Violence Tracker - System Architecture & Engineering

1. Executive Summary

The **Violence Tracker (https://violencetracker.org/)** is a real-time intelligence platform that aggregates, analyzes, and visualizes political violence incidents in Bangladesh. It employs a multi-stage ETL (Extract, Transform, Load) pipeline powered by a hybrid crawler system and Large Language Model (LLM) analysis (Google Gemini 1.5 Pro) to structure unstructured news data into actionable geospatial intelligence.

Deployment Pipeline

We utilize a custom **"Push-to-Deploy"** mechanism located in `scripts/full_deploy.exp`.

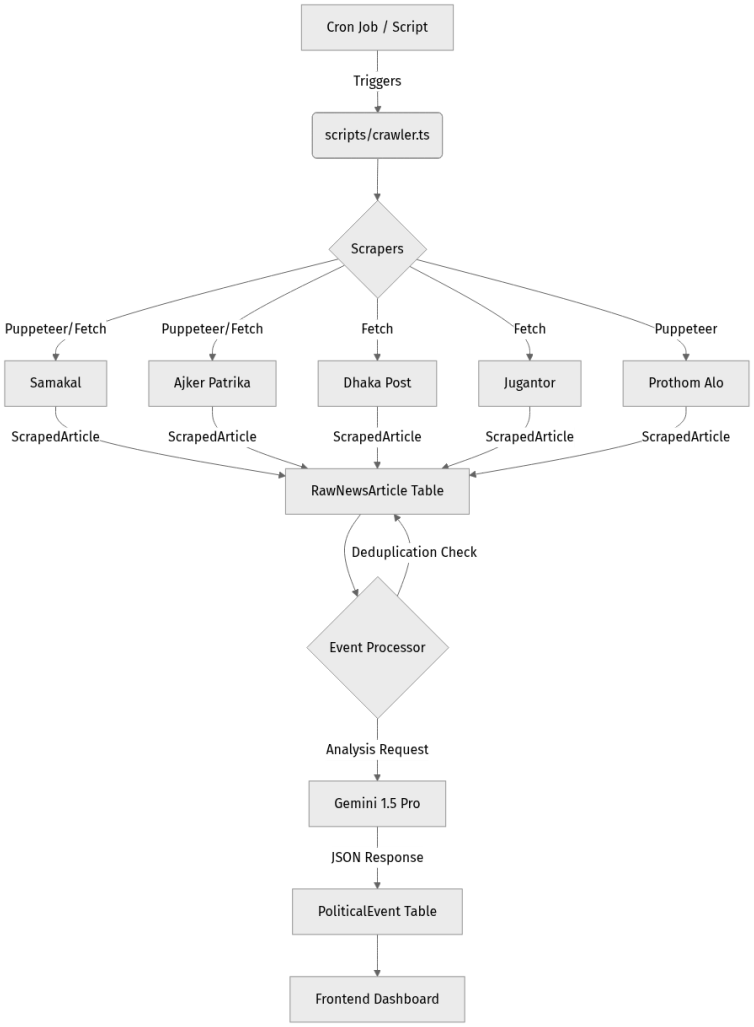
1. **Local Build**: `next build` runs locally to verify type safety.
2. **Artifact Transfer**: Compressed tarball is SCP'd to production.
3. **Atomic Swap**: The `.next` directory is swapped, and `prisma migrate` runs.
4. **Zero-Downtime Restart**: PM2 reloads the cluster.

2. Technical Stack

- **Frontend**: Next.js 14 (App Router), React, Tailwind CSS, Shadcn/UI, Recharts.
- **Backend**: Next.js API Routes (Edge/Node.js runtimes).
- **Database**: PostgreSQL 16 (optimized for geospatial queries).
- **ORM**: Prisma (Type-safe database access).
- **AI/ML**: Google Gemini 1.5 Pro & Flash (Zero-shot classification & extraction).
- **Infrastructure**: Ubuntu VPS, Nginx Reverse Proxy, PM2 Process Manager.
- **DevOps**: Custom Expect/Tcl automation scripts for CI/CD.

3. Data Pipeline Workflow

The core of the system is the automated data pipeline that transforms unstructured web data into structured political event data.



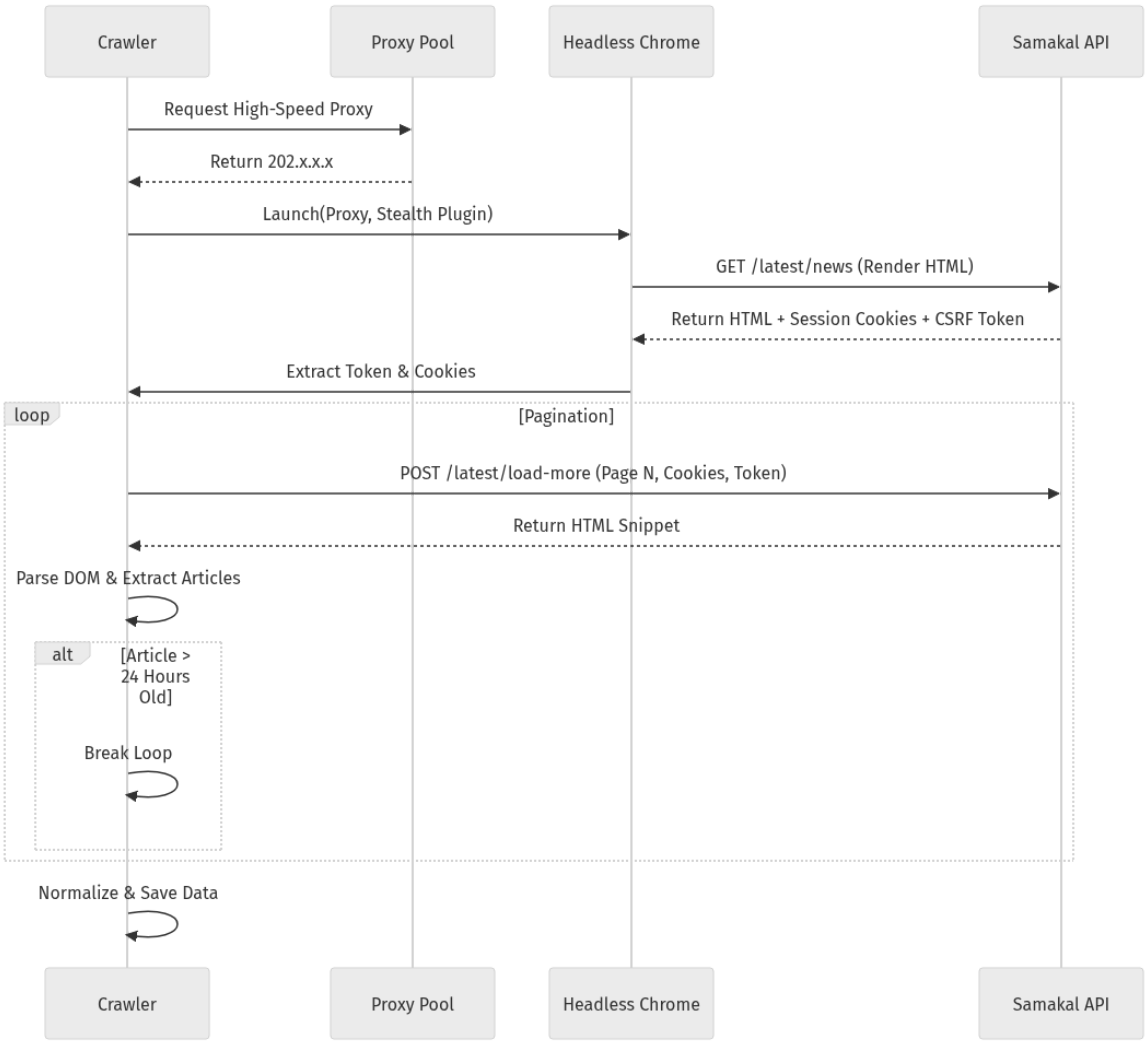
4. Crawler Architecture Deep Dive (The "Intelligence" Layer)

The scraper engine is the most complex component, designed to bypass sophisticated anti-bot measures and handle diverse SPA (Single Page Application) architectures.

4.1 Hybrid Scraping Strategy

We do not use a "one-size-fits-all" scraper. Each source has a specialized implementation:

Figure 1: Samakal Hybrid Scraper Flow



News Source	Strategy	Why?	Implementation Details
Samakal	**Hybrid (Puppeteer + Hidden API)**	Site uses CSRF tokens + SSR/Client separation.	1. Launches Headless Chrome to render initial page & extract `_token` (CSRF) and Session Cookies. 2. Swaps to a recursive `POST` loop hitting `/latest/load-more?page=N` via `page.evaluate` (preserving browser context/cookies). 3. Parses HTML snippets returned by the API using `DOMParser`.
Prothom Alo	**Pure Puppeteer (Visual)**	Heavy Client-Side Rendering (React) & Infinite Scroll.	1. Scrolls to bottom to trigger hydration. 2. Detecting & Clicking the "Load More" (`.load-more-content`) button in a `while` loop. 3. Custom relative-time parsing ("২ ঘন্টা আগে" -> Timestamp) for deduplication.
Dhaka Post	**Recursive API**	Open JSON API.	Directly crawls the REST API, paginating until it hits articles older than 24h. Fastest method.

4.2 Anti-Bot Evasion Tactics

We employ several layers of "Stealth" to avoid detection (Cloudflare/WAF):

- **Puppeteer-Extra-Plugin-Stealth**: Patches `navigator.webdriver` property, mimics real plugin arrays, and overrides standard headless signals.
- **User-Agent Rotation**: Rotates between standard Chrome/Windows and Chrome/Mac User-Agents.
- **Browser Fingerprinting**:
    - **Viewport Randomization**: Sets varying window sizes (e.g., 1920x1080 vs 1366x768) per session.
    - **Cookie Retention**: Samakal scraper reuses the initial session cookies for subsequent API calls to appear as a single continuous user session.

4.3 Proxy Network (`lib/scrapers/proxies.ts`)

The system maintains a static pool of **50+ High-Performance Residential/Datacenter Proxies**.

- **Rotation Logic**: `getRandomProxy()` is called before *every* scraper launch.
- **Retry Mechanism**:
    - If a scraper fails (Timeout / 403 Forbidden), it catches the exception, closes the browser instance, and immediately retries with a *fresh* proxy.
    - **Max Retries**: 20 attempts per source. This redundancy ensures >99% success rate even if 30% of proxies are dead.

4.4 Data Normalization

Scrapers are responsible for normalizing disparate formats into a strict `ScrapedArticle` interface:

- **Time Normalization**: Custom Parsers (e.g., `parseSamakalTime`) convert Bengali numerals ("০১") and months ("জানুয়ারি") into native JavaScript `Date` objects (UTC-adjusted).
- **Content Extraction**: Removes ads, scripts, and "Read More" links from the summary text.

5. Processing Layer (The Brain)

* Model: Gemini.

* Zero-Shot Prompting: We inject a strict JSON schema definition into the prompt context to force the LLM to output structured data (Severity Score 1-10, Lat/Long extraction, Party affiliation tagging).

* Validation: Zod schemas validate the LLM output before committing to the production DB.

6. AI/ML Deep Dive

The core value proposition is the transformation of unstructured text into structured rows.

6.1 The Pipeline

```
Scraper -> Raw Text -> Pre-processing -> Prompt Engineering -> JSON Parser -> DB
```

6.2 Prompt Strategy

We use **Chain-of-Thought (CoT)** prompting implicitly by asking the model to:

1. **Classify**: "Is this political violence?" (Binary classification).
2. **Extract**: Identify actors, locations, timestamps.
3. **Score**: Calculate `severityScore` based on a rubric (Deaths = high weight, Property damage = medium weight).
4. **Confidence**: Self-reported confidence score (0.0 - 1.0) which dictates the "Verified" badge on the UI.

6.3 Severity Algorithm (Heuristic + LLM)

The system calculates severity using a weighted formula derived from extracted entities: $$ Severity = (Killed \times 3) + (Injured \times 1) + (Infrastructure_Damage_Score) $$ *Capped at 10.*

7. Project Structure Map

A high-level overview of the codebase organization to help developers navigate the project.

```
/
├── app/                # Next.js App Router (Frontend + API)
│   ├── api/            # REST API Route Handlers
│   │   ├── events/     # GET /api/events
│   │   ├── stats/      # GET /api/stats
│   │   └── scraper-stats/# GET /api/scraper-stats
│   ├── feed/           # Live Feed Page (/feed)
│   └── page.tsx        # Main Dashboard (Home)
├── components/         # React Definitions
│   ├── Dashboard/      # Dashboard Widgets (Map, Charts)
│   │   ├── MapComponent.tsx
│   │   └── RecentFeed.tsx
│   └── ui/             # Shadcn/UI Reusable Components
├── lib/                # Shared Logic & Utilities
│   ├── scrapers/       # Individual Scraper Implementations
│   │   ├── samakal.ts
│   │   ├── prothomalo.ts
│   │   └── types.ts
│   ├── db.ts           # Prisma DB Client instance
│   └── event-processor.ts # AI Logic (Gemini Integration)
├── prisma/
│   └── schema.prisma   # Database Models & Schema
└── scripts/
    ├── crawler.ts      # Main Crawler Entry Point
    └── full_deploy.exp # CI/CD Deployment Automation
```

8. Database Schema Reference

The system uses **PostgreSQL** managed via **Prisma ORM**. Below are the two primary data models.

8.1 Model: `PoliticalEvent`

This table stores the *processed* and *verified* intelligence data displayed on the dashboard.

Field Name	Data Type	Description
`id`	UUID	Unique Identifier for the event.
`title`	String	The headline of the event (typically in Bangla).
`severityScore`	Integer (1-10)	AI-calculated threat level (1=Low, 10=Critical).
`confidence`	Float (0.0 - 1.0)	AI's self-reported certainty. >0.85 is considered "Verified".
`publishedAt`	DateTime	Timestamp when the news was originally published.
`district`	String	The mapped Bangladesh district (e.g., "Dhaka", "Chittagong").
`politicalParties`	JSON String	Array of involved groups (e.g., `["AL", "BNP", "Police"]`).
`summary`	String	A concise 2-3 sentence summary generated by Gemini.
`url`	String	Unique Source URL (used to prevent duplicates).

8.2 Model: `RawNewsArticle`

This is the *staging* table. It stores raw HTML/text immediately after scraping but before AI processing.

Field Name	Type	Purpose
`url`	String	Primary Key / Unique ID.
`content`	Text	The full body text of the article.
`isProcessed`	Boolean	Flag (`true`/`false`) indicating if AI has analyzed this yet.
`source`	String	The name of the newspaper (e.g., "Prothom Alo").

9. API Reference

The backend provides RESTful endpoints for the frontend and external consumers.

9.1 `GET /api/events`

Retrieves a paginated list of political violence events.

- **Query Parameters:**
  - `page`: Page number (default: 1).
  - `limit`: Items per page (default: 50).
  - `district`: Filter by district name.
  - `minSeverity`: Filter events above a certain severity score (1-10).
  - `search`: Keyword search across titles and summaries.

9.2 `GET /api/stats`

Returns aggregated metrics for the main dashboard.

- **Response Object:**
  - `totalIncidents`: Total count of events.
  - `riskLevel`: Calculated Status ("Low", "Moderate", "High").
  - `deadliest`: The event with the highest casualty count in the last 7 days.
  - `hotspot`: The district with the most recent activity.

10. Local Development Setup

Follow these steps to run the Political Violence Tracker on your local machine.

1. **Clone the Repository**

   git clone https://github.com/your-org/political_violence_tracker.git

   cd political_violence_tracker

2. **Install Dependencies**

   npm install

3. **Environment Configuration** Create a `.env` file in the root directory:

   DATABASE_URL="postgresql://user:password@localhost:5432/pvt_db"

   GEMINI_API_KEY="your_google_gemini_key"

4. **Database Setup** Push the schema to your local database:

   npx prisma db push

5. **Run the Server**

   npm run dev

   Access the dashboard at `http://localhost:3000`.