev
```

App available at `http://localhost:5173`.

5.6 Project Structure (Frontend)

```
src/
  ├── Navbar.jsx       % Navigation bar
  ├── Mood.jsx         % Mood Feed page
  ├── City.jsx         % Vibe Explorer page
  ├── AboutUs.jsx      % About page
  ├── App.jsx          % Main landing page
  ├── assets/          % Backgrounds & images
  └── main.jsx         % Entry point
```

5.7 How It Works

• On homepage: Click “Get All The Vibes” (or “Update The Vibes” if data exists) to fetch data via backend API.

• Data stored in LocalStorage (`aura_fetch_data`, `aura_mood_data`).

• Navigate to Mood Feed or Vibe Explorer for city views.

• Redirects to the homepage if data is missing.

6 Future Improvements

• Live API integration for real-time vibes.

• Advanced analytics (mood trends over time).

• Interactive world map with city vibes.

• Mobile PWA support.

7 Video

Video Link - Click Here

8 Conclusion

This documentation provides a high-level overview; refer to the code and API docs for implementation details. For issues or contributions, check the Git repository.