Earthquake Early Warning Mobile App - Software Requirements Specification (SRS)

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) defines the necessary components for the development of the Earthquake Early Warning Mobile App. The document serves as a detailed guideline for the developers, testers, and stakeholders involved in the app's development, deployment, and operation. It describes the structure, functionalities, workflows, and system requirements of the app to ensure that all parties involved have a clear understanding of the project's scope, objectives, and deliverables.
The purpose of the Earthquake Early Warning Mobile App is to provide real-time notifications and alerts to users when an earthquake is imminent. It aims to improve preparedness and safety by delivering early warnings to individuals who are at risk of being affected by seismic events. The app will also allow users to customize alert preferences, monitor seismic activity in their area, and access additional earthquake-related information.

1.2 Overall Design

1.2.1 Theoretical Model and Structure

The Earthquake Early Warning App follows a **three-tier architecture** model, which divides the system into distinct layers to separate concerns and ensure scalability, maintainability, and testability.

Controller Layer: This layer manages user interactions with the mobile app. It handles user input, such as location preferences, notification settings, and feedback. The controller sends user requests to the service layer and processes the response to display the appropriate information to the user. This layer is responsible for the app's front-end user interface and user experience.

Service Layer: This layer encapsulates the core business logic of the app. It is responsible for processing incoming seismic data, determining the magnitude and impact of detected earthquakes, generating appropriate alerts based on the user's location, and managing the lifecycle of notifications. This layer acts as a bridge between the front-end controller and the data access layer, coordinating the communication between them.

Data Access Layer (DAO): The DAO layer interacts with external seismic data sources (such as seismic monitoring services or APIs) and manages the storage and

retrieval of data in the app's database. It ensures that the app has access to the necessary information, such as seismic event data, user profiles, alert histories, and notification preferences. This layer is responsible for executing data queries and abstracting the details of the underlying data storage systems.

The overall workflow involves receiving real-time seismic data, processing it, and sending alerts to users based on their location and the severity of the detected earthquake.

Workflow: When an earthquake event occurs, real-time seismic data is fetched by the data access layer from external sources. This data is passed to the service layer, where it is processed to determine whether it meets the threshold for an alert. If so, the service layer sends an alert to the user, who receives a notification on the app's front-end interface (controller layer). The user's preferences and settings will dictate how and when these alerts are displayed.

1.2.2 Technologies

The Earthquake Early Warning Mobile App utilizes the following technologies:

Frontend: Developed using Flutter to provide a consistent and responsive user interface across both iOS and Android platforms. Flutter allows the app to be cross-platform, reducing development time and ensuring a smooth user experience on various devices and screen sizes.

Backend: The backend is powered by Node.js and Express, which enable fast and scalable processing of real-time seismic data. The app's backend provides RESTful APIs that are used to manage user interactions, handle data processing, and trigger notifications. Express handles routing and middleware, making the backend scalable and easy to maintain.

Database: PostgreSQL is used as the database management system for storing structured data related to users, alerts, seismic events, and notification histories. PostgreSQL is chosen for its robustness, ability to handle complex queries, and high-performance capabilities.

Communication: The app communicates with external seismic monitoring systems (such as USGS, EMSC, or local monitoring stations) using REST APIs. The app processes incoming seismic data in JSON format, which is a lightweight data interchange format that is easy to parse and work with in real-time**.**

1.2.3 Expected System Performance

The app's performance expectations are as follows:

Real-Time Notifications: The app should be capable of sending real-time earthquake notifications to up to **500,000 concurrent users** without significant delays or system failure. This will be accomplished through efficient notification delivery mechanisms, such as push notifications or local alerts.

Seismic Data Processing: Incoming seismic data must be processed within **1 second** of receipt to ensure that alerts are timely and provide adequate time for users to respond. The app needs to prioritize low-latency data processing to meet this requirement.

Scalability: The app should be scalable and capable of handling increased demand during high-activity periods, such as after a large earthquake or in areas with frequent seismic activity. This scalability can be achieved by using cloud infrastructure, load balancing, and auto-scaling mechanisms to handle varying traffic loads effectively.

High Availability: The system should have high availability, ensuring that users receive accurate and timely earthquake alerts even during periods of heavy traffic or network congestion. This includes implementing redundant systems and data backup procedures to maintain uptime.

1.3 References

This document is based on the following references and standards:

- **Earthquake Monitoring API Documentation**: Guides for integrating with seismic data providers (e.g., USGS, EMSC).
- **Regional Seismic Data Policies and Standards:** Local regulations and best practices for earthquake data collection, analysis, and distribution.
- **Mobile Development Frameworks and Guidelines for iOS and Android:** Flutter framework documentation and mobile UI/UX design guidelines for ensuring compatibility across devices.
- **Notification System Guidelines:** Industry standards for mobile notification delivery and management, such as push notifications, location-based alerts, and frequency management.
- **Security Best Practices:** Recommendations for securing user data and ensuring privacy compliance (e.g., GDPR, CCPA).

These references will guide the development process, ensuring that the app adheres to industry standards, meets the necessary performance criteria, and complies with relevant regulations for seismic data and user privacy.

2. System Overview

2.1 Structural Design

The Earthquake Early Warning Mobile App is built on a layered architecture that ensures modularity, scalability, and maintainability. The system is divided into three main components:

1. **Controller Layer**

Primary Role: Manages the mobile user interface and user interactions.
Responsibilities:
Capturing user inputs such as location preferences, alert settings, and feedback.
Communicating with the service layer to request data and trigger alerts.
Displaying real-time notifications and seismic event details in an intuitive format.

2. **Service Layer**

Primary Role: Implements the core business logic of the application.
Responsibilities:
Receiving real-time seismic data from the data access layer.
Processing and analyzing seismic data to determine key metrics (e.g., magnitude, epicenter location, and user proximity).
Evaluating whether an event meets the predefined criteria for issuing alerts.
Generating alert notifications and managing their lifecycle, including scheduling and dispatch.

3. **Data Access Layer (DAO Layer)**

Primary Role: Manages data retrieval and storage.
Responsibilities:
Interfacing with external seismic data providers (such as USGS or local monitoring stations) via RESTful APIs.
Fetching real-time earthquake data in JSON format.
Storing seismic events, user profiles, and alert histories in a PostgreSQL database.
Ensuring that the service layer has access to reliable and up-to-date data.
Workflow Example:
When a seismic event is detected, the data access layer fetches the latest real-time data from external providers. This data is then passed to the service layer, where it is analyzed against predefined criteria (e.g., magnitude threshold and proximity). If the event qualifies for an alert, the service layer generates an alert message, which is then relayed to the controller layer. The controller layer, in turn, displays the notification on the user's mobile device, enabling timely action.

2.2. Design Principles

To ensure a robust, efficient, and adaptable system, the following design principles are observed:

Compatibility

Cross-Platform Support: The app must operate seamlessly on both iOS and Android platforms, adapting to various screen sizes and device capabilities.
API Integration: It should maintain compatibility with existing seismic data provider APIs (e.g., USGS, local monitoring stations) by utilizing standard protocols (REST) and data formats (JSON).

Interoperability: The system is designed with open and standardized interfaces, allowing easy integration with external systems, such as disaster management platforms or additional sensor networks.

Robustness

Input Validation: All user inputs, including location data and notification settings, must be rigorously validated to ensure correctness and security.

Exception Handling: The system includes robust exception handling to manage network disruptions, invalid data, or external service failures, ensuring continuous operation or graceful degradation.

Logging and Monitoring: Comprehensive logging of critical events, such as alert generation and data retrieval, is implemented. This facilitates real-time monitoring, performance analysis, and rapid troubleshooting when issues arise.

Flexibility

Modular Architecture: The system is designed with loosely coupled modules, enabling independent updates or enhancements without affecting the overall functionality.

Expandable Interfaces: The architecture includes expandable APIs, allowing future integration of new seismic data sources or additional features (e.g., advanced push notifications, location-based customization).

Dynamic Alert Strategies: The app supports dynamic configuration of alert thresholds and strategies, which can be tailored based on regional seismic activity and individual user preferences.

Distributed Deployment: To handle high traffic and ensure high availability, especially during seismic events, the system is designed for distributed deployment with load balancing and auto-scaling capabilities.

By adhering to these design principles, the Earthquake Early Warning Mobile App is positioned to deliver timely, accurate, and user-friendly alerts, significantly enhancing safety and preparedness during seismic events.

3. Database Design(改写)

3.1 User Profile Table

Field Name	Data Type	Key	Nullable	Description
User_ID	INT	Primary Key	No	Unique identifier for each user.
Name	VARCHAR(100)		No	User’s full name.
Email	VARCHAR(100)		No	User’s email address.
Location	VARCHAR(100)		Yes	User's registered location.
Notification_Pref	BOOLEAN		Yes	Whether user wants to receive notifications.

3.2 Event Table

Field Name	Data Type	Key	Nullable	Description

Field Name	Data Type	Key	Nullable	Description
Event_ID	INT	Primary Key	No	Unique identifier for each seismic event.
Magnitude	DECIMAL(3,1)		No	Magnitude of the earthquake.
Location	VARCHAR(100)		No	Location of the seismic event.
Event_Time	DATETIME		No	Timestamp of when the event occurred.
User_Alerted	BOOLEAN		Yes	Whether alerts have been sent to users.

3.3 Seismic Data Table

Field Name	Data Type	Key	Nullable	Description
Data_ID	INT	Primary Key	No	Unique identifier for the seismic data.
Event_ID	INT	Foreign Key	No	Associated event identifier.
Seismic_Value	DECIMAL(5,2)		No	Recorded seismic value at a specific time.
Timestamp	DATETIME		No	Timestamp of the seismic data.

4. Module Design（改写）

4.1 Earthquake Detection Module

4.1.1 Function Description

The Earthquake Detection Module is the core component responsible for continuously collecting, processing, and analyzing real-time seismic data from external sources such as USGS or other regional monitoring systems. Its primary goal is to identify potential earthquake events by calculating key parameters (e.g., magnitude, epicenter, depth) and determining whether these events meet predefined alert criteria. Upon detecting a significant event, this module triggers the generation of alerts that are then forwarded to the Notification Module for delivery to users.

4.1.2 Functional Requirements

Seismic Data Retrieval:
Continuously fetch real-time seismic data from multiple external sources using RESTful APIs.
Support asynchronous data fetching with error handling and retries in case of connectivity issues.
Data Normalization and Validation:
Normalize and validate the raw seismic data to ensure consistency (e.g., standardizing units, filtering out noise).

Magnitude Calculation and Analysis:
Process raw sensor data to calculate the earthquake's magnitude using established methods (e.g., Richter scale, moment magnitude scale).
Determine additional event characteristics such as epicenter coordinates, depth, and duration.
Alert Criteria Evaluation:
Compare the calculated parameters with predefined thresholds (e.g., magnitude > 5.0, user within 50 km of the epicenter).
Consider additional factors like event duration and intensity for enhanced decision-making.
Alert Generation:
Generate alert records when an event qualifies, including detailed event information.
Log events and processing outcomes for audit and future analysis.

4.1.3 Logical Process

Data Collection:
The system fetches seismic data from external sources via scheduled or continuous API calls.
Normalization & Validation:
Raw data is normalized (e.g., unit conversion, timestamp formatting) and validated to remove any erroneous entries.
Event Processing:
The normalized data is analyzed to compute the event's magnitude, epicenter location, and other critical metrics.
Criteria Evaluation:
The processed data is compared against the alert criteria (e.g., magnitude threshold, proximity of registered users).
Alert Generation:
If an event qualifies, a UserAlert is generated for users within the affected area.
Logging & Handoff:
The event and alert details are logged, and the alert is passed to the Notification Module for dispatch.

4.1.4 Primary Objects

SeismicEvent
Attributes:
Event_ID: Unique identifier for the seismic event.
Magnitude: Calculated magnitude of the event.
Location: Geographic coordinates or descriptive location of the epicenter.
Event_Time: Timestamp indicating when the event occurred.
Optional: Depth, Intensity, Duration.

UserAlert
Attributes:

User_ID: Identifier of the user who is to receive the alert.
Event_ID: Associated seismic event's unique identifier.
Alert_Time: Timestamp when the alert was generated.
Alert_Status: Indicator showing whether the alert is pending, sent, or acknowledged.

4.1.5 Functions

fetchSeismicData():
Retrieves real-time seismic data from external APIs, handling connectivity issues and parsing the JSON response.
normalizeData():
Processes raw seismic data to ensure consistency, including unit conversion and timestamp normalization.
processEvent():
Analyzes the normalized data to identify valid seismic events by calculating key parameters.
evaluateEventCriteria():
Compares event metrics against predefined thresholds to decide whether an alert should be generated.
generateAlert():
Creates and dispatches an alert for users within the event's impact area and logs the alert details.
logEvent():
Records the details and outcomes of the event processing for monitoring and audit purposes.

4.1.6 API Design

- **Fetch Seismic Data**
  - **URL: /api/seismic-data**
  - **Method: GET**
  - **Request Parameters:**
    - **Optional parameters such as region, time range, or data source identifier.**
  - **Response:**
    - **A JSON object containing seismic data (e.g., magnitude, epicenter location, event time, depth).**
- **Generate Alert**
  - **URL: /api/alert**
  - **Method: POST**
  - **Request Parameters:**
    - **Event_ID: Unique identifier of the seismic event.**
    - **User_ID: Identifier(s) of the user(s) targeted for the alert.**
    - ***Optional*: Alert priority or channel information.**
  - **Response:**

- **A JSON object indicating the success or failure of the alert generation, along with status details.**

4.2 Notification Module

4.2.1 Function Description

The Notification Module is responsible for the delivery of earthquake alerts to end users. It manages the dispatch of notifications through various channels (e.g., push notifications, SMS, email) based on user preferences. This module ensures that alerts are delivered in a timely manner, reflecting real-time updates and allowing users to customize their notification settings to match their specific requirements and local circumstances.

4.2.2 Functional Requirements

Send Alerts:

- **Dispatch notifications to users as soon as an alert is generated.**
- **Support multiple notification channels, ensuring redundancy in alert delivery.**

User Preference Management:

- **Enable users to configure their alert settings, including types of notifications (emergency vs. informational), frequency, and quiet hours.**

Real-Time Updates:

- **Provide ongoing updates about an earthquake event, such as changes in magnitude or additional safety instructions.**

Delivery Confirmation and Feedback:

- **Capture acknowledgment of received alerts and provide mechanisms for users to report issues or inaccuracies.**

Reliability and Retry Logic:

- **Implement retry mechanisms for failed notification attempts to ensure high delivery success rates.**

4.2.3 Logical Process

Preference Retrieval:

- Upon receiving an alert from the Earthquake Detection Module, the system retrieves the user's notification preferences from the database.

Alert Preparation:

- A notification message is prepared, incorporating essential event details (e.g., magnitude, location, timestamp) and safety recommendations.

Dispatch Notification:

- The notification is sent to the user via the configured channel (e.g., push notifications using Firebase Cloud Messaging, SMS via a messaging gateway).

Delivery Confirmation:

- The system verifies that the notification has been successfully delivered. In cases of failure, it triggers retry logic.

Logging and Feedback:

- Delivery status and user responses are logged for monitoring and further optimization of the notification process.

4.2.4 Primary Objects

Notification

- **Attributes:**
  - **User_ID: Identifier for the recipient.**
  - **Event_ID: Identifier for the associated seismic event.**
  - **Alert_Type: Specifies the nature of the alert (e.g., emergency, informational).**
  - **Alert_Time: Timestamp when the notification was sent.**
  - **Delivery_Status: Status of the notification (e.g., Sent, Delivered, Failed).**
  - ***Optional*: Message content, communication channel, acknowledgment status.**

4.2.5 Functions

sendNotification():
Sends the prepared notification to the user via the selected communication channel and handles success or failure responses.
updateUserPreferences():
Provides an interface for users to update their notification settings and preferences.
resendNotification():
Implements a retry mechanism to resend notifications in the event of initial delivery failures.
logNotification():
Records details of each notification sent, including delivery status and timestamp, for monitoring and troubleshooting.
acknowledgeNotification():
Processes user acknowledgments, confirming receipt of the notification and optionally capturing user feedback.

4.2.6 API Design

Send Notification

- **URL: /api/notify**
- **Method: POST**
- **Request Parameters:**
  - **User_ID: The identifier of the recipient.**
  - **Event_ID: The associated seismic event's unique identifier.**

    - **Notification_Type: The type of notification (e.g., emergency, informational).**
    - ***Optional*: Communication channel (e.g., push, SMS).**
- **Response:**
    - **A JSON object indicating the status of the notification dispatch (e.g., success, failure, pending) and any relevant error messages.**

5. Conclusion

The Earthquake Early Warning Mobile App is engineered to deliver real-time, actionable earthquake alerts to users, significantly enhancing their ability to respond promptly during seismic events. Through a well-structured modular design—dividing responsibilities between the Earthquake Detection and Notification Modules—the app efficiently processes real-time seismic data, evaluates event criteria, and disseminates alerts based on user-specific parameters. By leveraging robust external data sources, advanced processing algorithms, and flexible notification channels, the system is poised to improve safety and preparedness in earthquake-prone regions.
o3-mini-high