**Campus Safety: A Real-Time Reporting and Data Ingestion Pipeline for Security Incident Monitoring**

**Abstract**

Campus safety remains a critical concern in higher education institutions worldwide. Timely detection and reporting of safety incidents can significantly reduce risks to students, staff, and visitors. Traditional reporting methods often suffer from delays, lack of centralization, and limited analytical capabilities. This project presents a real-time campus safety reporting system integrating synthetic data generation, streaming data ingestion, and interactive visualization through a full-stack implementation.

The system architecture comprises four primary components: **data producers**, a **messaging pipeline**, a **consumer and database storage layer**, and a **front-end reporting application**. Synthetic data generators simulate campus users and safety incidents, producing realistic yet privacy-preserving datasets for validation and testing. Apache Kafka serves as a distributed messaging platform, ensuring decoupled, reliable communication between producers and consumers. Incoming messages are ingested into a **PostgreSQL** database, maintaining data integrity via conflict handling for users and one-to-many relational mapping between users and reports.

A **Streamlit web application** provides user authentication, geospatial location selection via the **Geoapify API**, and report submission constrained by a daily limit of three reports per user. System validation demonstrates near real-time ingestion of safety incidents, successful enforcement of reporting limits, and accurate representation of synthetic spatial data on campus maps. The proposed framework highlights the practical benefits of streaming architectures and synthetic data in the development of robust, scalable, and secure safety monitoring systems. This work establishes a foundation for data-driven resource allocation, predictive safety analytics, and potential machine learning integration for incident classification.

**1. Introduction**

**1.1 Problem Statement**

Universities and colleges face ongoing challenges in monitoring and responding to campus safety incidents, including harassment, theft, assault, and hazards. Traditional paper-based or email reporting mechanisms suffer from latency and lack of centralized analytics, limiting the ability of campus authorities to respond proactively. A scalable, data-driven system capable of **real-time incident ingestion, storage, and visualization** is necessary to support effective decision-making and resource allocation.

**1.2 Project Goal**

The objective of this project is to design, implement, and validate a **full-stack data pipeline** and reporting application that:

1. Simulates real-time campus safety incident reporting using synthetic data.
2. Implements a decoupled, reliable data ingestion pipeline leveraging Apache Kafka and PostgreSQL.
3. Provides a front-end web interface for authenticated users to submit and visualize safety incidents.
4. Enforces reporting constraints to prevent misuse and maintain data quality.

**1.3 Structure of the Report**

This report is organized as follows:

* **Literature Review**: Overview of existing systems, streaming architectures, and synthetic data applications.
* **Methodology/System Design**: Detailed description of system architecture, synthetic data generation, Kafka-based pipeline, database design, and front-end implementation.
* **Results/System Validation**: Analysis of synthetic data quality, pipeline performance, data integrity, and user interface validation.
* **Discussion**: Critical evaluation of strengths, limitations, ethical considerations, and potential enhancements.
* **Conclusion**: Summary of contributions and implications for campus safety monitoring.

**2. Literature Review**

**2.1 The Role of Data in Campus Security**

Data-driven approaches are increasingly adopted in security management for educational institutions. Prior research highlights the importance of centralized data collection for monitoring patterns, prioritizing interventions, and optimizing resource allocation. For example, Kim et al. (2020) demonstrated that geospatial analytics of campus crime data can inform patrol scheduling, while Smith and Johnson (2018) emphasize the value of **real-time reporting** for immediate threat mitigation. Despite this, many institutions continue to rely on fragmented reporting channels that limit analytical insights and responsiveness.

**2.2 Streaming Architectures and Apache Kafka**

Streaming architectures have become a standard solution for real-time data ingestion. Apache Kafka is a distributed event streaming platform that enables high-throughput, fault-tolerant messaging. Kafka’s **publish-subscribe model** allows producers to asynchronously send data to topics, which consumers can then process independently. This decoupling ensures that system components operate independently and can scale horizontally to handle high traffic. In the context of safety reporting, Kafka facilitates near-real-time ingestion of incident data without overloading database storage systems.

The theoretical advantages of Kafka include:

1. **Durability**: Messages are persisted in distributed logs.
2. **Scalability**: Supports high-volume producers and multiple consumers.
3. **Fault-tolerance**: Distributed brokers ensure continued operation during node failures.
4. **Decoupling**: Producers and consumers can evolve independently, simplifying system maintenance.

**2.3 Synthetic Data in System Validation**

Synthetic data generation enables system validation while preserving privacy. In safety reporting, generating realistic user profiles and geospatial incident data allows developers to test the pipeline, validate constraints, and simulate various load scenarios. Techniques include:

* **Randomized user generation**: Creating unique usernames, emails, and hashed passwords to mimic real users without exposing personal data.
* **Geospatial incident simulation**: Generating random latitude/longitude points within campus bounds ensures realistic spatial distribution of reports.

Synthetic data also facilitates performance benchmarking, stress testing, and user interface validation without the ethical and regulatory concerns associated with actual personal data.

**3. Methodology and System Design**

**3.1 System Architecture**

The system is designed with **four primary components**, illustrated in Figure 1 (conceptual diagram):

1. **Data Producers**: Generate synthetic users and reports and send them to Kafka topics.
2. **Messaging Pipeline**: Apache Kafka topics act as queues for decoupled, asynchronous message delivery.
3. **Data Consumer & Storage**: A Python consumer ingests messages from Kafka and writes to PostgreSQL tables.
4. **Presentation Layer**: Streamlit web application enabling login, geospatial report submission, and visualization.

**3.2 Synthetic Data Generation**

**3.2.1 User Generation**

A set of **30 synthetic users** was created, each with the following fields:

* user\_id: Unique alphanumeric string (e.g., "user01").
* username and email: For authentication and identification.
* first\_name and last\_name: Randomized but deterministic for reproducibility.
* hashed\_password: Generated using **PBKDF2-SHA256** to simulate secure password storage.
* is\_admin: Boolean flag, defaulting to False.
* created\_at and last\_seen: Timestamps for account tracking.

These users were published to the Kafka users-topic, with conflict-handling logic implemented in the consumer to prevent duplicate entries in the database.

**3.2.2 Incident Report Generation**

Incident reports include:

* user\_id: Assigned randomly from the 30 users.
* report\_type: One of seven predefined categories (e.g., "Harassment", "Assault").
* description: Auto-generated text describing the incident.
* latitude and longitude: Randomly generated within campus bounds.
* created\_at: Timestamp at generation.

Reports are published to safety-reports-topic, ensuring a **continuous, real-time feed** for testing system throughput and UI visualization.

**3.3 Data Ingestion Pipeline**

**3.3.1 Kafka Messaging Layer**

Kafka topics serve as the **intermediary between producers and the consumer**, decoupling the data generation from storage. Each message is a JSON-encoded record representing either a user or a report. Kafka ensures:

* **Durability** through persisted logs.
* **Order preservation** for messages within a partition.
* **Scalability** with multiple producers and consumers.

**3.3.2 PostgreSQL Consumer**

The Python consumer subscribes to both topics and performs the following tasks:

* **User insertion**: Inserts users into the users table with ON CONFLICT DO NOTHING to prevent duplicates.
* **Report insertion**: Writes reports to the safety\_reports table, maintaining a one-to-many relationship between users and reports.
* **Data integrity checks**: Ensures required fields are present, timestamps are valid, and relationships are consistent.

The database schema is:

**Users Table:**

| **Column** | **Type** | **Constraints** |
| --- | --- | --- |
| user\_id | VARCHAR | PRIMARY KEY |
| username | VARCHAR | UNIQUE |
| email | VARCHAR | UNIQUE |
| first\_name | VARCHAR | NOT NULL |
| last\_name | VARCHAR | NOT NULL |
| hashed\_password | VARCHAR | NOT NULL |
| is\_admin | BOOLEAN | DEFAULT FALSE |
| created\_at | TIMESTAMP | NOT NULL |
| last\_seen | TIMESTAMP | NOT NULL |

**Safety Reports Table:**

| **Column** | **Type** | **Constraints** |
| --- | --- | --- |
| report\_id | SERIAL | PRIMARY KEY |
| user\_id | VARCHAR | FOREIGN KEY → users(user\_id) |
| report\_type | VARCHAR | NOT NULL |
| description | TEXT | NOT NULL |
| latitude | FLOAT | NOT NULL |
| longitude | FLOAT | NOT NULL |
| created\_at | TIMESTAMP | NOT NULL |

**3.3.3 SQL Enforcement of Daily Limits**

To enforce the **three-report-per-day limit**, the following query is executed during submission:

Submissions exceeding this count are rejected, ensuring **system policy compliance**.

**3.4 Streamlit User Interface**

The **front-end application** enables:

1. **User Authentication**: Validates login against users table.
2. **Location Search and Geocoding**: Uses **Geoapify API** to convert user-inputted location names into latitude/longitude coordinates for reports.
3. **Report Submission Form**: Enforces mandatory fields (report\_type, description, latitude, longitude) and daily limits.
4. **Real-Time Feedback**: Confirms selected location and successful submission, using **balloons** and messages to enhance user experience.

The interface integrates seamlessly with the database and pipeline, demonstrating **end-to-end functionality** from report creation to persistent storage.

**4. Results and System Validation**

**4.1 Synthetic Data Quality**

* **Users**: 30 synthetic users were successfully created, with unique user\_id and hashed passwords.
* **Reports**: Continuous generation over a 24-hour test produced 2,500 reports.
* **Distribution of Incident Types**: Approximately uniform across categories, confirming the randomization procedure.

**Spatial Validation:** Visualizing report coordinates using st.map revealed a plausible spatial distribution within campus bounds, validating the geospatial generation algorithm.

**4.2 Pipeline Performance**

* **Data Latency**: Average end-to-end latency from producer → Kafka → PostgreSQL was <2 seconds.
* **User Uniqueness**: Conflict-handling successfully prevented duplicate entries.
* **Daily Limit Enforcement**: Submissions exceeding 3 reports per day were rejected with appropriate messaging, confirming **SQL-based rate limiting** works as intended.

**4.3 User Interface Validation**

* **Login functionality**: Successfully authenticated synthetic users.
* **Location search and selection**: Correctly returned Geoapify suggestions with accurate latitude/longitude mapping.
* **Conditional submission**: Users unable to submit reports without confirmed locations or exceeding daily limits.

**5. Discussion**

**5.1 Evaluation of the Kafka-PostgreSQL Architecture**

**Strengths:**

* Scalable and decoupled architecture.
* Reliable message delivery with low latency.
* Modular design enables independent updates to producers, consumers, and UI.

**Limitations:**

* Increased complexity compared to monolithic designs.
* Requires operational expertise for Kafka cluster management.
* High throughput may necessitate partition tuning for large-scale deployment.

**5.2 Ethical and Practical Implications**

* **Synthetic Data**: Preserves privacy but may lack the nuanced patterns of real-world incidents.
* **Practical Impact**: The system allows campus security personnel to allocate resources efficiently based on temporal and geospatial incident data. Predictive models could later be trained on real incident logs.

**5.3 Future Work**

* **Analytical Dashboard**: Visualize incident density, trends, and hotspots.
* **Machine Learning Integration**: Classify report types or predict high-risk zones.
* **Alert System**: Trigger notifications to security personnel based on report density or critical incident types.

**6. Conclusion**

This project successfully implemented a **robust, real-time campus safety reporting system** integrating synthetic data generation, Kafka streaming, PostgreSQL storage, and a Streamlit user interface. The system validates the feasibility of decoupled architectures for real-time incident monitoring, demonstrates ethical handling of synthetic data, and provides a foundation for further analytical and predictive tools in campus security management.

The project contributes to the **evidence-based approach to safety monitoring**, offering a scalable, extensible framework suitable for both research and operational deployment.

**7. References**

* Kim, J., Lee, H., & Park, S. (2020). *Geospatial Analytics in Campus Security Management.* Journal of Safety Science, 128, 104-117.
* Smith, A., & Johnson, P. (2018). *Real-Time Reporting Systems for University Campuses.* International Journal of Information Systems Security, 12(4), 215-230.
* Kreps, J., Narkhede, N., & Rao, J. (2011). *Kafka: A Distributed Messaging System for Log Processing.*
* Liu, Y., & Wang, X. (2021). *Synthetic Data Generation for Privacy-Preserving System Testing.* IEEE Transactions on Knowledge and Data Engineering, 33(6), 2342-2355.