# B.R.A.M. (BORIS,RAMSON,AKA,MONDE) Disaster Management App

GROUP MEMBERS:

**EBAN BORIS MANYO  SENG22SE006**

**NYENTY OBI RAMSON SENG22SE008**

**AKAWUNG GABRIEL  SENG22TE001**

**IKOME BRYAN JOHNSON  SENG22SE007.**

**COURSE INSTRUCTOR: DR VALERY**

**Technology Stack: React Native (JavaScript), Expo,Mobil device .**

## 1. Introduction

The **B.R.A.M. (BORIS,RAMSON,AKA,MONDE)** Disaster Management App is a mobile application designed to empower residents of Buea and surrounding areas with tools for disaster preparedness, reporting, and awareness. Developed using React Native, the app aims to provide a user-friendly interface for citizens to report real-time disaster events, receive alerts, access crucial self-rescue tips, and manage their personal profiles. The current version serves as a functioning app, focusing on frontend functionality with local data storage using MySQL.

## 2. Core Features

The B.R.A.M. app encompasses the following key functionalities:

- Disaster Reporting: Users can report various types of disasters (Flood, Fire, Landslide, Earthquake) by providing descriptions and uploading media (images/videos).
- Real-time Alerts: The app displays alerts based on reported disasters, with visual indicators for new or critical events at a particular time.
- User Authentication: A robust user system allows for registration and login, controlling reporting privileges.
- Profile Management: Users can create and manage their profiles locally.
- Disaster Feed: Dedicated sections for each disaster type to view community-reported incidents.

- Self-Rescue Tips: A dedicated section providing essential guidance for various disaster scenarios.
- Local Data Storage: For this app, all user data (profiles, reported disasters) are stored locally on the device using My SQL for storing the user data and feeds

### 3. Application Screens and Functionality

The B.R.A.M. app is structured around a tab-based navigation system, providing seamless access to its core features:

### 3.0. Loading Screen


### 3.1. Welcome Screen (Initial Splash/Onboarding)

- Purpose: The very first screen a user sees when opening the app.
- Content:

➢ Prominently displays "Welcome to B.R.A.M."
➢ A brief tag line like "Get real-time alerts and report disasters."
➢ (Optional: Can include quick onboarding slides for first-time users).

- Transition: Automatically transitions to the Home Screen after a short delay or upon user interaction.

### 3.2. Home Screen

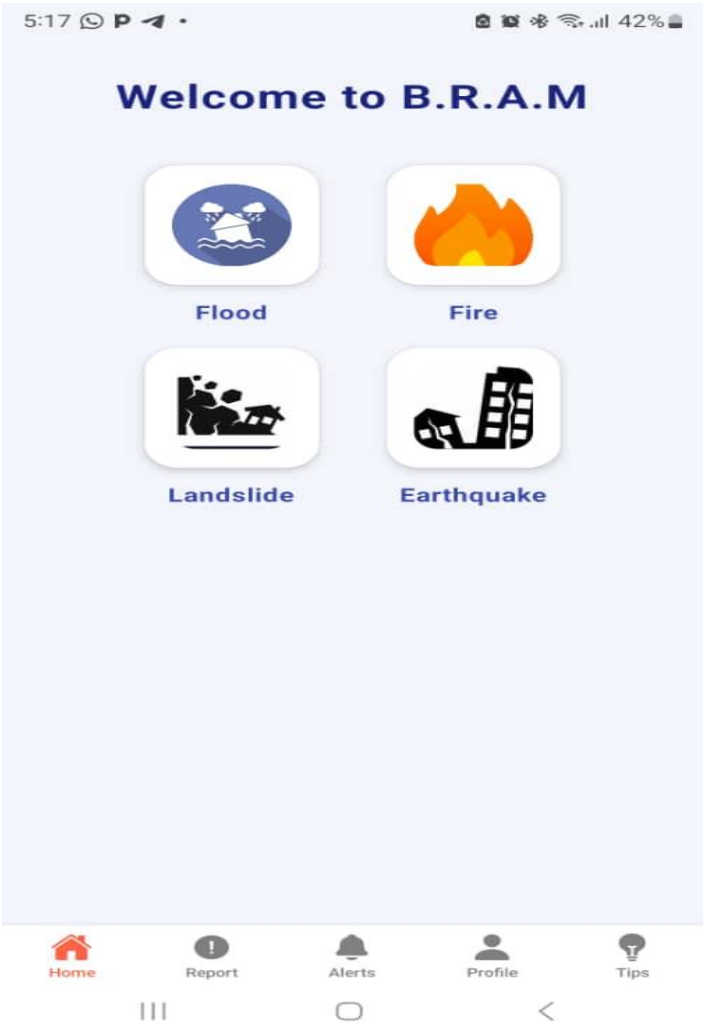
- **Purpose:** The central hub for overviewing disaster types and navigating to specific disaster feeds.
- **Content:**

➢ "Welcome to B.R.A.M." at the top.
➢ A grid of four prominent icons representing the main disaster types:

✓ Flood
✓ Fire
✓ Earthquake
✓ Landslide

- **Functionality:**

Clicking on any disaster icon (e.g., Flood) navigates the user to the respective Disaster Feed Screen (e.g., Flood Feed) to view detailed reports related to that specific disaster type on the report screen..




### 3.3. Report Screen

- Purpose: Allows authenticated users to report disaster incidents.
- Content:

➢ "Select Disaster Type" Section:
➢ Large, tappable icons for: Flood, Fire, Landslide, Earthquake.
➢ Selecting an icon highlights it.
➢ Media Upload:

✓ A button or icon (e.g., a camera icon) to "Upload Image/Video." (For demo, this simulates picking a file without actual media storage outside of a base64 string or URI placeholder).

➢ Description Input:

A text input field for the user to provide a brief description of the disaster (e.g., "Water levels rising near checkpoint street," "Smoke seen from at certain  building location").

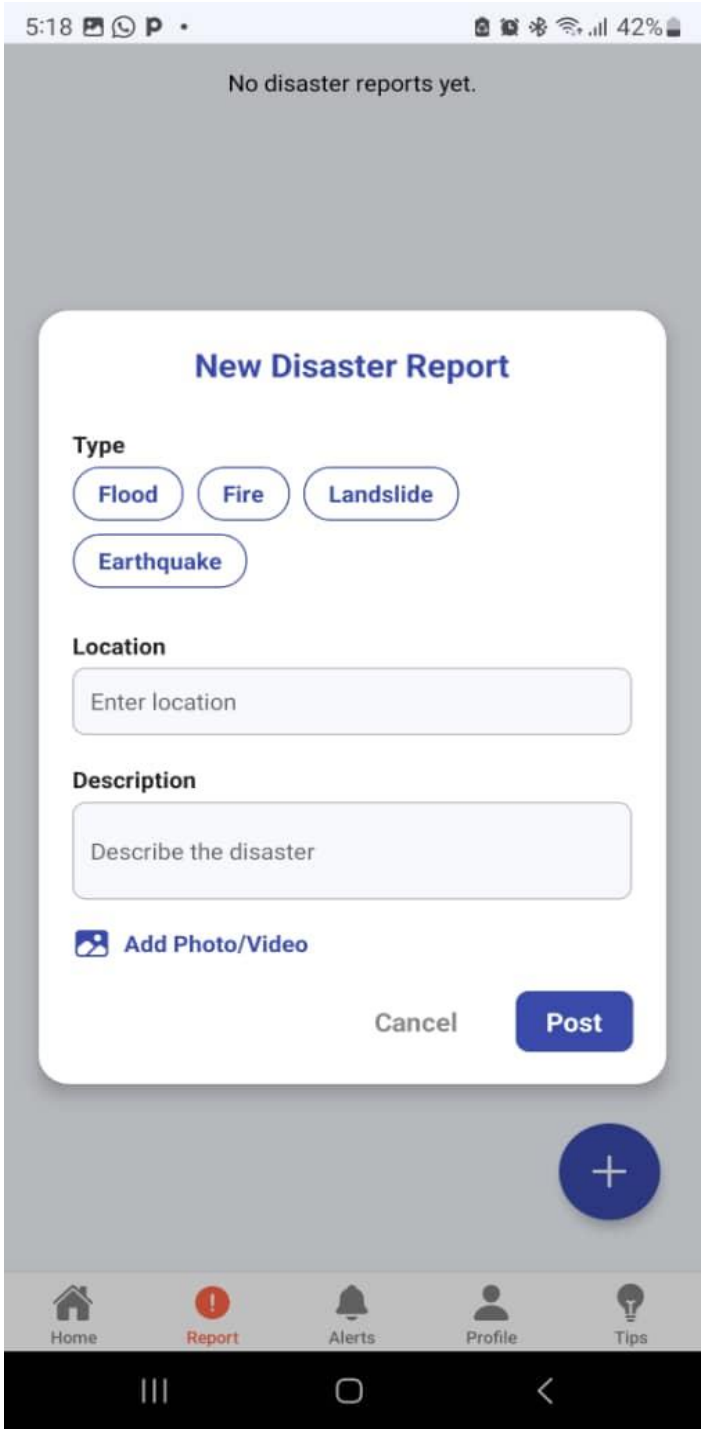
- Location Input:

A text input field for the user to manually enter the location of the disaster. (For future versions, this could integrate GPS).

- "Submit Report" Button: Initiates the report submission process.
- Functionality:

Authentication Check: Before a user can submit a report, the app checks their login status.If not logged in, a modal or alert message appears: "Please register or log in to post a disaster report." The submission is blocked. If logged in, the report proceeds.

- Local Storage: Upon submission, the report data (disaster type, description, simulated media reference, location, and the reporting user's ID/username) is saved locally using MySQL Database server.
- Update Disaster Feed: The newly submitted report immediately appears in the corresponding Disaster Feed Screen (e.g., if a Flood is reported, it shows up on the Flood Feed screen).
- Alert Indication: A small red dot is displayed on the Alerts tab icon, and potentially on the specific disaster type icon within the Alerts screen, indicating a new report.




### 3.4. Disaster Feed Screen (Dynamic, accessed from Home)

- Purpose: This Displays a list of all reported incidents for a specific disaster type showing the list of all the reports made by users.
- Content: A list (e.g., FlatList component) of individual disaster report cards.

➢ Each report card includes:

✓ (Simulated) media thumbnail (image/video).
✓ Brief description provided by the user.

- ✓ Location of the reported disaster.
- ✓ Timestamp of the report.
- ✓ Name/ID of the reporting user.

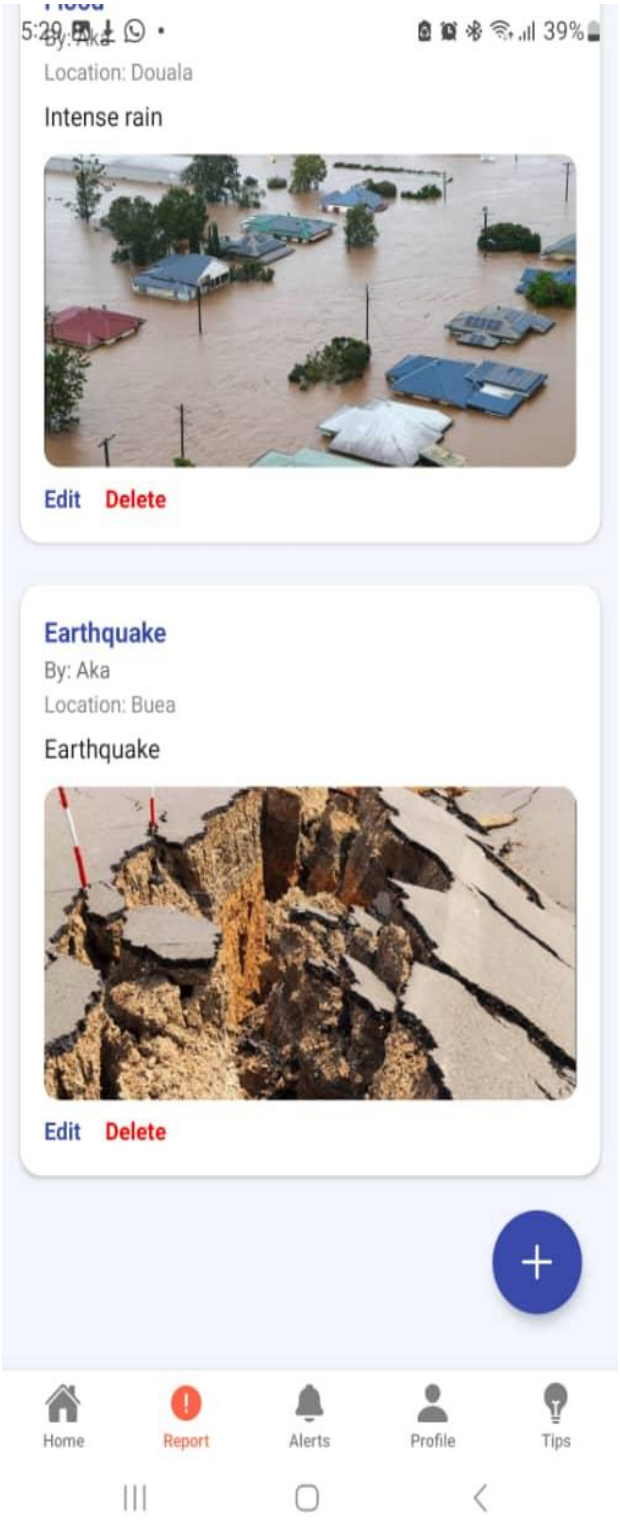
- Functionality:

- ➢ Viewing: All users (logged in or not) can view reports on this screen. Where they can editing/deletionby the user him or her self who uploaded the disaster .
- ➢ (Authenticated User):

Only the user who posted a particular disaster report can see options (e.g., an "Edit" icon, "Delete" icon) to modify or remove their own report.

Clicking "Edit" would pre-populate the Report Screen with the existing data for modification.

Clicking "Delete" would prompt for confirmation and then remove the report from local storage and the feed.




### 3.5. Alerts Screen

- Purpose: Provides a centralized view of all active or recently reported disaster alerts.
- Content: A list of active alerts, similar in appearance to the icons on the Home screen.

Each alert item prominently features:

Disaster type icon (Flood, Fire, landslide and Earhquake).

Brief description (e.g., "Flood 30m ago Nearby").

Time since report.

A red dot indicator next to the specific disaster type icon if there are new or unread alerts of that type.

- Functionality:

Displays all reported incidents in a chronological or proximity-based order (for the app, chronological is simpler).

The red dot on the Alerts tab icon (in the bottom navigation bar) serves as a general notification that new reports or significant alerts are available.


### 3.6. Profile Screen

- **Purpose**: Manages user authentication (login/signup) and displays user-specific information.
- **Content (Conditional Rendering):**

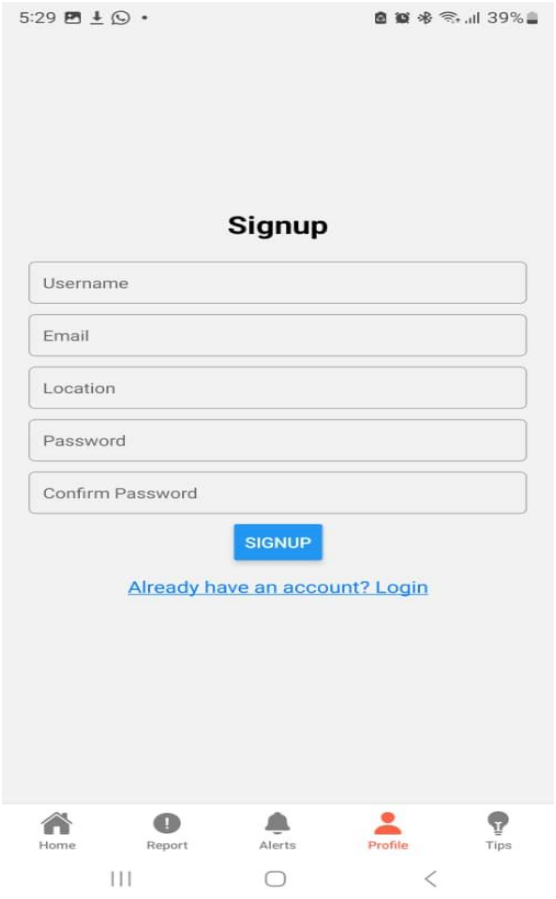
If User NOT Logged InProminent "Welcome to Profile!" message."Login" button.

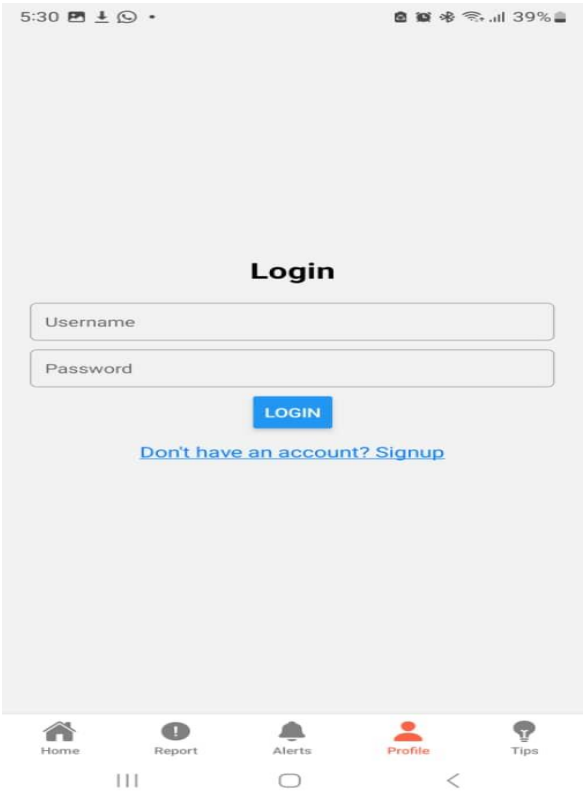
"Register" (or "Sign Up") button.(Optional: Link to "Tips for Self-Disaster Rescue" can be accessible here for unauthenticated users too.). which contain basic information fields like (name,email,location,password,confirmpassword).

If User Logged In it display a"Welcome, [Username]!" message which display of basic profile information (e.g., Username,email, and location).and the "Logout" button. Which show the user can log himself out of the app if he or show likes."

- Functionality:

- Local Authentication: User credentials (username, password) are stored securely using Mysql, and node.js.
- Management: Once logged in, the user's logged-in status and ID are maintained in local storage (Mysql) to persist the session.
- Profile Storage: User profile details (like username) are stored in the MySQL database upon registration.

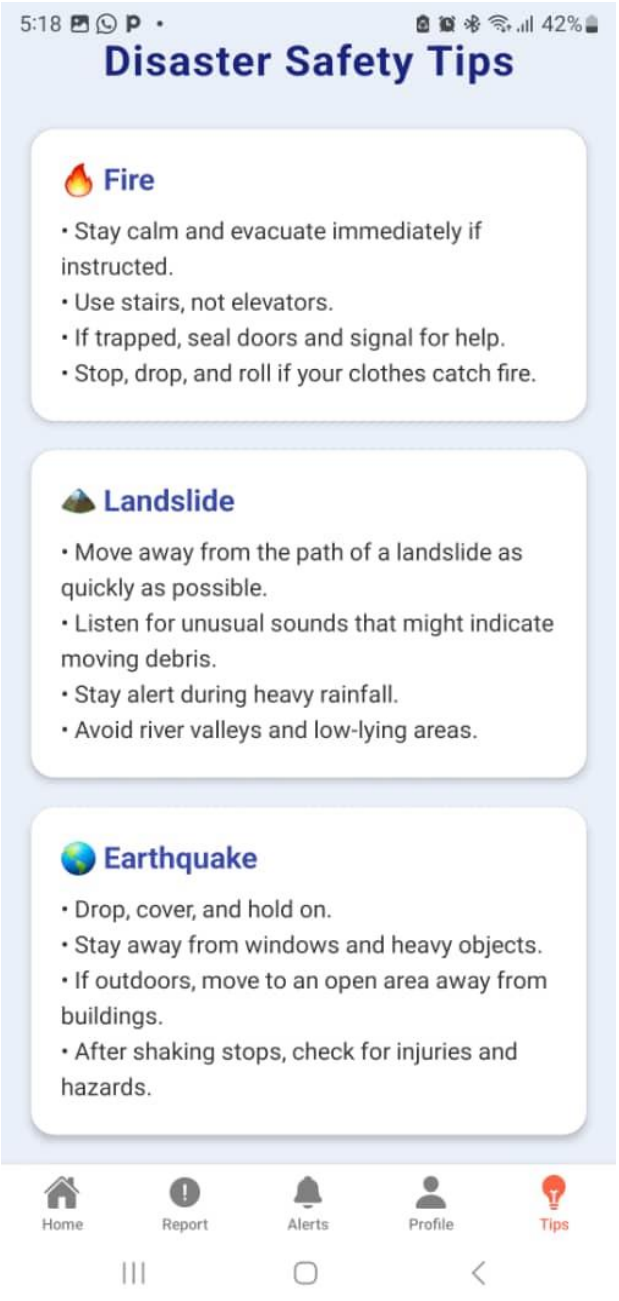



### 3.7. Tips for Self-Disaster Rescue Page:

A dedicated screen or modal accessible from the app navigation bar (and potentially other screens).

Content includes actionable advice for various disaster types (e.g., "What to do during a Flood," "Fire Safety Checklist," "Earthquake Survival Guide").This content is static for the savety tip for user in danger.




**4. Technical Implementation Notes (React Native / JavaScript)**

-

Navigation: react-navigation will be extensively used, primarily createBottomTabNavigator for the main tabs and createStackNavigator for navigating within tabs (e.g., from Home to Disaster Feed).

State Management: React's useState and useContext hooks will manage application state (e.g., isLoggedIn, userProfile, disasterReports).

Local Storage: @react-native storage/MySQL will be the primary tool for persisting user credentials and reported disaster data locally on the device.

UI Components: Standard React Native components (View, Text, Image, TextInput, TouchableOpacity, ScrollView, FlatList, Alert) where used for building the user interface.

Icons: react-native-vector-icons will provide the necessary icons for the tab bar and disaster type representations.

Simulated Media Upload: For the app, media upload will be simulated. Instead of actual file uploads, a placeholder (e.g., a specific URI or simply indicating "Image/Video Uploaded") will be used in the data model.

No Backend Database: Crucially, for this app, there will be no external backend or database. All data persistence is handled via MySQL server, where data is save in the Server.

**5. Future Enhancements (Beyond app)**

- Integration with (e.g., Firebase, AWS Amplify, custom Node.js/Express API) for robust data storage, user management, and scalability.

- The Real-time push notifications for alerts which will help in the displace or disaster alert in real time so as one will be able to know when an incident happen.

- Geolocation services for automatic location tagging of reports and displaying alerts based on user proximity.

- Image/video compression and actual upload to cloud storage.

- Map integration to visualize reported disaster locations.
- User verification (email, phone).

- Search and filter capabilities for disaster reports.
- Community features (comments, upvoting reports).