Mutual Aid Database (M.A.D) – Quick Fire Response

Key Concept

- An app that uses a network of volunteers, local fire departments, and community members to notify nearby users, coordinate assistance, and track resources during a fire.

Features:

Real-time alerts.

- Integrates with local or regional fire department systems (if possible) to send out notifications to users in the impacted area (for example, "Fire spotted at X location").
- Community Water Access: Map or database of fire hydrants, water tanks, or ponds.
- Incident Submission Form
- A simple, intuitive web form where users can input the location (with an address or by dropping a pin on an embedded map) and basic details of the fire (size, visible flames, smoke, injuries, etc.).
- Optional photo upload (from phone or computer).
- Leverage a mapping API (Google Maps, Map box, or OpenStreetMap) to show current fire reports.
- Color-code or size indicators to show severity or urgency level (e.g., small brush fire vs. large-scale building fire).
- Volunteers can list their skills (firefighting experience, first aid, rescue, transportation) and availability.
- Database integration for local emergency shelters or safe zones.

Resource Database

- Catalogs available resources—fire extinguishers, water tanks, shelters, vehicles.
- Integrates with local businesses or homeowners who may volunteer resources (pools, water hoses, etc.)

1. Technical Architecture

a. Front End

- **Framework**: React (or Vue/Angular), which can handle dynamic real-time updates and map integrations smoothly.

- **Responsive Design**: Ensure that the web app is mobile-friendly for users reporting fires on smartphones in the field.

b. Back End

- **Server and API**: Node.js (Express) or Python (Django/Flask) are popular choices.
- **Real-Time Updates**:
    - Use **WebSockets** (Socket.io for Node.js or Django Channels for Python) to push incident updates, volunteer statuses, and chat messages to connected clients in real time.
- **Database**:
    - A relational database like **PostgreSQL** can handle structured data (user profiles, incident logs, resources).
    - Optionally, a NoSQL database like **MongoDB** if you expect unstructured or varied data (images, chat logs, etc.).

c. Mapping & Geolocation

- **Map API**: Google Maps, Mapbox, or OpenStreetMap for location services.
- **Coordinates Storage**: Store latitude and longitude for each incident or resource location.
- **Reverse Geocoding**: Convert lat/long to a human-readable address for better clarity.

d. Notifications & Alerts

- **Browser Push Notifications**:
    - Service workers in modern browsers can send push notifications (e.g., “New Fire Alert in Your Area!”).
- **SMS Integration**:
    - Services like Twilio can send text messages to volunteers or community members without relying on them being in the app.
- **Email Alerts**:
    - Less urgent but still useful for sign-up confirmations or post-incident updates.

User Workflow

1. Report

- A community member sees a brush fire and opens the web app on a phone or computer.
- Fills out a quick form with address or pinned map location, adds a short description, and (optionally) uploads a photo.

2. Alert & Verification

- The system flags it as a new fire incident.
- Local fire station and volunteers in a 5-mile radius get a push notification.

3. Volunteer Dispatch

- Volunteers switch their status to “Responding.”
- The app updates a public dashboard: “3 volunteers en route, Fire Station #2 dispatched.”

4. Resource Coordination

- The system suggests nearby water sources, or local farmland owners who volunteered large water tanks.
- If escalated, official channels are automatically notified.

5. Containment & Transition

- Once fire is controlled, volunteers mark the incident “Resolved.”
- The system transitions to “Recovery” mode: local shelters, donation drives, or medical support for affected people.