Design and Implementation of a Geo-Aware Content Delivery Platform

Simone Tassi
Department of Computer Science and Engineering, University of Bologna, Italy
simone.tassi@studio.unibo.it

***Abstract*—This report presents the design and implementation of a distributed location-aware system for personalized content delivery. The proposed architecture employs geo-fencing techniques to define virtual geographic boundaries, triggering context-aware notifications and data delivery upon user entry or exit events. To improve performance and reduce latency, the system adopts an edge computing paradigm, in which a central geo-aware proxy orchestrates requests toward localized content-repositories deployed on a geo-distributed infrastructure.**

Particular attention is given to privacy, addressed through location perturbation mechanisms that allow users to mask their exact coordinates while preserving service functionality. In addition, the system addresses availability and scalability through containerized microservices and Kubernetes based orchestration, incorporating geo-aware routing, load balancing, and failover mechanisms across edge nodes. The evaluation analyzes the trade off between privacy perturbation and the quality of service (QoS) in geofence event detection.

***Index Terms*—geo-fencing, edge computing, content delivery, location-aware systems, privacy, load-balancing**

I. Introduction

A. Context-Awareness in Modern Environments

The evolution of ubiquitous computing has significantly influenced how digital systems interact with the physical environment. **Context-Aware Systems** (CAS) extend beyond basic location tracking and are now widely used in domains such as **Retail**, **Smart Cities**, and **Tourism**. In these scenarios, the ability to trigger digital content based on physical proximity, commonly referred to as *geo-fencing*, enables localized services such as delivering museum guides to tourists or traffic alerts to citizens.

However, deploying these systems at scale introduces several technical challenges, including latency, bandwidth consumption, and user privacy. Centralized cloud architectures may struggle to deliver high bandwidth media content (e.g., videos or brochures) with the responsiveness required in proximity based interactions. Moreover, continuous location tracking raises privacy concerns that must be addressed at the architectural level.

B. Project Objectives

This project addresses these challenges by proposing a distributed architecture with integrated privacy mechanisms. Although the functional prototype is developed for a retail scenario, the system is designed to be domain agnostic and applicable to other location based services.

The main technical objectives are:

- **Edge-Based Content Delivery:** To mitigate latency limitations of centralized infrastructures, the system adopts a distributed topology. A central proxy manages lightweight metadata, while high bandwidth content is served by containerized **Edge Nodes** located closer to the user, improving the Quality of Service (QoS).
- **Robust Geo-fencing:** The development of a mobile client capable of background monitoring and accurate detection of *entry* and *exit* events within polygonal geofences, while limiting resource consumption.
- **Privacy by Design:** The integration of algorithmic privacy mechanisms, specifically location perturbation, to reduce the exposure of precise user trajectories while maintaining system functionality.
- **Data-Driven Analytics:** The inclusion of a backend analytics component that applies clustering techniques to spatial data in order to identify engagement patterns and high activity zones without exposing individual user information.

C. Document Structure

The remainder of this report is organized as follows: Section II describes the logical architecture and data modeling strategies. Section III details the technology stack, including the backend, mobile client, and orchestration layer. Finally, Section IV presents the experimental evaluation, focusing on the trade off between privacy perturbation and service accuracy.

II. Architecture Design

The system is designed as a distributed, multi-tier ecosystem composed of four specialized functional blocks. This logical separation ensures clear decoupling between administrative management, content storage, user interaction, and spatial processing. The architecture follows a Proxy–Edge pattern, in which high level orchestration is centralized while data intensive delivery is delegated to edge nodes.

A. System Components

- **Centralized Backend:** This component acts as the central orchestration layer of the platform. It represents the authoritative source of system state, managing CRUD operations for spatial entities (geofences) and user identities. In addition to metadata management, it functions as a spatial proxy: it stores lightweight content descriptors

but does not serve raw media files directly. The backend also hosts the analytics engine responsible for processing spatial events and generating aggregated usage insights.

- **Edge Content Repositories:** To reduce latency and optimize bandwidth utilization, the architecture incorporates multiple distributed storage nodes. These repositories are deployed in a geo-distributed fashion, positioned close to the physical locations where geofences are active. Each repository manages the lifecycle of high bandwidth assets (images, videos, PDFs) and exposes dedicated endpoints for secure file streaming and Time-To-Live (TTL) metadata management.
- **Administrative Frontend:** This web based dashboard provides an interface for system administrators to manage spatial data and monitor system activity. It enables the interactive definition of polygonal geofences on geographic maps and supports real time engagement monitoring through live telemetry and heatmap visualizations.
- **Mobile Client:** The mobile application represents the primary interaction layer between users and the system. It autonomously monitors the user's geographic position using a hybrid precision strategy: lightweight proximity checks are performed for energy efficiency, while high precision polygon containment checks are activated only when necessary. The mobile client also integrates the privacy preserving logic, allowing users to control the granularity of the location data shared with the backend.

The logical relationships between these components are formalized in the database schema. As illustrated in Figure 1, the data model associates *Users* with *Events* and links *Geofences* to *Content Metadata*. In this structure, the geofence entity acts as the contextual bridge between user location events and the content delivered by edge repositories.

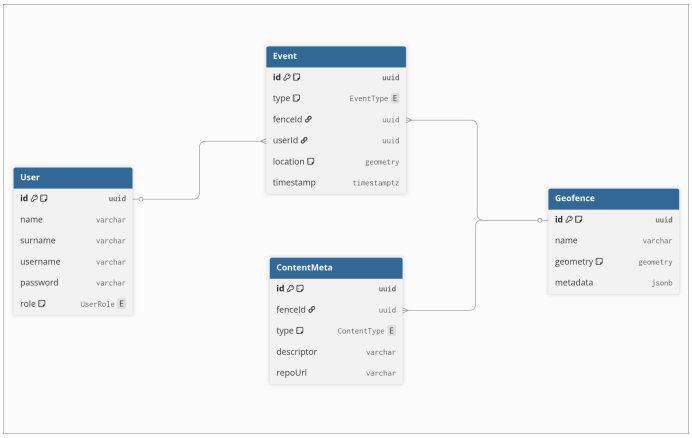


Fig. 1: Entity-Relationship Diagram showing the logical connections between Users, Spatial Events, Geofences, and Content Metadata.

B. Deployment Strategy and Information Flow

The deployment architecture is designed to support a scalable, location-aware infrastructure in which the physical placement of services directly influences Quality of Service (QoS).

The infrastructure establishes a continuum between a Central Core and multiple Edge Sites. The Core hosts the Backend and the primary database, ensuring consistency and providing a single administrative entry point. In contrast, Edge Sites are deployed at specific points of interest. Each Edge Site includes a Content Repository, effectively functioning as a localized media cache.

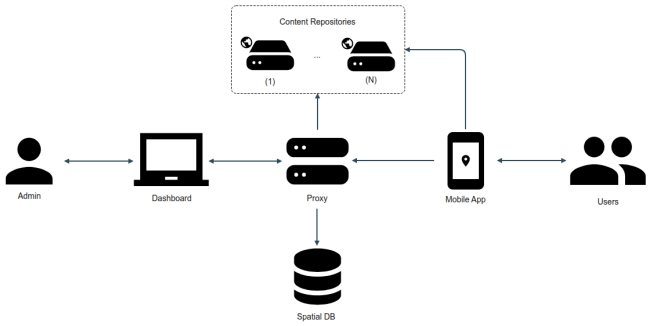


Fig. 2: High level system architecture illustrating the interaction flow between the different architectural components.

When the Mobile Client detects an entry event into a geofence, it queries the Central Backend. Acting as a geo-aware proxy, the backend determines the Edge Site associated with the geofence and redirects the client to the appropriate localized repository. This strategy provides several advantages:

1) **Traffic Localization:** High bandwidth file transfers occur over shorter network paths, reducing core network congestion.
2) **Modular Scalability:** New physical locations can be integrated by deploying additional Edge Nodes without modifying central orchestration logic.
3) **Improved Availability:** If the central orchestrator becomes temporarily unreachable, the mobile client can rely on locally cached data and proximity based logic to preserve basic functionality.

III. Implementation

The implementation of the platform follows a modular, microservice oriented approach designed to satisfy the requirements of a distributed location-aware system. This section describes the technical realization of the system, from the selected software stack to the internal logic of individual components.

A. Technology Stack Overview

The technology stack was selected to ensure strong typing, efficient spatial query processing, and cross platform compatibility for the mobile environment.

- **Centralized Orchestrator (NestJS):** The core orchestration layer is implemented using **NestJS**, a Node.js framework responsible for managing spatial metadata, user authentication, and real time event broadcasting via

WebSockets. It operates as a geo-aware proxy, coordinating client redirection toward localized resources.

- **Edge Content Repositories (Fastify & SQLite):** These nodes are implemented using **Fastify** to minimize overhead during high bandwidth media streaming. They manage localized file storage and implement TTL based lifecycle policies to preserve content freshness at the network periphery. To satisfy efficiency constraints inherent in edge deployments, the system uses **SQLite** for local metadata persistence. Its compact, serverless architecture enables rapid data access without the operational overhead of a centralized RDBMS.
- **Geospatial Database (PostGIS): PostgreSQL** serves as the primary data store and is extended with **PostGIS** to support geographic objects. This configuration enables native spatial operations, such as verifying whether a point is covered by a polygonal geofence using the ST_Covers function.
- **Mobile Development (React Native & Expo):** The mobile client is developed using **React Native** within the **Expo** ecosystem. This stack provides native support for background location tracking, task management, and persistent storage via *AsyncStorage*.
- **Frontend Dashboard (Angular & Leaflet):** The administrative interface is implemented using **Angular**, leveraging its reactive component based architecture. **Leaflet** is integrated as the mapping engine to support advanced spatial visualization, including real time heatmaps and interactive polygon editing.
- **Analytics & Logic: Python** is employed as a companion runtime environment to perform advanced data analysis, specifically clustering geofence engagement metrics using the K-Means algorithm.
- **Infrastructure (Docker & Kubernetes):** The ecosystem is containerized using **Docker** and orchestrated via **Kubernetes**. This enables deployment across an edge–cloud continuum by supporting stateful sets for localized content repositories deployed near physical landmarks.

B. Centralized Backend & Data Modeling

The Centralized Backend functions as the primary orchestration component of the system. It manages the lifecycle of spatial entities and coordinates state across the distributed environment. The service is implemented using **NestJS**, which provides a modular structure separating concerns into dedicated modules for geofencing, content metadata, user management, and real time events.

1) Spatial Schema and TypeORM Integration: To manage geographic data at scale, the backend integrates **TypeORM** with the **PostGIS** extension for PostgreSQL. Geographic entities are mapped directly to the application domain model using native `geometry` types, enabling the persistence of polygonal geofences.

The service layer leverages native PostGIS functions for topological operations. The `by-coords` lookup endpoint employs the ST_Covers operator to determine whether a user's latitude and longitude fall within a geofence. The data model defines a one-to-many relationship between `Geofence` and `ContentMeta`, with a `CASCADE` delete policy to ensure automatic removal of associated metadata when a geofence is deleted, thereby preserving referential integrity.

2) User and Session Management: The system implements an identity layer through the `UsersModule`, which distinguishes between standard mobile users and administrative accounts with elevated privileges. The `User` entity stores unique credentials and role based identifiers (e.g., ADMIN, USER), supporting registration and authentication workflows.

3) Data Validation: Strict request validation is enforced through a global `ValidationPipe` configured in `main.ts`. This ensures that all incoming Data Transfer Objects (DTOs) for user creation and login comply with predefined security and formatting constraints before being processed by the service logic.

4) Real time Event Bus: Real time visibility for the administrative dashboard is achieved through a WebSocket based broadcasting system. The implementation centers on an `EventsGateway` built with **Socket.io**, which manages persistent WebSocket connections from frontend clients.

When a mobile client records an entry, exit, or content-view event, the `EventsService` both persists the record for historical analysis and immediately pushes the payload to connected administrative clients. Events are categorized by type, enabling the dashboard to render live indicators and provide immediate feedback on system engagement.

5) Content Orchestration and Proxy Logic: Acting as a geo-aware proxy, the backend manages the association between physical locations and distributed assets without hosting large media files. Each `ContentMeta` record contains a `repoUrl` that identifies the Edge Content Repository storing the asset.

When content is requested, the backend returns a `302 Found` redirect response, routing the client to the appropriate edge node for optimized streaming. The service also exposes a filtering mechanism to retrieve only active geofences. This is achieved by querying edge metadata endpoints to verify content expiration status, ensuring that only valid content is served.

6) API Documentation and Standardization: To ensure interoperability across system components, the backend implements standardized documentation using the **OpenAPI** specification.

By integrating **Swagger** into the NestJS application, the system automatically generates a machine readable JSON specification of all endpoints, request schemas, response types, and validation constraints. This documentation acts as both a development reference and a formal contract between the orchestrator, mobile client, dashboard, and edge repositories, promoting consistent communication across the distributed environment.

C. Edge Content Repositories

The Edge Content Repositories are designed as autonomous storage nodes positioned at the network periphery to reduce latency and optimize high bandwidth content delivery. These services are implemented using **Fastify**, selected for its minimal overhead and efficient handling of asynchronous I/O operations.

1) High Performance Streaming and Lightweight Storage: To support multimedia streaming with low latency, the implementation uses **Node.js streams** and the *@fastify/multipart* plugin. This allows file data to be piped directly from local storage to the client socket without loading entire assets into memory.

Each repository uses **SQLite** for metadata persistence. Its serverless architecture enables rapid lookups and local autonomy without the administrative overhead of a traditional RDBMS.

2) Content Lifecycle and TTL Management: To preserve content freshness and optimize disk usage, the system implements a **Time-To-Live (TTL)** mechanism. When an asset is deployed to an edge node, an optional TTL header defines its validity period. The repository stores an expiration timestamp in SQLite and verifies validity upon each request.

If an asset has expired, the service returns a `410 Gone` status. The central backend can query edge nodes to determine geofence activity based on content availability, ensuring that users receive up to date notifications and media.

3) Distributed Resilience and Statelessness: Edge repositories are fully containerized and remain largely stateless relative to the central orchestrator. Operational data is maintained locally through SQLite and filesystem storage.

Upon startup, repositories automatically create required storage directories and database tables, ensuring consistent initialization across deployments. This design supports rapid provisioning of new edge sites through environment configuration and persistent volume mounting.

D. Mobile Client & Geofencing Logic

The mobile application represents the most critical frontier of the system, as it must maintain continuous situational awareness while operating under the strict energy and resource constraints of a handheld device. The implementation, built with **React Native** and **Expo**, centers on a background monitoring engine that balances location precision with battery preservation.

To support this continuous monitoring paradigm, the application requires a carefully designed onboarding process that ensures both user authentication and the explicit granting of the necessary location permissions. This initial configuration phase is essential to enable reliable background operation while maintaining transparency toward the user. The user onboarding flow, illustrating the authentication and permission granting process required for this monitoring, is shown in Figure 3.

1) Hybrid Monitoring and Precision Geofencing: To optimize power consumption, the app avoids constant high precision GPS polling. Instead, it utilizes a two tier **Hybrid Monitoring Strategy**. In the first tier, the system monitors large circular "buffer" zones around geofences using low power native region monitoring. When the user enters one of these buffers, the app triggers a second, high precision tier: a **Precision Tracking Task** that monitors for actual entry into the complex polygonal boundary. This polygon level detection is performed using the *@turf/turf* library, which executes geometric containment checks locally on the device. This approach ensures that the app only consumes significant resources when the user is in immediate proximity to a landmark.

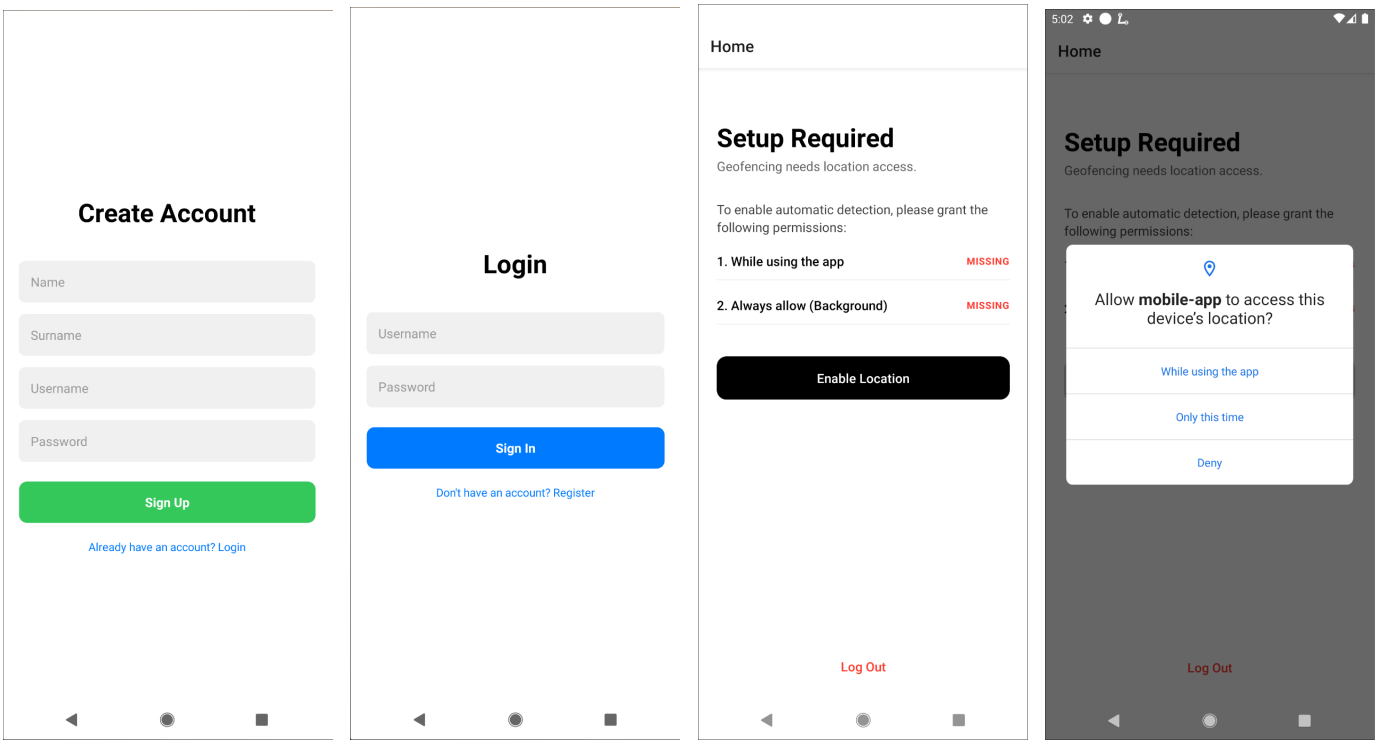
2) Background Tasks and Persistent State: The system's reliability depends on its ability to function even when the application is not actively in the foreground. This is achieved through the integration of **Expo's TaskManager**, which registers a dedicated background task to handle location updates and geofence events. When a transition (entry or exit) is detected, the task automatically triggers local notifications (see Figure 4b) and attempts to sync the event with the central backend. To ensure a seamless user experience, the app utilizes **Zustand** for state management and **AsyncStorage** for persistence, allowing the client to maintain a local "History" of events (Figure 4d) and cached content metadata even during periods of limited connectivity.

3) Privacy by Design - Location Cloaking: A key feature of the mobile implementation is the **Privacy Layer**, which allows users to control the spatial granularity shared with the backend. Through a toggle in the *HomeScreen* UI, users can enable "Location Privacy (Cloaking)". When active, the app utilizes the `applyCloaking` utility to snap precise coordinates to a discrete grid with a `GRID_SIZE` of 0.001 degrees. This process effectively truncates the meaningful precision to 3 decimal places, discarding any data beyond the thousandths place, to reduce accuracy to a radius of approximately 110 meters (at the equator). This perturbation ensures that the system delivers relevant localized content while masking the user's exact trajectory and street level position from the server.

4) Content Prefetching and Viewing: To enhance the perceived Quality of Service (QoS), the app implements an aggressive **Content Prefetching** logic. As soon as a user enters a geofence's circular buffer, the `useContentStore` begins downloading the associated assets (e.g., PDFs, videos) to a local cache. By the time the user enters the actual polygon and receives the notification, the content is already available for offline viewing. The app includes a robust `ContentViewerScreen` (Figure 4c) that utilizes native file handlers and intents to open various media types, ensuring that content delivery is both fast and platform consistent.

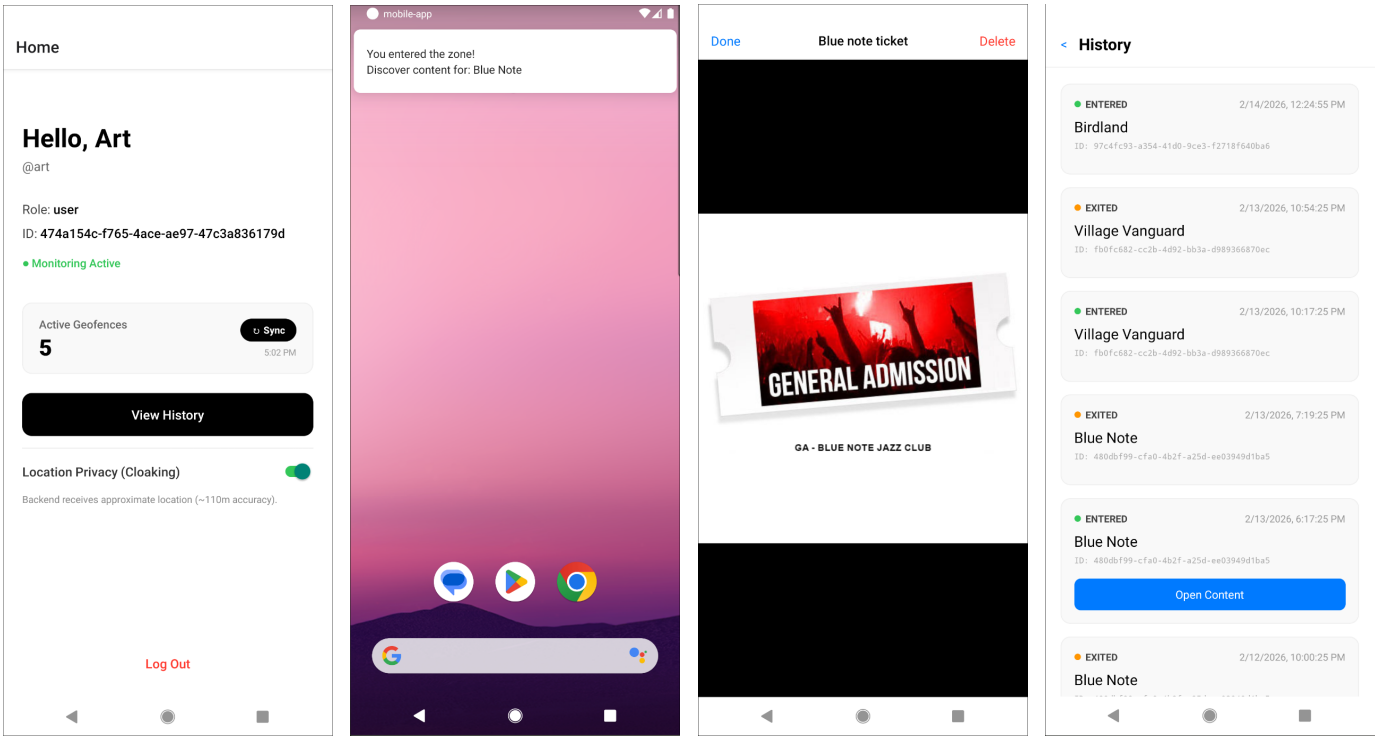
E. Administrative Dashboard

The administrative dashboard serves as the visual command center for the system, allowing for the real time management of geographic data and the analysis of user interactions.



(a) Registration (b) Login (c) Permission Request (d) OS Approval

Fig. 3: User onboarding and permission flow. Explicit OS approval (d) is critical for the background monitoring engine to function.



(a) Home & Privacy (b) Geofence Notification (c) Content Viewer (d) Event History

Fig. 4: Core application features. (a) Home screen with Location Cloaking toggle enabled. (b) System notification triggered by the background task upon geofence entry. (c) In app viewing of cached content. (d) Persistent history of user movements.

Developed with **Angular**, the application follows a strict **Smart/Dumb component architecture**, which ensures a clean separation between the complex spatial engine, the UI logic, and the reactive data layer.

1) Interactive Spatial Editing and Visualization: To manage polygonal geofences, the dashboard integrates **Leaflet** as its core mapping engine. The interface allows administrators to interactively draw, move, and reshape boundaries directly on the map through the **Geoman** plugin.

- **Spatial Synchronization:** A custom utility was implemented to parse Leaflet layers into strictly formatted **GeoJSON Polygons**, ensuring that the complex geometries created in the UI remain fully compatible with the PostGIS backend.
- **Reactive State Management:** The dashboard utilizes **RxJS BehaviorSubjects** within a centralized *GeofencesService*. This ensures that any update is instantly reflected across the UI panels and the map without requiring manual refreshes.
- **Visual Feedback:** To improve the operator's situational awareness, the system implements a "pulse" mechanism: when a live event is received via WebSockets, the corresponding geofence on the map visually pulses with color coded feedback (e.g., green for entry).

2) Real time Analytics and Heatmapping: The platform implements a dedicated analytics tray and dynamic map overlays to provide real time insights from collected spatial data (Figure 5). These tools enable the visualization of geographic trends and system engagement directly within the dashboard interface.

- **Advanced Visualization (Heatmaps & Clustering):** Utilizing the *leaflet.heat* plugin and server side aggregation, the dashboard renders live visualizations of user activity. This includes heatmaps for density analysis (Figure 5b) and spatial clustering (Figure 5c) to group high volume data points.
- **Engagement Metrics:** The dashboard includes a real time analytics panel (Figure 5d) that displays live traffic trends, top active areas, and event logs using **Chart.js**. This panel allows administrators to switch between "Live" monitoring and "Analysis" mode to view deeper metrics like conversion rates and session duration.

3) Secure Administrative Access: To protect the system's configuration and sensitive spatial data, a complete authentication flow was implemented. Access to the dashboard is restricted via an *authGuard* that verifies the user's role; only accounts with **ADMIN** privileges are permitted to access the management routes. The *AuthService* manages these sessions by exposing the current authentication state as an observable, ensuring a secure and consistent user experience across the management environment.

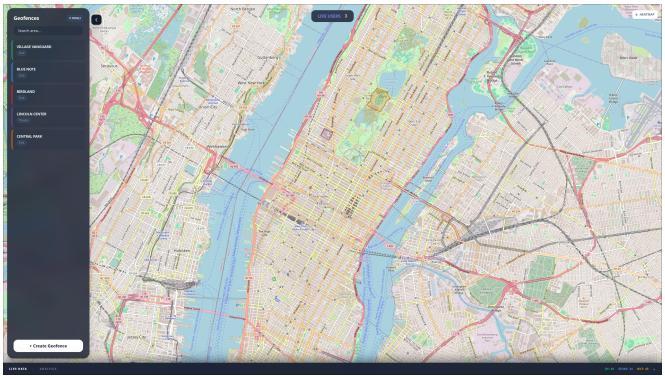
F. Analytics & Clustering Engine

A fundamental objective of this architecture is the transformation of raw, high volume geographic events into actionable insights, moving beyond simple data collection toward a truly data driven location-aware service. This is achieved through a dedicated analytics engine that bridges the real time capabilities of the **NestJS** backend with the statistical processing power of **Python**. By offloading complex computational tasks to a dedicated script environment, the system maintains high responsiveness while performing analysis of user behavior and spatial trends.

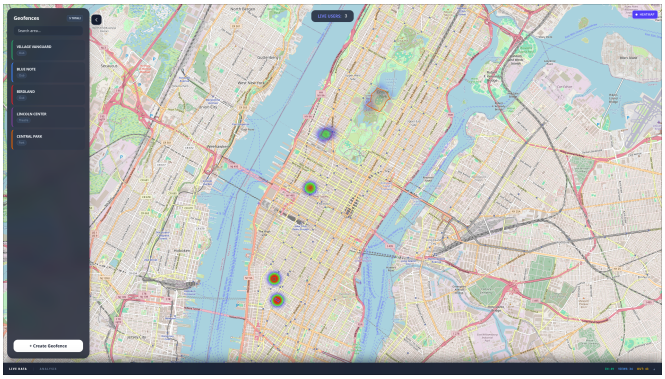
1) K-Means Clustering and Engagement Analysis: To identify patterns in user engagement and categorize geographic areas by their performance, the engine implements a **K-Means Clustering** algorithm. The implementation follows a "sidecar" execution pattern where the `AnalyticsService` orchestrates the lifecycle of a standalone Python script. During this process, the system aggregates geofence specific metrics to perform K-Means clustering, categorizing areas into distinct behavioral profiles. The model utilizes three primary features: *total entries*, *conversion rate* (defined as the ratio between physical entries and subsequent digital content views) and *dwell time* (which measures the average duration of user presence within the geofence perimeter). This multidimensional approach allows the system to differentiate between high traffic transit areas and high engagement zones based on both spatial and temporal user behavior. The clustering logic then processes these multi dimensional features to categorize each geofence into one of three clusters: **HOTSPOT** areas characterized by high traffic and engagement, **COLD** zones with low activity, or **STANDARD** zones. To support this hybrid execution in a production environment, the system's `Dockerfile` was specifically engineered to bundle the Python runtime and scientific libraries like *scikit-learn* alongside the primary Node.js application, ensuring a consistent environment for the analysis engine across the edge-cloud continuum.

2) Spatial Density and Heatmap Generation: Beyond high level clustering, the engine provides the granular data necessary for advanced visual analytics and spatial monitoring. The `AnalyticsModule` exposes specialized endpoints that process historical coordinate data into weighted point density arrays. This data serves as the foundation for the dashboard's dynamic heatmap, allowing administrators to visually identify high congestion zones and optimize the placement of virtual boundaries based on real world movement patterns. This approach directly addresses the need for a comprehensive visualization of system engagement without compromising the performance of the live tracking services.

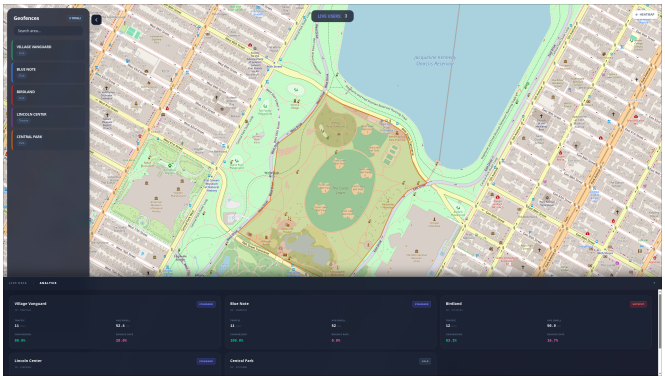
3) Privacy Simulation and Topological Consistency: Reflecting the project's core commitment to user privacy, the engine includes a specialized `PrivacyAnalysisModule` designed to evaluate the impact of location perturbation. This module allows administrators to run simulations that compare ground truth coordinates against "cloaked" or perturbed data, calculating specific error metrics and containment success rates. Such simulations are vital for determining the optimal trade off between user anonymity and service quality.



(a) Geofence Viewer



(b) Density Heatmap



(c) Event Clustering

(d) Analytics Metrics

Fig. 5: Administrative Dashboard capabilities. (a) Initial geofence view. (b) Heatmap visualization of user density. (c) Spatial clustering of high volume events. (d) Real time engagement metrics and charts.

G. Orchestration & Deployment

The final layer of the implementation focuses on the realization of an ”Edge-Cloud Continuum,” translating the abstract distributed architecture into a resilient and scalable infrastructure. This stage is designed to provide a geo-distributed environment capable of handling localized service delivery and intelligent traffic routing, ensuring the system remains responsive regardless of the physical distance between the user and the central orchestrator.

1) Containerization and Multi-Stage Image Optimization: To maintain environmental consistency across the entire lifecycle of the project, every component is encapsulated within **Docker** containers. A significant implementative choice was the adoption of **multi-stage Dockerfiles**. This approach allows for a clean separation between the resource intensive build environment and the final, light runtime image. During the build phase, heavy dependencies such as the TypeScript compiler, native SQLite build tools, and the Python scientific suite are utilized to generate artifacts; however, only the compiled binaries and productio grade modules are copied into the final execution stage. This strategy significantly reduces the footprint of the images, which is a critical factor for ensuring rapid deployment and updates in edge locations where large data transfers would otherwise degrade performance.

2) Kubernetes Topology and the Distribution of Edge Nodes: The production environment is managed through a **Kubernetes** orchestration layer, which ensures the stability and availability of the distributed nodes. A core architectural decision was the deployment of the PostGIS database and the localized Edge Content Repositories as **StatefulSets**. Unlike standard deployments, this guarantees that each node retains a stable network identity and a dedicated **PersistentVolumeClaim**. This persistence is critical for maintaining the integrity of the local **SQLite** metadata and physical media assets, as it ensures that any rescheduled pod automatically re-attaches to its specific data volume.

To effectively implement the Edge-Cloud hierarchy, the infrastructure utilizes **Node Affinity** and **Taints/Tolerations** to enforce strict workload placement. The ”Core” components, such as the PostGIS database and the NestJS Backend, are constrained to central cloud nodes via `nodeAffinity` rules, ensuring they reside on high performance hardware. Vice versa, Edge Content Repositories are deployed to specific geographic locations. By applying **Taints** to edge computing hardware, the orchestrator prevents generic system pods from consuming limited local resources. Only pods with the corresponding **Tolerations** can be scheduled on these specialized nodes. This ensures that a repository for a specific landmark

is physically hosted on the server nearest to that location, minimizing latency.

Resilience is further reinforced through Kubernetes' native **failover** mechanisms, which continuously synchronize the cluster's actual state with the desired configuration. If an edge node or backend pod crashes, the orchestrator triggers a failover sequence by rescheduling the workload onto a healthy worker. For stateful components, the use of `volumeClaimTemplates` prevents data loss during these migrations, as storage remains bound to the pod's unique identity.

The system's **load balancing** strategy operates at both the network and application levels. Internally, Kubernetes **Services** provide stable entry points for each component. While the current deployment utilizes single replicas to ensure data consistency within the local SQLite environments, the infrastructure is architecturally ready to distribute traffic across multiple instances.

3) Ingress Routing and Real time Connectivity: The networking layer ensures that client requests are efficiently routed within the cluster using a centralized **Ingress Controller**. The system utilizes path based routing rules to distinguish between standard API calls, dashboard traffic, and localized content requests. For example, when a mobile client requests an asset associated with a specific landmark, the Ingress logic automatically directs the traffic to the corresponding localized Edge Repository, minimizing the physical distance the data must travel. Furthermore, the configuration includes dedicated annotations to support persistent **WebSocket** connections, which are essential for the real time event bus. This enables the bidirectional communication required to push live geofence signals from the orchestrator to the dashboard without the latency or overhead associated with traditional polling. Through these modern orchestration techniques, the platform successfully delivers a high availability infrastructure that ensures multimedia content is always served from the most efficient network location.

IV. Results

The evaluation of the platform focuses on assessing the impact of the distributed architecture and the privacy preserving mechanisms on overall system behavior. The main outcomes of the implementation are summarized as follows:

- **Offline Caching and Prefetching:** The mobile client effectively exploited the buffer zone logic to prefetch content metadata and assets. This approach improved the perceived Quality of Service (QoS), allowing users to access content immediately upon entering a geofence, including in conditions of limited or unavailable connectivity.
- **Edge-based Content Delivery:** The deployment of localized repositories implemented with **Fastify** reduced the latency associated with high bandwidth media delivery. By offloading large assets to edge nodes, the system limited the impact of the redirection step performed by the central proxy, which remained negligible compared to the improvement in transfer performance.
- **Interpretability of Spatial Analytics:** The integration of the **K-Means clustering** component enabled the categorization of geofences into groups such as `HOTSPOT` and `COLD`. These results, visualized through dashboard heatmaps, provide an interpretable representation of user engagement and conversion rates across geographic areas.
- **Resilience and Failover:** The use of **Kubernetes** contributes to system reliability by providing built in mechanisms for workload management and recovery. The adoption of **StatefulSets** and **PersistentVolumes** ensures that stateful components can be automatically restored in the event of pod disruptions, enabling the correct reattachment of SQLite metadata and stored assets without manual intervention.
- **Privacy Preserving Geofencing Assessment:** The relationship between user privacy and system utility was evaluated using the `PrivacyAnalysisModule`. This module simulates real-world usage by generating a dataset of 1,000 synthetic location samples distributed around a single representative urban geofence. The analysis considers the effect of the grid based cloaking algorithm (grid size $\approx$ 110m at the equator) on the accuracy of Entry/Exit event detection.
  - **Privacy Gain (Perturbation Magnitude):** As shown in Figure 6, the cloaking mechanism introduces a spatial error with an average range around 70m. This distribution indicates that the mechanism reduces the precision of user localization, introducing a measurable privacy buffer.

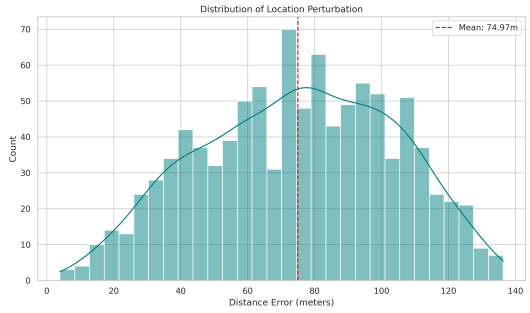


Fig. 6: Distribution of spatial error (perturbation magnitude) introduced by the cloaking mechanism.

  - **Spatial Discretization:** Figure 7 illustrates the simulated trajectory. Real user positions (blue dots) are mapped to discrete grid points (red crosses) before transmission to the backend. This discretization masks the exact trajectory while preserving approximate spatial context.
  - **Service Reliability (QoS):** Despite the introduced spatial perturbation, Figure 8 shows that the system maintains a high Quality of Service (QoS), with most location evaluations producing the correct logical

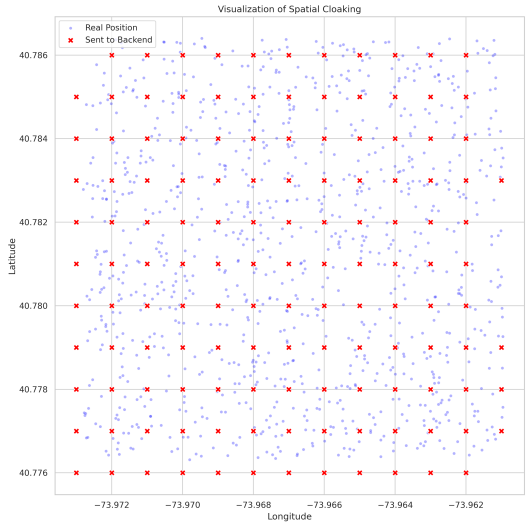


Fig. 7: Trajectory visualization comparing real user positions (blue) and cloaked grid points (red).

state (Inside vs. Outside). This suggests that the privacy mechanism does not significantly affect the core functionality of the application.

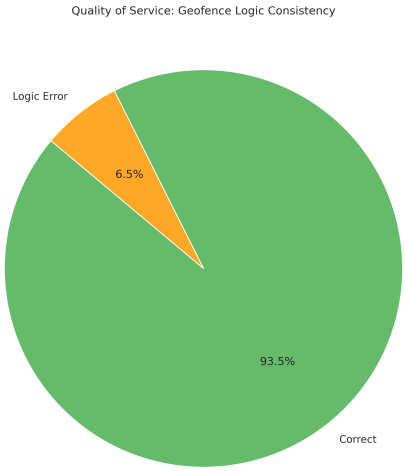


Fig. 8: Quality of Service (QoS) analysis showing the ratio of correct vs. incorrect logic states under privacy constraints.

– **Privacy vs. QoS Trade off:** The relationship between spatial error and service reliability is presented in Figure 9. The results indicate a non linear correlation: large spatial deviations (e.g., > 100m) do not necessarily produce incorrect classifications. As defined by the algorithm, errors are primarily localized near the geofence boundaries, where grid snapping may alter the containment result. This introduces a limited zone of uncertainty at the perimeter, while maintaining correct classification for positions well inside or outside the geofence.

Fig. 9: Trade off analysis correlating spatial error magnitude with logic reliability.

The obtained results demonstrate that the proposed platform successfully integrates edge based content delivery, prefetching strategies, and privacy preserving mechanisms into a cohesive architecture, ensuring efficient content distribution, continuous service availability, and the reliable provision of context-aware functionalities.