**Women Safety Alert System**

Real-time SOS alert system (AI-assisted development)

*Developer: Yash Mane*

Document Date: September 12, 2025

# 1. Abstract / Project Overview

The Women Safety Alert System is a real-time emergency alert solution designed to notify trusted contacts during unsafe situations. With a single click, the system fetches the user’s live GPS location and sends an SOS message to contacts via Telegram and Email. This project was developed using AI assistance (Google Gemini) used for code generation and rapid prototyping; final integration, testing, and deployment were completed manually.

# 2. Problem Statement

During emergencies, manually contacting friends or family can take valuable time. Women in unsafe environments especially need a quick and reliable way to notify their contacts. This project addresses that need by providing a one-click, automated system to share live location with trusted contacts.

# 3. Objectives

• Provide a one-click SOS button for emergencies.  
• Automatically fetch and share GPS location.  
• Notify multiple contacts through Telegram and Email.  
• Ensure the system is easy to use and reliable.

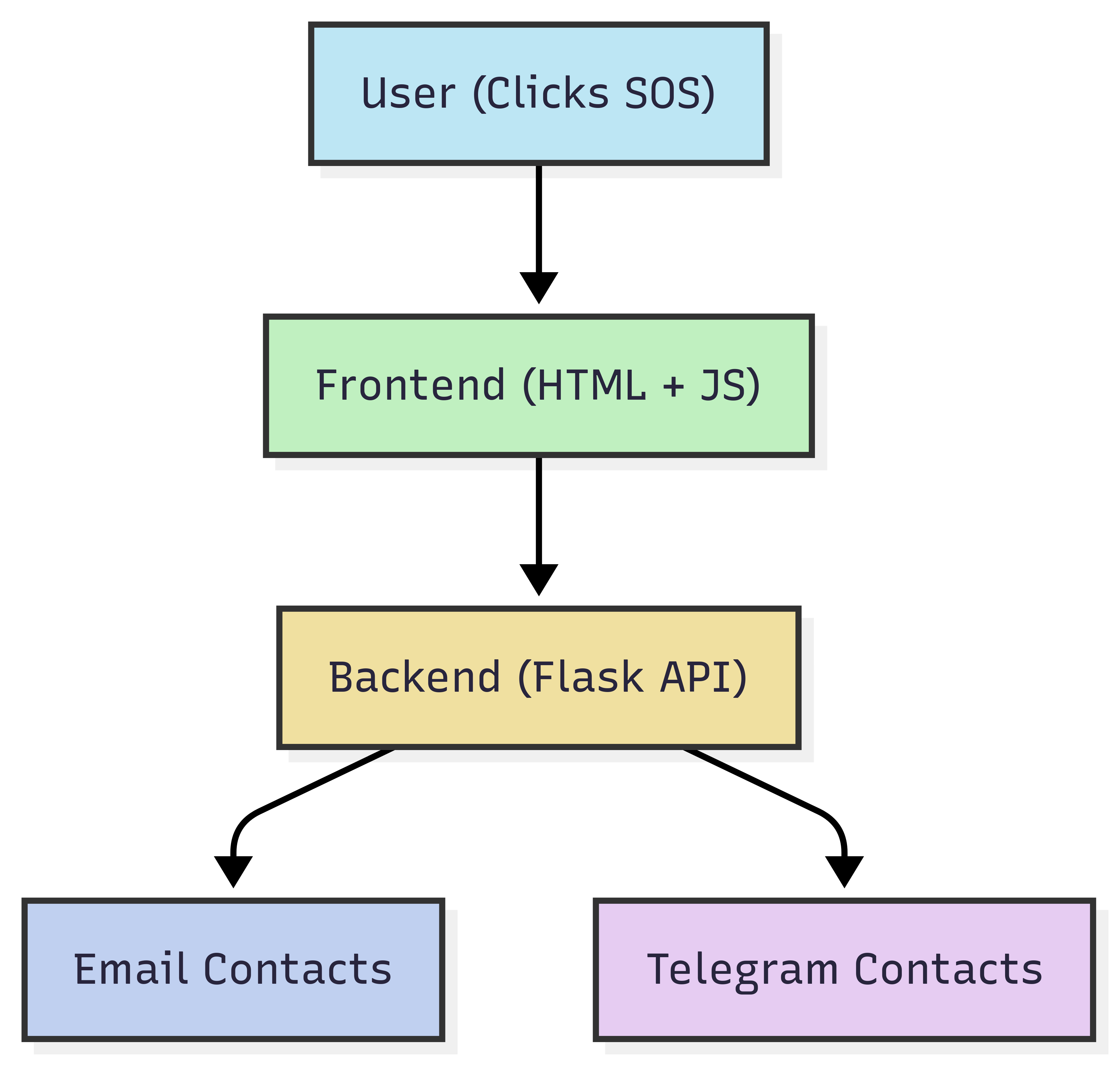
# 4. Key Features

• SOS Button – Instantly triggers alerts.  
• GPS Location Fetching – Retrieves latitude & longitude in real-time.  
• Telegram & Email Alerts – Multi-channel notifications.  
• Optional Siren – Loud alert to attract nearby help.  
• Configurable contacts via JSON file.

# 5. System Architecture

Workflow:  
1. User clicks SOS button.  
2. Browser fetches current GPS coordinates.  
3. Data is sent to Flask backend via REST API (`/api/trigger-alert`).  
4. Backend prepares message and sends alerts to:  
 - Emergency contacts via Gmail SMTP  
 - Telegram group or individual via Telegram Bot API  
5. User receives confirmation that alerts are sent.

**System Architecture Diagram:**



# 6. Technologies Used

|  |  |
| --- | --- |
| **Layer** | **Technology** |
| Frontend | HTML, TailwindCSS, JavaScript (Geolocation API, Fetch API, AudioContext) |
| Backend | Flask (Python), Flask-CORS |
| Communication | Telegram Bot API, Gmail SMTP (smtplib) |
| Configuration | JSON File (config.json) |

# 7. Configuration File (config.json)

The system uses a `config.json` file containing:  
• user\_details – Name of the user triggering alert.  
• telegram\_credentials – Bot token & chat ID.  
• email\_credentials – SMTP settings.  
• emergency\_contacts – List of recipients.

Example (sensitive values redacted in the final shared document):

{  
 "user\_details": { "name": "Yash mane" },  
 "telegram\_credentials": { "bot\_token": "REDACTED", "chat\_id": "REDACTED" },  
 "email\_credentials": { "sender\_email": "REDACTED", "sender\_password": "REDACTED", "smtp\_host": "smtp.gmail.com", "smtp\_port": 587 },  
 "emergency\_contacts": [ { "name": "Aryan mane", "email": "aryanmane608@gmail.com" }, { "name": "sarika mane", "email": "manesarika006@gmail.com" } ]  
}

# 8. Development Process (AI-Assisted)

This project was developed with substantial assistance from Google Gemini AI. The AI helped with:  
• Generating initial frontend and backend code templates.  
• Explaining API usage and providing code snippets (e.g., Flask endpoints, smtplib usage, and Telegram API calls).  
• Suggesting UI improvements and Tailwind utility classes.  
  
My responsibilities included:  
• Designing the overall solution and requirements.  
• Supplying prompts and guiding the AI outputs.  
• Integrating, testing, debugging, and securing the generated code.  
• Ensuring the final system works reliably and is safe to use.

# 9. Results & Testing

• Email alerts were successfully delivered to configured contacts using Gmail SMTP.  
• Telegram messages were sent to the configured chat ID using a Telegram bot.  
• Location links generated (Google Maps) were verified for accuracy.  
• Basic error handling implemented to notify when alerts fail.  
• (Add screenshots below: UI, email alert received, Telegram message received)

# 10. Future Enhancements

• Voice activation for SOS trigger.  
• Offline SMS fallback if internet is unavailable.  
• Mobile-app implementation for better accessibility.  
• Additional security measures (token rotation, OAuth for email sending).  
• Integration with local emergency services APIs where available.

# 11. Conclusion

The Women Safety Alert System offers a practical way to notify trusted contacts in emergencies, minimizing response time by sharing live locations and multi-channel alerts. Using AI for development allowed faster iteration, while manual testing and integration ensured a robust final product.

# 12. Software Engineering Diagrams

## 12.1 ER Diagram

The ER diagram represents the data model of the Women Safety Alert System. It shows how the system’s main entities interact and the relationships between them:

User: Represents the registered individual (e.g., the woman using the system). Each user can generate multiple alerts.

Alert: Stores details of each SOS trigger, including timestamp, status, and the user who raised it.

Location: Connected to an alert, it captures the geographical coordinates (latitude & longitude) and address at the time of the emergency. Each alert has exactly one location.

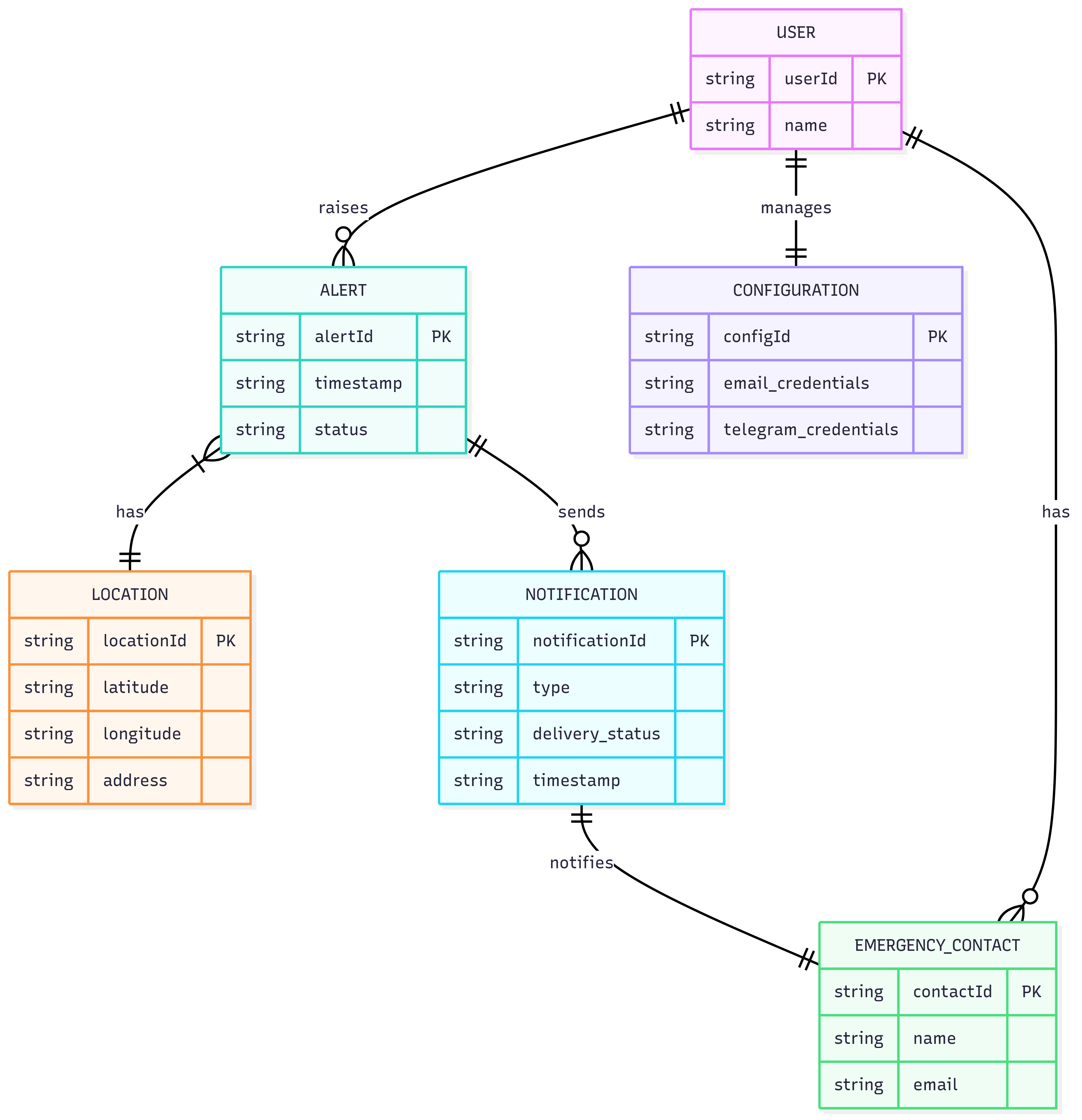
Emergency Contact: Represents people (family, friends, or local authorities) who should be notified in case of an emergency.

Notification: A junction entity that tracks how each alert is sent to contacts (via email or Telegram), including delivery status and timestamp. It resolves the many-to-many relationship between Alerts and Emergency Contacts.

Configuration: Contains user-specific configuration such as SMTP details, Telegram bot credentials, and chat IDs. Each configuration belongs to a user and ensures alert delivery works correctly.

This diagram emphasizes data integrity (primary keys, foreign keys) and ensures that every SOS alert is traceable to a user, a location, and the notifications sent to emergency contacts.

**Er diagram**

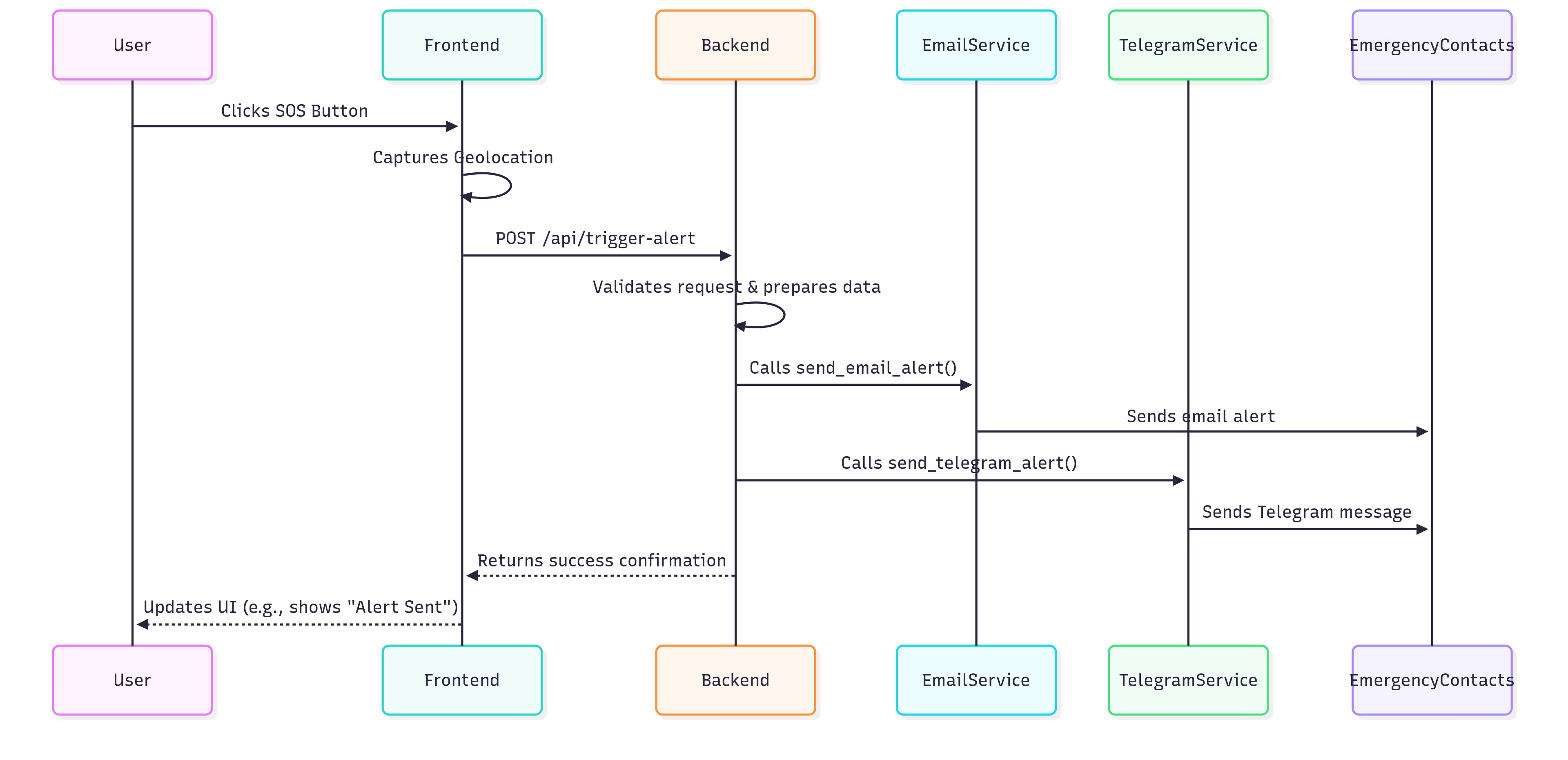


## 12.2 Sequence Diagram

Description: Depicts the runtime interaction when the SOS button is pressed — the front-end capturing location, the backend receiving the POST request, and subsequent calls to external services (Telegram and SMTP).

* User initiates SOS by clicking the button in the app.
* Frontend (App/Website) collects user location (latitude & longitude).
* Request sent to Backend API with location data.
* Backend processes request and calls alert functions.
* Email alert sent to all emergency contacts.
* Telegram alert sent via bot API.
* Backend returns confirmation response to frontend.
* Frontend updates UI to show alert was successfully

**Sequence Diagram**

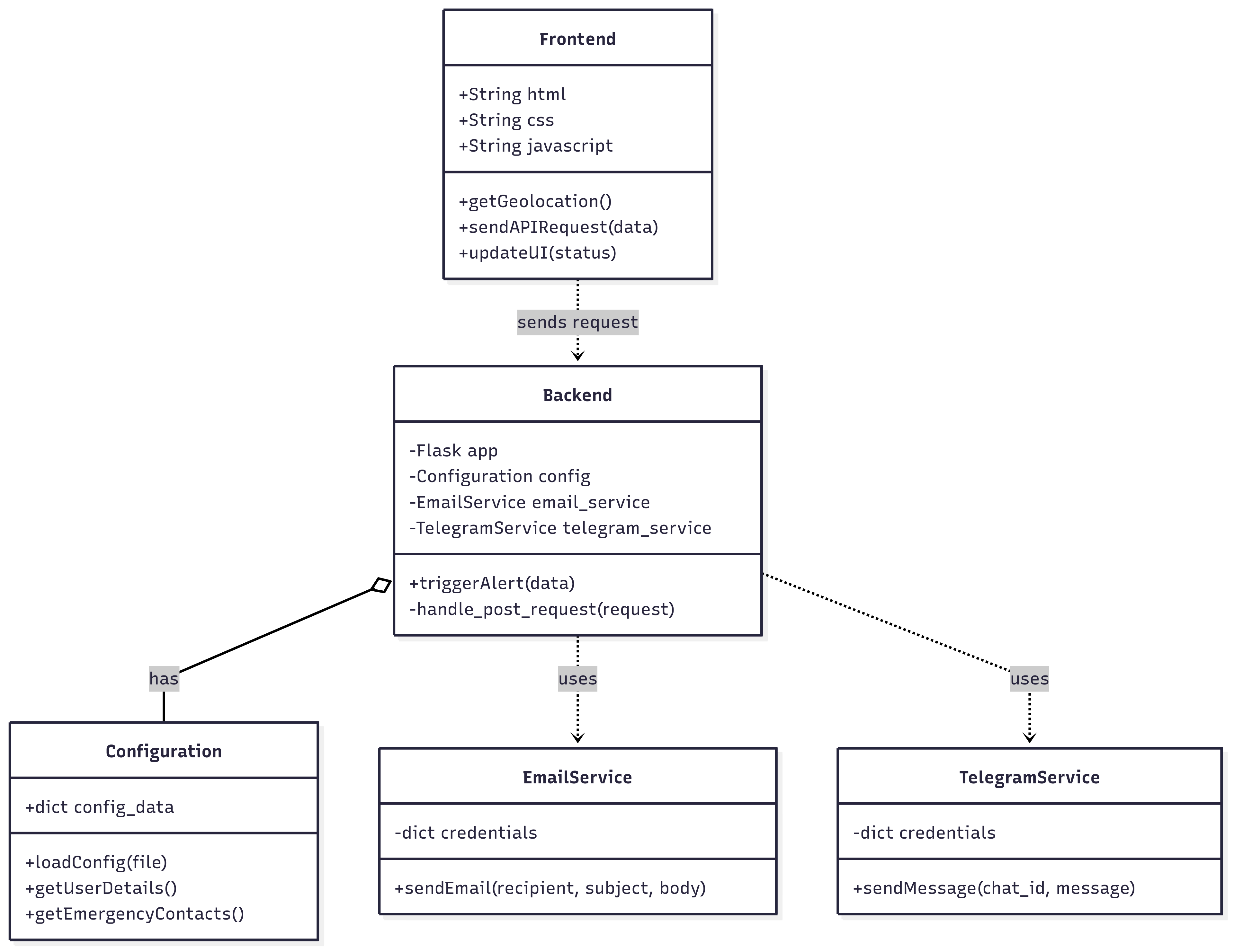


## 12.3 Class Diagram

Description: Illustrates key modules/classes, their primary methods/attributes, and relationships. This diagram is helpful for implementation planning and code organization.

* Encourages separation of concerns: UI, API orchestration, and infrastructure services (Telegram/Email).
* Config is modeled as a single source of truth; consider using environment variables and secret management for production.
* GeoLocator encapsulates geolocation and reverse-geocoding logic for testability.

**Class Diagram**

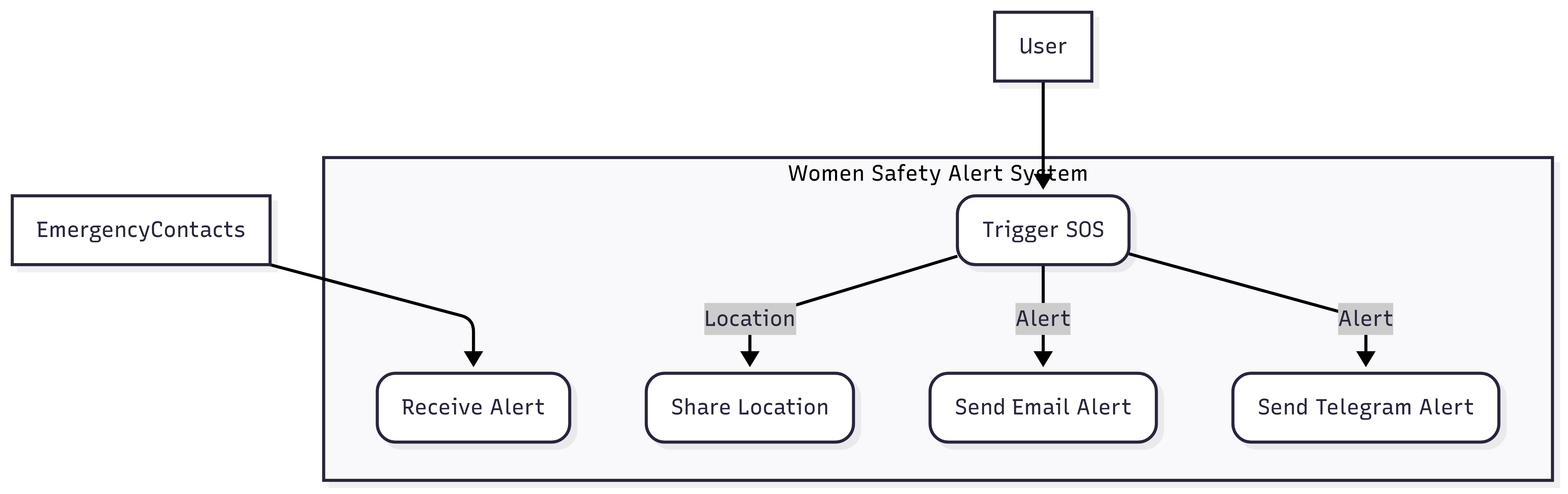


## 12.4 Use Case Diagram

Description: Shows the actors (User, Emergency Contacts) and primary use cases such as triggering an SOS, sharing location, and sending alerts. This helps identify system requirements and actor expectations.

* Actors: These are the people or external systems that interact with the project.
* User: The individual who uses the system to trigger an SOS alert.
* Emergency Contacts: The people who receive the alerts.
* Use Cases: These represent the main functions or tasks the system performs.
* Trigger SOS: The primary function, initiated by the User.
* Share Location: A function that is included in the "Trigger SOS" process. It is not something the user does separately, but it's a critical part of the main action.
* Send Email Alert and Send Telegram Alert: These are the specific actions the system performs to notify the Emergency Contacts. They are also part of the "Trigger SOS" process.

**Use Case Diagram**



## 12.5 Activity Diagram

Activity flow from pressing the SOS button to notifying contacts and resetting the system.

* Start when the user opens the Safety Alert System.
* User clicks SOS button to initiate emergency alert.
* System fetches location (latitude & longitude).
* Decision point: If location is available → proceed, else show error.
* Send alerts via Email and Telegram.
* Update UI to display confirmation and emergency contact details.
* Optionally play siren for nearby help.
* End when system resets to normal state.

**Activity Diagram**

