BCSE498J Project-II – Capstone Project

Rescue Path

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

21BCE0654 Gaurav Yadav

21BCE3813 Shaivya Srivastava

Under the Supervision of

Dr. Manoov R

Assistant Professor Sr. Grade 1

School of Computer Science and Engineering (SCOPE)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

April 2025

DECLARATION

I hereby declare that the project entitled Rescue Path submitted by me, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering* to VIT is a record of bonafide work carried out by me under the supervision of Dr. Manoov R.

I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date :

Signature of the Candidate

CERTIFICATE

This is to certify that the project entitled Rescue Path submitted by Gaurav Yadav 21BCE0654 **School of Computer Science and Engineering**, VIT, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering*, is a record of bonafide work carried out by him under my supervision during Winter Semester 2024-2025, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The project fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date :

Signature of the Guide

Internal Examiner **External Examiner**

Dr. Umadevi K S

Computer Science and Engineering

ACKNOWLEDGEMENTS

I am deeply grateful to the management of Vellore Institute of Technology (VIT) for providing me with the opportunity and resources to undertake this project. Their commitment to fostering a conducive learning environment has been instrumental in my academic journey. The support and infrastructure provided by VIT have enabled me to explore and develop my ideas to their fullest potential.

My sincere thanks to Dr. Jaisankar N, Dean - School of Computer Science and Engineering (SCOPE), for his unwavering support and encouragement. His leadership and vision have greatly inspired me to strive for excellence.

I express my profound appreciation to Dr. Umadevi K S, the Head of the Software Systems, for her insightful guidance and continuous support. Her expertise and advice have been crucial in shaping throughout the course. Her constructive feedback and encouragement have been invaluable in overcoming challenges and achieving goals.

I am immensely thankful to my project supervisor, Dr. Manoov R, for his dedicated mentorship and invaluable feedback. His patience, knowledge, and encouragement have been pivotal in the successful completion of this project. My supervisor's willingness to share his expertise and provide thoughtful guidance has been instrumental in refining my ideas and methodologies. His support has not only contributed to the success of this project but has also enriched my overall academic experience.

Thank you all for your contributions and support.

Gaurav Yadav

EXECUTIVE SUMMARY

The RescuePath project addresses a critical gap in modern disaster management by providing an intelligent, real-time, and user-friendly platform to assist individuals during natural calamities. With the increasing frequency and unpredictability of natural disasters such as floods, wildfires, earthquakes, and hurricanes, communities are often left without timely, location-specific guidance. RescuePath emerges as a solution to this problem—an emergency web-based application that empowers users with localized safety recommendations and disaster preparedness resources.

Built using React.js for the frontend and Node.js for backend services, the platform integrates live data from OpenWeatherMap, OpenStreetMap, and YouTube APIs to deliver up-to-date weather alerts, hospital locations, and disaster awareness content. It supports both automatic geolocation and manual city input, enabling users to instantly access nearby emergency facilities and resources relevant to their current disaster context. RescuePath emphasizes accessibility, offering a responsive UI that works across mobile and desktop devices even under low-bandwidth conditions—critical during emergencies.

Functionally, the system offers hospital locators, real-time weather alerts, emergency contacts, and disaster checklists. The data is visualized using interactive maps and categorized modules to improve decision-making in high-stress situations. The application's design ensures scalability and reliability, incorporating a modular architecture using Docker for cloud deployment. Its testing strategy employed unit, integration, and usability testing with tools like Jest, Cypress, and Lighthouse.

Key outcomes include a streamlined user experience, accurate disaster tracking, and rapid access to emergency support services. Furthermore, the platform's lightweight build, open-source frameworks, and modular design make it cost-effective and easy to maintain or upgrade.

Looking ahead, RescuePath aims to expand by integrating additional disaster types, multi-language support, offline access, and predictive analytics using AI. By aligning with societal needs and leveraging advanced technologies, RescuePath holds the potential to become a vital community-centered disaster management tool that saves lives and enhances resilience in vulnerable regions.

TABLE OF CONTENTS

Sl.No	Contents	Page No.
	Acknowledgement	**iii**
	Executive Summary	**iv**
	List of Figures	
	List of Tables	
	Abbreviations	
1.	**INTRODUCTION**	**1**
	1.1 BACKGROUND	**1**
	1.2 MOTIVATIONS	**1**
	1.3 SCOPE OF THE PROJECT	**2**
2.	**PROJECT DESCRIPTION AND GOALS**	**5**
	2.1 LITERATURE REVIEW	**5**
	2.2 RESEARCH GAP	**8**
	2.3 OBJECTIVES	**9**
	2.4 PROBLEM STATEMENT	**10**
	2.5 PROJECT PLAN	**10**
3.	**TECHNICAL SPECIFICATION**	**12**
	3.1 REQUIREMENTS	**12**
	3.1.1 Functional	**12**
	3.1.2 Non-Functional	**13**
	3.2 FEASIBILITY STUDY	**14**
	3.2.1 Technical Feasibility	**14**
	3.2.2 Economic Feasibility	**14**
	3.2.2 Social Feasibility	**14**
	3.3 SYSTEM SPECIFICATION	**15**
	3.3.1 Hardware Specification	**15**
	3.3.2 Software Specification	**16**
4.	**DESIGN APPROACH AND DETAILS**	**18**

	4.1 SYSTEM ARCHITECTURE	**18**
	4.2 DESIGN	**20**
	4.2.1 Data Flow Diagram	**20**
	4.2.2 Class Diagram	**22**
	4.2.3 Workflow Diagram	**23**
	4.2.4 Work Breakdown Structure	**24**
5.	**METHODOLOGY AND TESTING**	**25**
	5.1 METHODOLOGY	**25**
	5.1.1 Module Description	**25**
	5.2 TESTING	**27**
6.	**PROJECT DEMONSTRATION**	**29**
	6.1 SYSTEM ARCHITECTURE	**29**
	6.2 WORKFLOW OVERVIEW	**30**
	6.3 DEMONSTRATION WALKTHROUGH	**30**
7.	**RESULT AND DISCUSSION**	**39**
8.	**CONCLUSION AND FUTURE WORKS**	**40**
9.	**REFERENCES**	**45**
	APPENDIX A – SAMPLE CODE	**47**

List of Tables

Table No.	Title	Page No.
1	Literature Review	5

List of Figures

Figure No.	Title	Page No.
1	Gantt Chart	11
2	Dataflow Level - 0 Diagram	20
3	Dataflow Level - 1 Diagram	21
4	Class Diagram	22
5	Workflow Model Diagram	23
6	Work Breakdown Structure	24
7	Rescue Path Home Page	31
8	Weather and Nearest Hospital	31
9	Calamity Type Page	32
10	Emergency Contacts Page	32
11	First Aid Page	33
12	Emergency Response Videos	34
13	Footer and Navigation	34
14	Navigating Hospitals in Vellore	35
15	Directions to the chosen hospital	36
16	Safe Zone Finder	37
17	Map displaying the safe path	37
18	Directions to the closest open area	38

List of Abbreviations

SPA	Single Page Application
UI	User Interface
UX	User Experience
API	Application Programming Interface
DFD	Data Flow Diagram
HTTP	HyperText Transfer Protocol
DB	Database
RAM	Random Access Memory
AWS	Amazon Web Services
CSS	Cascading Style Sheets
JS	JavaScript
JSX	JavaScript XML

Chapter - 1

1. INTRODUCTION

1.1 BACKGROUND

In recent years, the frequency, intensity, and unpredictability of natural disasters—such as earthquakes, floods, wildfires, hurricanes, and landslides—have markedly increased, largely due to the escalating effects of climate change and rapid urbanization. These catastrophic events pose significant and growing threats to human life, infrastructure, and property across the globe. In such critical situations, timely, accurate, and localized information becomes essential for saving lives, minimizing damage, and ensuring swift evacuation and relief operations. However, traditional emergency response systems often suffer from inefficiencies such as outdated alert mechanisms, lack of real-time updates, and the inability to deliver personalized, location-specific guidance. These limitations can delay response efforts and leave affected populations more vulnerable to harm.

With the rapid advancement of technology, there is now a significant opportunity to modernize disaster management systems. The integration of geostationary satellites, geographic information systems (GIS), artificial intelligence, and real-time weather data through powerful APIs enables the development of intelligent platforms capable of offering dynamic, data-driven safety solutions. These systems can analyze incoming data from multiple sources and generate tailored recommendations—such as identifying the safest evacuation routes, warning of impending hazards in specific neighborhoods, or directing individuals toward shelters based on their precise location.

Such innovations are crucial in bridging the gap between the abundance of raw environmental data and the need for accessible, actionable guidance during emergencies. By empowering individuals and communities with the right tools and information at the right time, these platforms can greatly enhance preparedness, support faster decision-making, and ultimately reduce the devastating impact of natural disasters on human lives and livelihoods.

1.2 MOTIVATION

Natural disasters, by their very nature, are sudden and often unpredictable events that can wreak havoc on communities, causing widespread disruption, injury, and loss of life. In the critical moments following such events, timely and effective communication becomes a lifeline. Unfortunately, conventional emergency response systems frequently fall short in delivering swift, accurate, and personalized evacuation plans. These systems often rely on generic alerts or delayed updates, which can lead to confusion, delayed

decision-making, and, ultimately, preventable casualties and damage.

The RescuePath project emerged from the pressing need to address these shortcomings and transform the way disaster response information is delivered to affected populations. Inspired by the potential of modern technology to save lives, RescuePath seeks to bridge the gap between the vast volumes of available disaster-related data and the actionable guidance that individuals and communities need in real time.

By harnessing cutting-edge tools such as geolocation services, real-time data integration, predictive analytics, and user-centric interfaces, RescuePath offers localized, adaptive, and intuitive guidance during emergencies. Whether it's identifying the safest evacuation routes, alerting users about nearby shelters, or issuing warnings based on live environmental conditions, the platform is designed to empower individuals with the critical information they need—right when they need it.

Ultimately, the goal of RescuePath is to enhance disaster preparedness and responsiveness at both individual and community levels. By providing intelligent, data-driven support during crises, RescuePath has the potential to significantly reduce the loss of life, protect property, and foster greater resilience in disaster-prone regions.

1.3 SCOPE OF THE PROJECT

RescuePath is a web-based disaster management platform focused on providing real-time, localized safety recommendations. The scope of the project is structured across its current functionality, planned expansions, operational constraints, and the communities it aims to serve.

Core Functional Scope:

1. City-Specific Safety Recommendations:
   Enables users to receive disaster-specific safety guidance based on geolocation or manual city input.

2. Evacuation Route Suggestions:
   Offers clear evacuation pathways using GIS mapping and elevation data.

3. Real-Time Weather Integration:
   Incorporates weather API data to provide live updates on evolving disaster conditions.

4. Disaster-Type Specific Alerts:
   Initially supports floods and wildfires, delivering tailored alerts and guidance based on the disaster type.

5. Interactive Map Interface:
   Displays danger zones, nearby shelters, hospitals, and safe zones on a responsive map.

6. Emergency Contact & Hospital Finder:
   Provides quick access to local emergency services based on the user's region.

7. Basic Disaster Education:
   Offers checklists, safety tips, and awareness materials for users to prepare before emergencies.

Extended Scope & Future Enhancements:

1. Support for Additional Disasters:
   Expand to include earthquakes, hurricanes, and landslides based on regional relevance.

2. Traffic & Terrain-Based Route Optimization:
   Incorporate real-time traffic and elevation data to dynamically update evacuation routes.

3. Multi-Language Support:
   Add language options to ensure accessibility for diverse populations.

4. User Feedback-Driven Improvements:
   Continuously refine features based on real-time user feedback and behavioral analytics.

Limitations and Considerations:

1. Disaster Type Scope:
   Initially limited to floods and earthquakes; other types are planned but not yet integrated.

2. Connectivity Requirements:
   Despite being lightweight, real-time features require stable internet for optimal performance.

3. No Government Integration (Yet):
   Not directly connected to government agencies or verified emergency alert systems at this stage.

4. Ethical & Privacy Considerations:
   Must ensure location data is handled securely and responsibly.

Target Audience and Use Cases:

1. General Public:
   Individuals residing in or traveling through disaster-prone regions seeking real-time safety advice.

2. Local Community Groups & NGOs:
   For quick dissemination of disaster alerts and coordination of grassroots response.

3. Urban Planners and Safety Officers:
   May use platform data as a supplemental tool for evacuation planning and awareness drives.

4. Students and Educational Institutions:
   Use RescuePath for disaster preparedness education and drills.

5. Travelers and Tourists:
   Access safety resources in unfamiliar areas during emergencies.

Chapter - 2

2. PROJECT DESCRIPTION AND GOALS

2.1 LITERATURE REVIEW

Table 1: Research papers

Title	Methodology	Key Findings	Author(s)/Source
How Geospatial Technologies Aid in Effective Disaster Management	Investigates the application of GIS and remote sensing in disaster management, emphasizing data collection, spatial analysis, and real-time visualization.	Geospatial technologies enhance all disaster management stages, from preparedness to recovery.	Innovation News Network Editorial Team
Exploring the Use of AI and IoT in Disaster Management	Examines the integration of AI and IoT in emergency scenarios to facilitate accurate disaster forecasting and efficient resource allocation.	AI and IoT enable faster, more accurate disaster anticipation and response, reducing casualties and damage.	IoT2Market Research Team
Smart City Integration: How IoT is Reducing Emergency Response Times and Saving Lives	Investigates IoT applications in urban emergency response systems, focusing on real-time data transmission and incident detection.	IoT integration in smart cities improves emergency response times and efficiency.	IEEE Public Safety Team

Role of GIS in Disaster Management	Analyzes the role of GIS in collecting, analyzing, and visualizing geospatial data during disasters.	GIS is crucial for effective emergency management strategies.	Ellipsis Drive Contributors
Enhancing Emergency Operations Centers with Artificial Intelligence: A New Frontier in Emergency Management	Explores AI applications in Emergency Operations Centers (EOCs), focusing on data analysis and resource management.	AI enhances data analysis, resource management, and communication in EOCs.	Firehouse Editorial Team
Using Geospatial Data for Natural Disaster Management	Discusses the use of geospatial data in disaster preparedness and response, emphasizing the integration of data layers for effective planning.	Geospatial data aids in identifying high-risk areas and planning effective disaster responses.	Planwisely.io Team
AI in Disaster Management: How Artificial Intelligence is Saving Lives in Emergencies	Examines AI's role in emergency management, focusing on predictive analytics and real-time decision-making.	AI improves disaster prediction, resource allocation, and real-time decision-making.	HashStudioz Research Team
Applications of Geospatial Technology/Data in Support of Disaster Management	Reviews geospatial technology applications in disaster management, highlighting mapping and remote monitoring.	Geospatial technologies provide valuable information for emergency preparedness and response.	United Nations Economic Commission for Latin America and the Caribbean (ECLAC)

How IoT Is Revolutionizing Disaster Prevention and Response	Explores IoT's impact on disaster management, focusing on sensor networks and early warnings.	IoT sensors offer early warnings and aid in managing disaster aftermaths.	IoT For All Contributors
AI in Emergency Management: Uses & Challenges	Analyzes AI applications and challenges in emergency management, including communication automation and ethical concerns.	AI automates communication, optimizes resources, and supports real-time decisions in emergencies.	SafetyCulture Editorial Team
Artificial Intelligence as a Tool for Enhancing the Performance of Public Health Emergency Operations Centers	Investigates AI integration in Public Health Emergency Operations Centers, focusing on data analysis and decision support.	AI supports decision-makers by integrating diverse information streams, enhancing situational awareness, and suggesting evidence-based actions.	PMC Article (2025)
Explainable Artificial Intelligence in Disaster Risk Management	Studies the application of explainable AI (XAI) in disaster risk management, addressing transparency and accountability.	XAI enhances trust in AI systems by providing transparent and understandable decision-making processes in disaster scenarios.	ScienceDirect Article (2025)
The Role of AI in Disaster Management	Explores AI's role in disaster management, focusing on predictive analytics and resource allocation.	AI aids in assessing vulnerability, guiding infrastructure updates, and optimizing resource	CentralSquare Article (2024)

		distribution during disasters.	
Remote Sensing in Disaster Management	Discusses the use of remote sensing technologies in monitoring and responding to disasters.	Remote sensing enhances disaster response and recovery by providing timely and accurate data.	NV5 Geospatial Software
AI in Disaster Management: Predicting and Preventing Catastrophes	Examines AI's role in predicting and preventing disasters through real-time analysis and decision-making.	AI enables continuous monitoring of weather patterns and social media feeds, improving response times and effectiveness.	JNAO-Nu Journal Article (2023)

2.2 RESEARCH GAP

Despite progress in disaster management technologies, key gaps remain that limit the development of fully integrated, responsive, and ethical systems. Addressing these challenges is essential to enhance platforms like RescuePath.

1. Data Integration & Interoperability:
   Diverse data sources vary in formats and standards, making seamless integration difficult. This hampers real-time coordination and situational awareness during emergencies.

2. Real-Time Decision Support Limitations:
   GIS platforms struggle with high-volume spatial data and lack real-time analytics capabilities, reducing their effectiveness during fast-evolving disaster scenarios.

3. AI Implementation Challenges:
   AI use in disaster management is limited by data quality, system compatibility, and ethical concerns such as bias and accountability. These must be addressed to ensure trust and reliability.

4. Underutilized Geospatial Technologies:
   Advanced geospatial tools are often confined to hazard mapping. Their broader potential for real-time monitoring, predictive modeling, and response planning remains untapped.

2.3 OBJECTIVES

The primary objective of RescuePath is to develop a responsive, data-driven web platform that delivers real-time, location-specific safety recommendations during natural disasters. The project aims to enhance disaster preparedness and response by integrating advanced technologies and prioritizing user accessibility.

1. Develop a User-Friendly Web Platform:
   By 19th February, create an intuitive application where users can input their city and receive disaster-specific safety guidance.

2. Integrate Real-Time Data Sources:
   By 7th March, incorporate satellite, GIS, and live weather API data to deliver current risk updates.

3. Implement Advanced Risk Assessment Algorithms:
   By 4th April, deploy algorithms that analyze elevation, weather, and satellite data to identify high-risk areas and suggest evacuation routes.

4. Ensure Platform Accessibility:
   By 10th April, make the platform accessible across devices and inclusive of users with disabilities.

5. Establish Feedback Mechanisms:
   By 20th April, add features for users to provide feedback and help improve the platform.

6. Monitor and Evaluate Performance:
   Conduct weekly evaluations to assess real-world performance and apply necessary improvements.

2.4 PROBLEM STATEMENT

In recent years, the frequency and intensity of natural disasters have increased, leaving communities vulnerable and often unprepared. During such emergencies, individuals lack access to **timely, accurate, and location-specific safety information**, which is critical for effective evacuation and risk mitigation.

The core problem is the absence of a unified, real-time platform that integrates diverse data sources—such as geostationary satellites, GIS, and live weather APIs—to deliver personalized, city-specific safety recommendations.

Existing systems either operate in silos or offer generalized insights, making it difficult for users to take quick, informed actions when it matters most. This gap in real-time, data-driven guidance not only delays emergency response but also increases the potential for loss of life and property. Addressing this challenge requires an accessible, intuitive platform that bridges the divide between raw data and actionable disaster response.

2.5 PROJECT PLAN

Phase	Duration	Key Activities
1. Project Initiation	2 weeks	- Define project scope and objectives - Identify stakeholders
2. Requirements Gathering	1 week	- Analyze existing systems - Document functional and non-functional requirements
3. System Design	3 weeks	- Design system architecture - Develop data integration framework - Create user interface mockups
4. Development Phase	4 weeks	- Set up development environment - Implement backend services - Develop frontend application - Integrate real-time data sources
5. Testing Phase	2 weeks	- Perform unit testing - Conduct system integration testing - Execute user acceptance testing

6. Deployment and Training	2 weeks	- Deploy system to production environment - Conduct user training sessions - Prepare user manuals and documentation

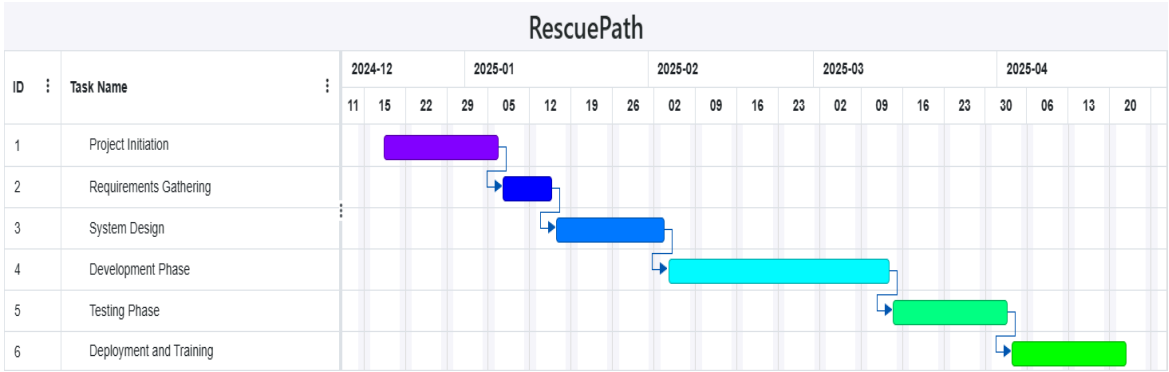

Figure 1: Gantt Chart

Chapter - 3

3. TECHNICAL SPECIFICATION

3.1 REQUIREMENTS

3.1.1 Functional Requirements

1. User Location Acquisition
   → Automatically detect user location using HTML5 Geolocation API.
   → Allow manual city/address input with autocomplete support.

2. Emergency Resource Search
   → Fetch nearby hospitals and medical facilities using third-party APIs (e.g., Google Places).
   → Display results on an interactive map with real-time filtering.

3. Multimedia Disaster Awareness
   → Embed educational videos using YouTube iFrame API.
   → Allow in-app dynamic switching of videos without redirecting users.

4. Real-Time Notifications and Alerts
   → Deliver push and SMS alerts for natural disaster warnings using cloud messaging platforms.
   → Ensure alerts are based on user location and relevant disaster type.

5. Reporting and Analytics
   → Log user interactions, alert triggers, and incident reports for analytical insights.
   → Generate post-incident analytics to refine emergency protocols and improve recommendations.

6. Feedback System
   → Allow users to rate the usefulness of alerts and provide suggestions for improvement.
   → Store feedback in a secure database for further analysis.

7. Disaster Type Customization
   → Adapt UI content and alert logic dynamically based on the type of disaster (e.g., flood vs wildfire).

→ Offer disaster-specific preparedness checklists and tips.

3.1.2 Non-Functional Requirements

1. Performance
   → Ensure API response time remains under 2 seconds even during peak loads.
   → Optimize map rendering and data calls for minimal latency.

2. Scalability
   → Support dynamic scaling using Docker containers and Kubernetes orchestration.
   → Handle thousands of concurrent users during emergency spikes.

3. Security
   → Enforce HTTPS protocol across the platform.
   → Implement secure API key management and use OAuth 2.0 for user authentication.

4. Usability
   → Design a lightweight, responsive UI accessible on low-bandwidth devices.
   → Maintain a consistent and intuitive user experience across desktop and mobile platforms.

5. Reliability
   → Guarantee 99.9% uptime through load balancing, error handling, and fallback mechanisms.
   → Ensure system resilience with data redundancy and failover strategies.

6. Maintainability
   → Build a modular, microservices-based architecture to enable easy debugging, testing, and future updates.
   → Maintain thorough documentation and use version control for smooth collaboration.

3.2 FEASIBILITY STUDY

3.2.1 Technical Feasibility

The technical feasibility of the project is assessed based on the required technologies, resources, and expertise to successfully develop and deploy the platform. The system will require integration with geospatial data APIs, IoT devices, and cloud-based infrastructure. Using modern technologies like React.js for frontend development, Node.js with Express.js for backend services, and cloud platforms (AWS, Google Cloud, or Azure) for hosting, the platform's technical implementation is feasible within the specified time frame. Additionally, the integration of third-party services such as Google Places API for nearby hospital searches and YouTube iFrame API for multimedia content is achievable, considering the available resources and technical expertise.

3.2.2 Economic Feasibility

The economic feasibility evaluates the financial costs of developing, deploying, and maintaining the platform. The initial costs will include expenses for cloud hosting, API usage fees, development tools, and salaries for the development team. Given the scalability of cloud services and the use of open-source technologies, the project can be executed with a manageable budget. Furthermore, by leveraging cloud-native solutions like Docker and Kubernetes for auto-scaling, the platform can optimize resource usage, ensuring cost efficiency as user traffic fluctuates. In the long term, the platform could generate revenue through partnerships with local authorities, government bodies, or disaster relief organizations, covering maintenance and operational costs.

3.2.3 Social Feasibility

The social feasibility of the platform is crucial for its success, as it aims to provide real-time safety and disaster management information to communities. Given the increasing frequency of natural disasters and the growing reliance on mobile and web-based applications for emergency response, the platform aligns with societal needs. The platform's user-friendly interface and accessibility features (e.g., multilingual support, responsive design) ensure that it can serve a wide range of individuals, including those with disabilities. Additionally, public trust and engagement are essential for the platform's adoption, which can be fostered through effective community outreach, training, and awareness programs.

3.3 SYSTEM SPECIFICATION

3.3.1 Hardware Specification

To ensure optimal performance, scalability, and reliability of the RescuePath platform, the following hardware specifications are proposed for both the server infrastructure and client-side devices:

Server (Cloud Infrastructure):

- **Processor:**
  Cloud servers will utilize high-performance multi-core processors such as Intel Xeon or equivalent, capable of efficiently handling concurrent data processing, geospatial computations, and real-time API requests.

- **Memory (RAM):**
  A minimum of 16 GB RAM is required to support web application services, API handling, and analytics processes. The infrastructure will be configured to scale memory dynamically based on user load and system demands during disaster peaks.

- **Storage:**
  SSD-based storage with a minimum capacity of 500 GB will be provisioned to ensure fast read/write speeds. Storage systems will be configured with redundancy (e.g., RAID setup or cloud backups) to prevent data loss and ensure high availability.

- **Connectivity:**
  The cloud platform will maintain a high-speed, reliable internet connection with low latency and high throughput to ensure real-time responsiveness and seamless communication with external APIs.

Client Devices:

- **Supported Devices:**
  RescuePath is designed to be accessible via all modern digital devices, including smartphones (Android/iOS), tablets, and desktops/laptops.

- **Browser Compatibility:**
  The platform is compatible with modern browsers that support HTML5, CSS3, and JavaScript, such as Google Chrome, Mozilla Firefox, Microsoft Edge, and

Safari. This ensures cross-platform usability without additional installations.

3.3.2 Software Specification

To deliver a responsive, scalable, and maintainable platform, RescuePath employs a modern web development stack composed of open-source technologies for frontend, backend, and deployment.

Frontend:

- **Languages & Standards:**
  HTML5, CSS3, and JavaScript are used for the structural and visual components of the application, ensuring a responsive and mobile-first design.

- **Framework & Libraries:**
  The frontend is built using React.js, leveraging its component-based architecture for dynamic UI rendering. Routing is managed with React Router, while Axios is used for HTTP requests.

- **Build Tools & Enhancements:**
  Webpack and Babel are employed for module bundling and code transpilation, respectively. Progressive Web App (PWA) support is enabled through service workers to allow offline functionality and improved performance in low-connectivity scenarios.

Backend:

- **Server Environment:**
  The backend is developed in Node.js using the Express.js framework, chosen for its lightweight nature and scalability under high concurrency.

- **Databases:**

  - **NoSQL (MongoDB):** Used for storing flexible, unstructured data such as user logs, feedback, and real-time analytics.

  - **SQL (PostgreSQL):** Used for structured datasets like disaster records, city metadata, and emergency resource databases.

- **Third-Party Integrations:**
  Integration with the following APIs enhances platform capabilities:

    - Google Maps API and Overpass API for location mapping and nearby resource discovery

    - YouTube iFrame API for streaming disaster awareness videos

- **Security & Authentication:**
  OAuth 2.0 protocol is used for secure user authentication. All API keys are managed using environment variables and securely stored to prevent unauthorized access.

Deployment:

- **Cloud Platforms:**
  The platform is hosted on scalable cloud providers such as AWS, Google Cloud Platform, or Microsoft Azure. These platforms offer compute, storage, and monitoring services required for resilient deployment.

Chapter - 4

4. DESIGN APPROACH AND DETAILS

4.1 SYSTEM ARCHITECTURE

RescuePath is built as a React-based Single Page Application (SPA), offering a smooth, dynamic user experience without full-page reloads. The frontend uses reusable React components for features like location input, map display, real-time alerts, and disaster education videos. State management is handled using React Context API to ensure consistent data flow and real-time updates.

The backend is developed with Node.js and Express.js, serving as the bridge between the frontend and external data sources such as weather APIs, GIS services, and emergency databases. It supports secure user authentication using OAuth 2.0 and communicates with MongoDB and PostgreSQL for handling unstructured and structured data respectively.

All frontend-backend communication is handled via RESTful APIs secured over HTTPS. The system is containerized using Docker and deployed with Kubernetes for scalability, load balancing, and fault tolerance. This architecture ensures that RescuePath is fast, modular, secure, and capable of handling high user loads during disaster events.

Key Components

The RescuePath platform is composed of several modular components that collectively ensure a responsive, data-driven, and user-friendly experience.

1. Frontend:

- Built using React.js, enabling the development of reusable UI components and a scalable component-based architecture.

- CSS is used extensively for styling, with a mobile-first, responsive design that adapts across screen sizes.

- External APIs are integrated to fetch real-time data, ensuring dynamic and context-specific user interactions.

2. Routing:

- Managed by react-router-dom to support client-side navigation without page reloads.

- Key routes include:
  /first-aid – Basic emergency instructions
  /emergency – Real-time alerts and hospital information

/calamity-types – Educational content on various disasters
/about – Project background and team information

3. State Management:

- useState and useEffect hooks are used for managing local component state.
- Components such as WeatherAlerts and NearestHospital handle their own data fetching and updates independently, improving modularity and maintainability.

4. API Integration:

- OpenWeatherMap API for weather alerts and environmental conditions.
- OpenStreetMap / Overpass API for locating nearby hospitals and emergency centers.
- YouTube iFrame API for embedding disaster awareness and preparedness videos.

5. Responsive Design:

- Designed with CSS media queries to ensure optimal performance across smartphones, tablets, and desktops—even under bandwidth-constrained conditions.

4.2 DESIGN

4.2.1 Data Flow Diagram (DFD)

Level 0: Context Diagram

The context diagram identifies the primary actors, which are the User and external APIs providing essential data. The RescuePath Application serves as the central system interacting with the user and APIs.

Actors:

- User: Interacts with the system by searching for hospitals, viewing weather alerts, and accessing disaster-related resources.
- APIs: Provide dynamic data on weather alerts, nearby hospitals, and disaster-related multimedia content.
- System: The RescuePath application itself.

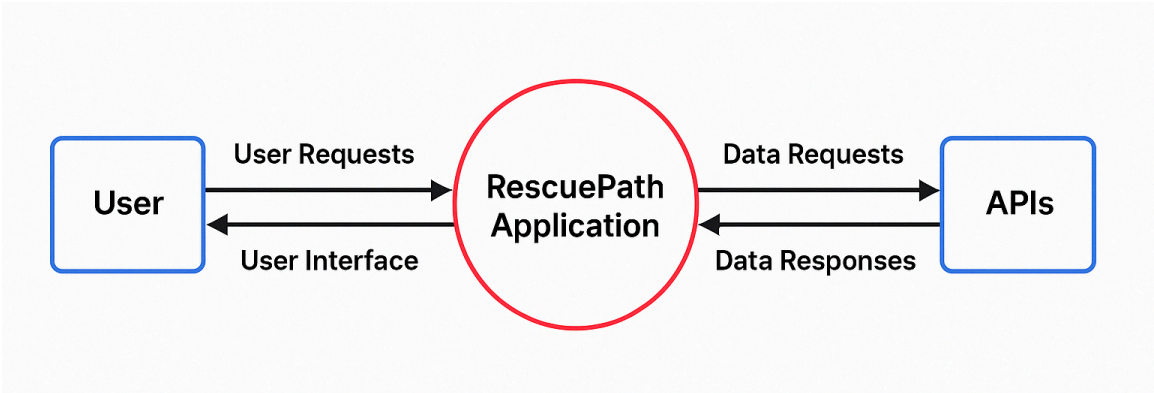

Figure 2: Dataflow Level - 0 Diagram

Level 1: High-Level DFD

This diagram outlines the major processes and data exchanges within the RescuePath application, showing how user input is transformed into meaningful, location-specific disaster guidance.

1. **User Input**
   The user initiates interaction by either entering their current city or allowing geolocation. They may also select specific disaster-related tags or categories such as “First Aid” or “Emergency Numbers” depending on their immediate need.

2. **Get User Input**
   This process captures the input and validates it (e.g., checks if the location is

accurate or if a disaster category was selected). It then forwards this structured input to the system for further processing.

3. **Fetch Data from APIs**
   The system uses the validated input to make real-time requests to three key external data providers:
   - **OpenWeatherMap API**: Provides live weather data and alerts based on the user's location.
   - **OpenStreetMap API**: Returns the locations of nearby hospitals, clinics, and other emergency facilities.
   - **YouTube API**: Supplies relevant educational videos related to the selected disaster type (e.g., how to respond during a flood).
4. **Process & Format Data**
   Once data is fetched, it is cleaned, filtered, and formatted to suit the frontend interface. This includes converting raw API responses into readable weather alerts, visual hospital markers on maps, and embedded video content.
5. **Display Results to User**
   The formatted information is rendered on the user interface. The user sees real-time alerts, a map of emergency resources, and quick access to awareness content—all tailored to their selected location and context.
6. **User Interaction Logs**
   The system records basic interaction data, such as what the user searched for, which content was accessed, and how they navigated. This is stored securely and used to improve system performance, prioritize content, and enhance user experience over time.

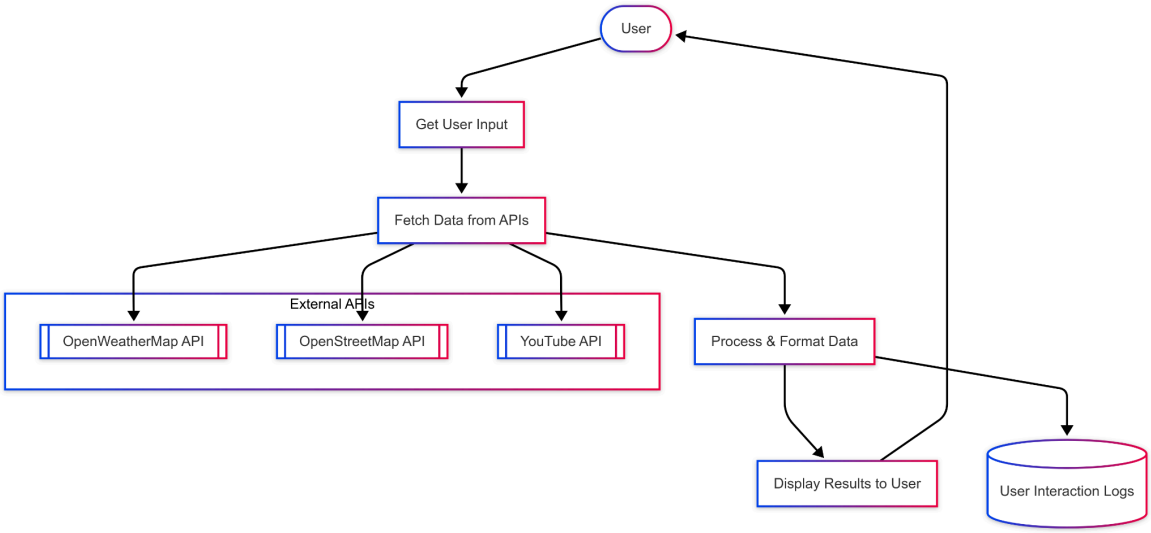

Figure 3: Dataflow Level - 1 Diagram

4.2.2 Class Diagram

Since the project is component-based, the class diagram represents the key React components and their relationships. Here's a basic structure:

Key Relationships:

- RescuePath: The root component responsible for managing routing.
- HomeContent: Contains child components like WeatherAlerts and NearestHospital.
- Footer: Includes modals like PrivacyModal and TermsModal, and contains essential footer information and navigation links.
- WeatherAlerts & NearestHospital: Each of these components handles specific functionality—fetching weather data and hospital locations, respectively. They also manage their own state and data fetching.

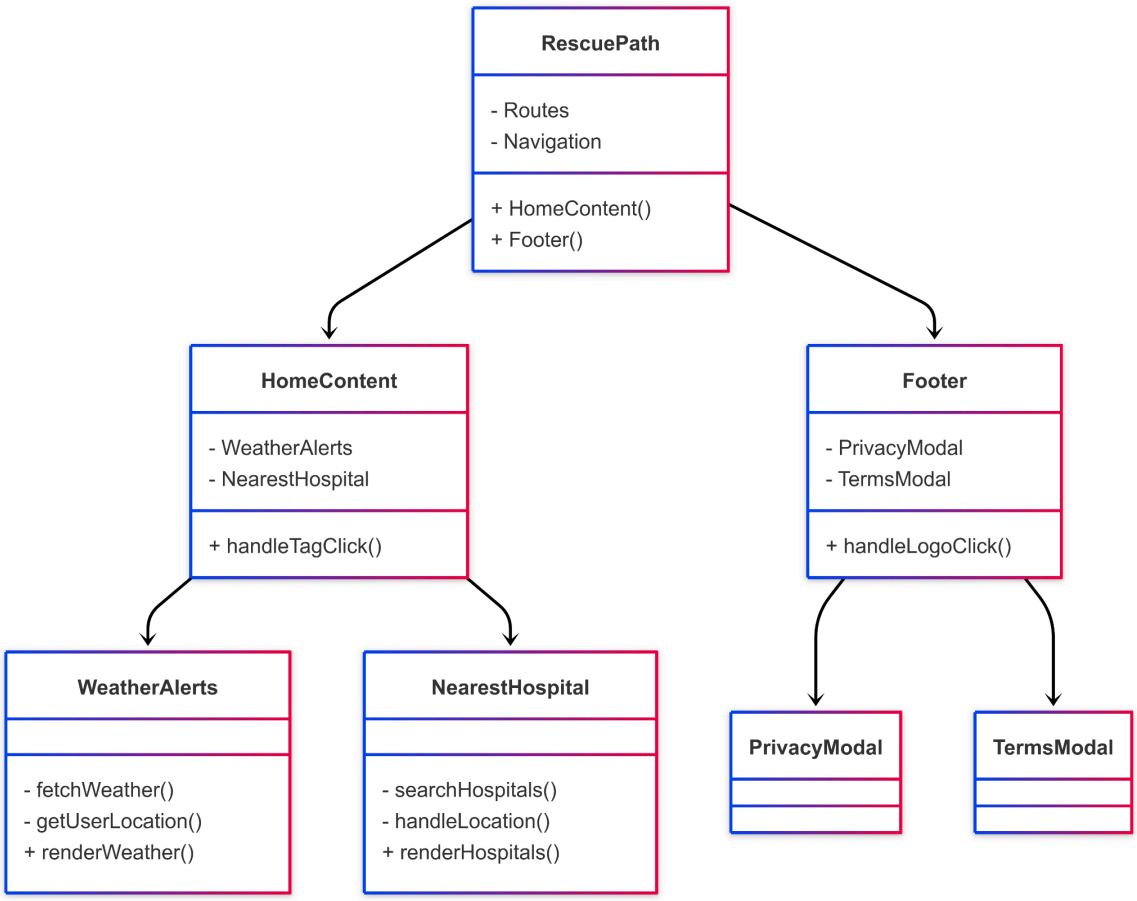

Figure 4: Class Diagram

4.2.3 Workflow Model Diagram

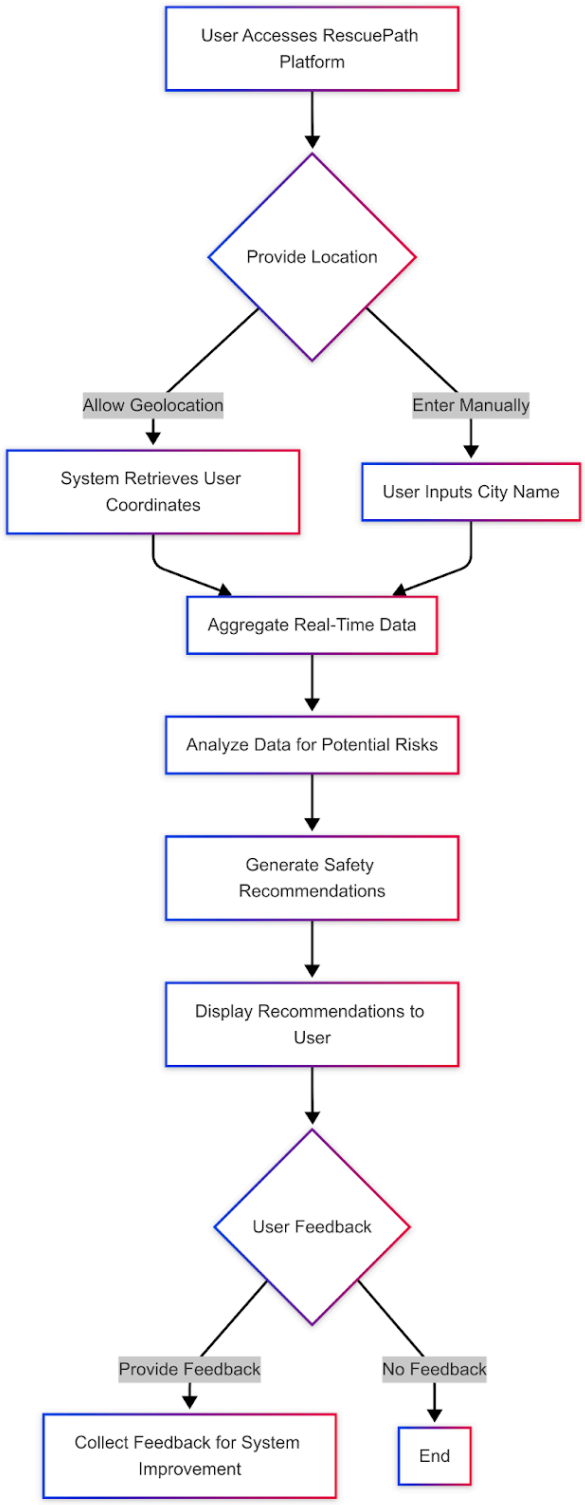

Figure 5: Workflow Model Diagram

4.2.4 Work Breakdown Diagram

The WBS divides the RescuePath project into seven main phases, each broken down into manageable work packages. This structure ensures clarity in task allocation, progress tracking, and resource planning.

1. *Project Planning*
   Establishes the foundation of the project by gathering requirements, defining scope, and setting timelines. It ensures alignment between team roles, deliverables, and deadlines before development begins.
2. *System Design*
   Covers the high-level technical planning of the system. This includes designing the architecture (frontend-backend interaction), identifying React components, and creating user interface mockups and data flow diagrams.
3. *Frontend Development*
   Focuses on building the user interface using React. Tasks include developing reusable components, integrating external APIs (weather, maps, videos), and ensuring the application is responsive across devices.
4. *Backend Development*
   Involves setting up the Node.js server, configuring API endpoints, securing access with OAuth, and managing database operations using MongoDB and PostgreSQL.
5. *Testing*
   Ensures reliability and quality. Unit tests verify component behavior, integration tests validate end-to-end workflows, and UI/UX testing ensures a smooth user experience under realistic conditions.
6. *Deployment*
   Handles packaging and releasing the system. This includes containerizing the app with Docker and deploying to a cloud platform like AWS or GCP.
7. *Maintenance & Feedback*
   Post-deployment, the system is monitored for performance and reliability. User feedback is collected and analyzed to fix issues and introduce improvements in future versions.

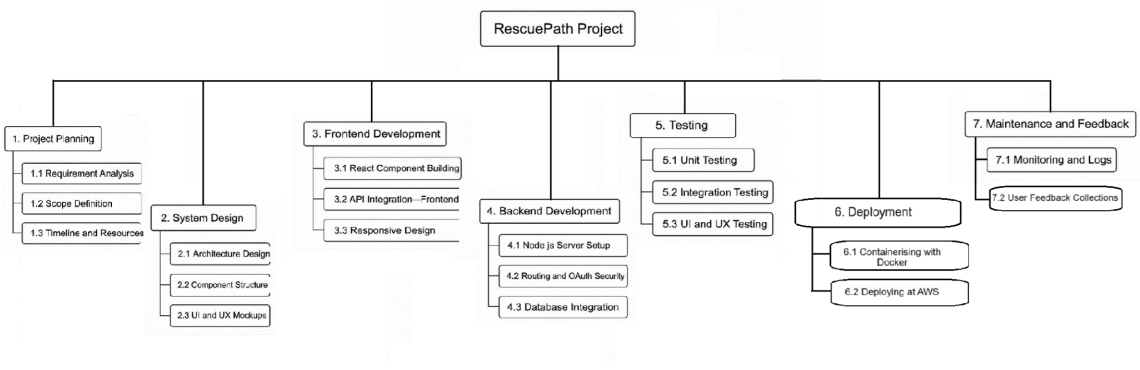

Figure 6: Work Breakdown Structure

Chapter - 5

5. METHODOLOGY AND TESTING

5.1 METHODOLOGY

The methodology adopted for RescuePath follows a modular, component-driven web development approach using the React.js framework. The project is structured as a Single Page Application (SPA), ensuring smooth, real-time interaction without full-page reloads. The development process follows agile principles, allowing iterative development and integration of features based on user feedback and testing.

The application is divided into front-end and back-end modules, integrated via RESTful APIs. The front end uses React components for rendering interactive elements such as location-based search, weather alert displays, and embedded disaster awareness videos. The back end (Node.js with Express) handles API routing, security, and external API communication. MongoDB and PostgreSQL are used for storing unstructured and structured data respectively.

The system relies on real-time data integration from multiple third-party APIs including OpenWeatherMap (for weather alerts), OpenStreetMap (for hospital locations), and YouTube (for disaster-related educational content). Data fetched from these sources is processed and formatted before being presented through a responsive UI optimized for both desktop and mobile users.

Each module is tested independently through unit and integration testing. Deployment is handled via Docker and Kubernetes on cloud platforms like AWS or GCP to ensure scalability, fault tolerance, and high availability.

5.1.1 Module Description

RescuePath is built using a modular architecture, with each feature encapsulated in its own component. This allows for better maintainability, code reusability, and efficient state management. Below is an overview of the key modules and their responsibilities:

1. Disaster Type Selection Module

- Users to select from predefined disaster types (e.g., flood, wildfire).
- Dynamically loads relevant safety checklists, response protocols, and awareness videos.
- Improves contextual relevance of alerts and resources shown.

2. Location Search Module

- Supports HTML5 geolocation and manual address input with autocomplete.

- Passes coordinates to backend APIs for retrieving localized data.
- Used as the base input for both weather alerts and hospital search.

3. Popular Searches Module

- Displays quick-access tags like “First Aid”, “Emergency Numbers”, and “Evacuation Tips”.
- On click, loads relevant content without needing user location.
- Improves discoverability of high-priority resources.

4. Weather Alerts Module

- Fetches current weather conditions and disaster alerts using the OpenWeatherMap API.
- Filters and formats data for readability (e.g., severity, region, timestamps).
- Alerts are updated in real time based on user location.

5. Hospital Locator Module

- Uses OpenStreetMap/Overpass API to identify nearby hospitals and emergency care facilities.
- Plots results on an interactive map with markers and info popups.
- Includes distance calculations and sorting by proximity.

6. Educational Content Module

- Embeds YouTube videos relevant to the selected disaster type.
- Includes dynamic switching without leaving the platform (via YouTube iFrame API).
- Helps in educating users about preparation, safety, and first aid.

7. Feedback & Logging Module

- Captures basic user interaction data and optional feedback.
- Logs are stored for future analysis and iterative improvement of the system.

5.2 TESTING

Testing plays a crucial role in ensuring the reliability, functionality, and usability of the disaster management application. A combination of unit testing, integration testing, functional testing, performance testing, and usability testing was carried out using both automated tools and manual evaluation techniques.

5.2.1 Unit Testing

Unit testing focused on verifying the correctness of individual components in isolation. We used Jest and React Testing Library to ensure that elements like dropdown menus, input fields, and buttons rendered correctly and responded as expected. For instance, we tested whether the disaster type dropdown displayed all available options, whether the location input field accepted text properly, and whether clicking on tags like "Medical Centers" triggered the right actions. We also validated the behavior of reusable components like WeatherAlerts and NearestHospital, ensuring they handled props, loading states, and empty data correctly.

5.2.2 Integration Testing

Integration testing was carried out using Cypress to test how different modules work together within the application. This included simulating a full user flow—selecting a disaster type, entering a location, and submitting the form—to confirm that the correct data was fetched and displayed. We also mocked API responses to test how the app handled and rendered weather alerts and hospital data. In addition, we validated internal routing behavior across different pages (e.g., navigating from the homepage to /first-aid) to ensure smooth transitions in the Single Page Application (SPA) environment.

5.2.3 Functional Testing

Functional testing was conducted manually to confirm that all core features of the application worked as intended across various environments. We tested the application on different devices (phones, tablets, desktops) and browsers (Chrome, Firefox, Safari) to check layout stability and feature accessibility. Key functions tested included input handling, button clicks, scroll behavior, and navigation via popular search tags. We ensured that clicking on elements like "Evacuation Zones" reliably navigated users to the appropriate content.

5.2.4 Performance Testing

We used Lighthouse, an open-source performance auditing tool available in Chrome DevTools, to assess the application's speed and responsiveness. Metrics such as First Contentful Paint (FCP), Time to Interactive (TTI), and API response times were recorded. We examined the time it took for weather data and hospital locations to load and render after user input, and identified any performance bottlenecks caused by large assets or API delays. Based on these insights, we optimized component rendering and reduced unnecessary re-renders to improve overall efficiency.

5.2.5 Usability Testing

Informal usability testing sessions were held with a small group of users, including peers and non-technical individuals. Testers were asked to use the app freely and provide feedback on layout, ease of use, and overall clarity. From these sessions, we received suggestions to improve the visibility of key buttons, enhance contrast in some UI elements, and simplify certain labels. Based on this feedback, minor UI adjustments were made to improve the intuitiveness and accessibility of the platform for all users.

Chapter - 6

6. PROJECT DEMONSTRATION

The RescuePath project was developed as a full-stack web application designed to assist users in locating emergency resources, understanding disaster preparedness, and accessing life-saving guidance in real-time. This section demonstrates the system's structure and functionality in three stages: system architecture, operational workflow, and user interface walk-through.

6.1 SYSTEM ARCHITECTURE

RescuePath is built using a modular and scalable architecture, following the principles of a Single Page Application (SPA). The system is primarily divided into three core layers: frontend, backend, and data integration via third-party APIs.

- Frontend:
  Developed using React.js, the frontend provides an interactive and responsive interface that enables users to access all disaster-related features without full-page reloads. Key libraries and tools used include React Router for client-side navigation, Axios for API communication, and Tailwind CSS for responsive styling.

- Backend:
  Implemented in Node.js using Express.js, the backend handles routing, processes incoming requests, and connects with external APIs. It provides a lightweight and efficient service layer that returns weather data, hospital locations, and search results to the client side.

- APIs and Data Sources:
  The application integrates several third-party APIs:
  → OpenWeatherMap API for real-time weather alerts.
  → OpenStreetMap/Overpass API for finding nearby hospitals and medical services.
  → YouTube iFrame API for embedding disaster awareness videos. All API keys and tokens are securely managed and fetched through server routes or environment variables.

- Deployment:
  The application is containerized using Docker, and cloud deployment (such as on AWS or GCP) allows scalability and high availability.

6.2 WORKFLOW OVERVIEW

The core project structure is organized into clear modules:

- **/src/components**: Contains reusable components like WeatherAlerts, NearestHospital, TagButtons, and Footer.
- **/src/pages**: Contains routed views such as HomePage, FirstAidPage, CalamityTypePage, and EmergencyContactsPage.
- **/src/App.js**: Acts as the central routing hub using react-router-dom, managing navigation between all pages.
- **/src/utils**: Hosts utility functions such as API fetch handlers and location resolvers.

The application workflow follows this sequence:

1. User lands on the homepage and is presented with a search bar and quick-access disaster tags.
2. Upon entering a location or clicking a tag, state is updated and the relevant component (e.g., WeatherAlerts, NearestHospital) fetches data using the associated API handler.
3. The API responses are rendered dynamically using React state hooks and passed down as props to child components.
4. All results (weather, hospitals, videos, etc.) are shown contextually on a single page without reloading, ensuring a seamless experience.
5. Optional logging modules track user interactions for feedback or usage analytics.

6.3 DEMONSTRATION WALKTHROUGH

The RescuePath platform provides a cohesive, interactive experience to guide users through various aspects of disaster preparedness and emergency resource discovery. Below is a detailed demonstration of how a user navigates and interacts with each section of the application.

Homepage
The landing page of RescuePath is designed with an emergency-themed background to immediately establish context and urgency. It features a prominent search bar where users can look up emergency services, guidelines, or disaster-specific information. Below the search bar are quick-access tags such as “First Aid,” “Emergency Numbers,” “Flood Safety,” “Earthquake Protocol,” and “Medical Centers” that allow users to instantly navigate to relevant content.

The top navigation bar includes links to core sections of the platform including Calamity

Type, Emergency Contacts, First Aid, and About Us. It also provides options to Login or Register for a more personalized experience.

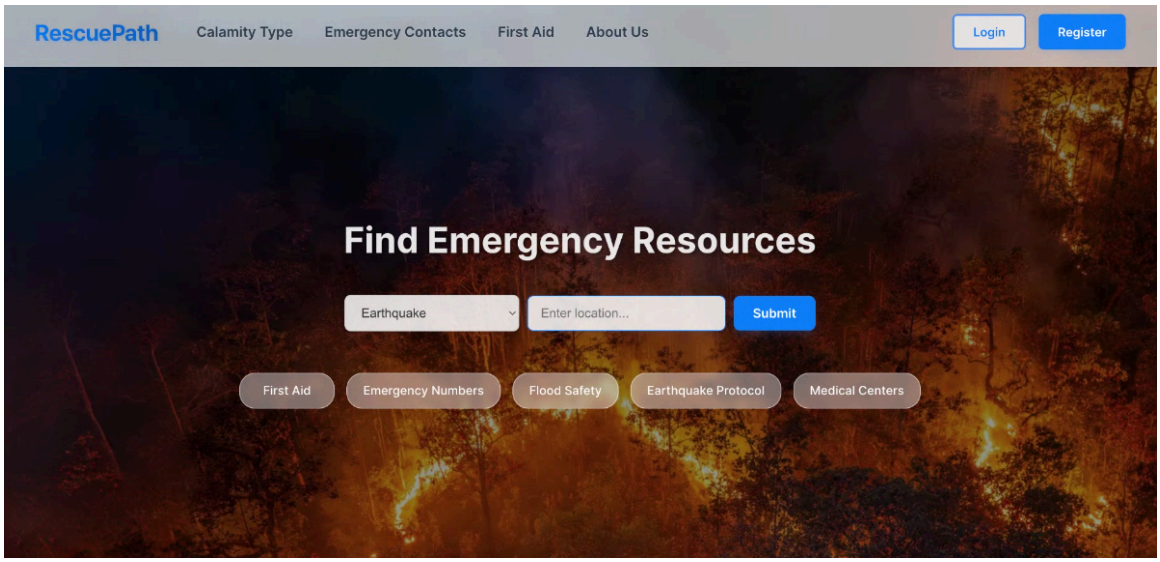

Figure 7: Rescue Path Home page

Weather and Hospitals:
Below the search bar, the weather module displays real-time temperature, humidity, and wind status based on user location. A "Refresh" button allows users to manually re-fetch data. The hospital locator enables users to find the nearest medical facilities by typing their city or address.

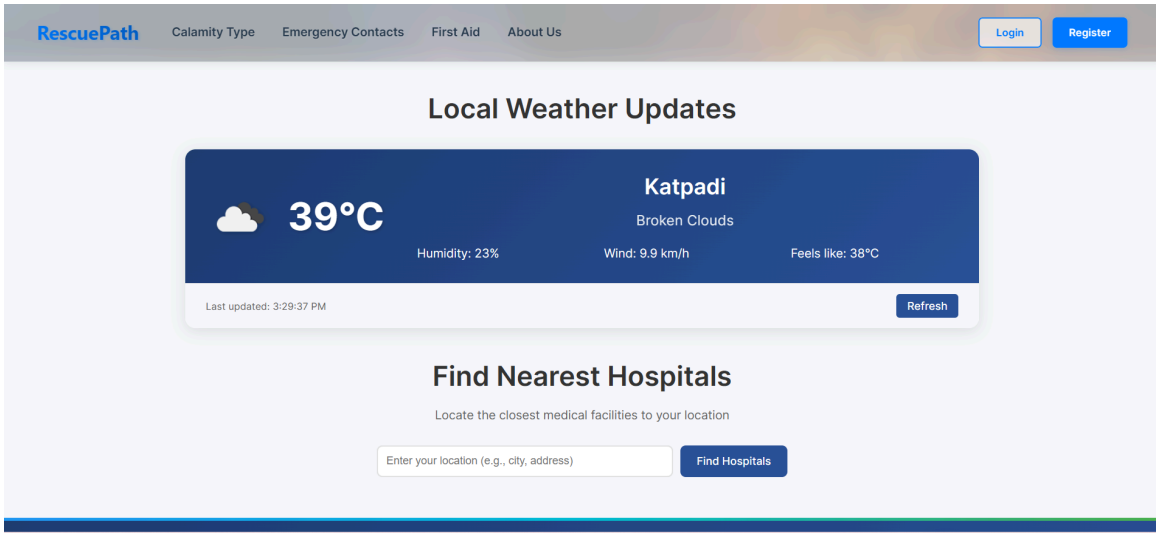

Figure 8: Weather and Nearest Hospital

Calamity Type Page:
This section lists various disasters—Earthquakes, Floods, Wildfires, Tsunamis, and more—each with a brief explanation and recommended safety measures. Users can click "Discover Safety" buttons for detailed guidance specific to each calamity.

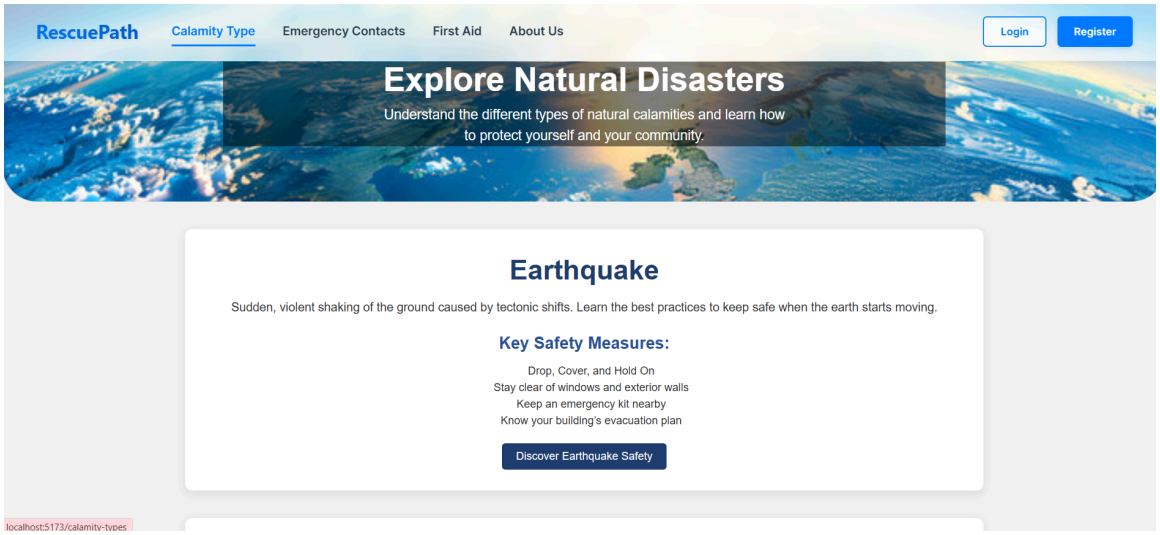

Figure 9: Calamity Type Page

Emergency Contacts Page:
Emergency numbers for ambulance, fire, police, and disaster response are displayed in card format. Secondary helplines such as child, women's, and senior citizen services are also provided. A FAQ section at the bottom answers common questions on emergency planning.

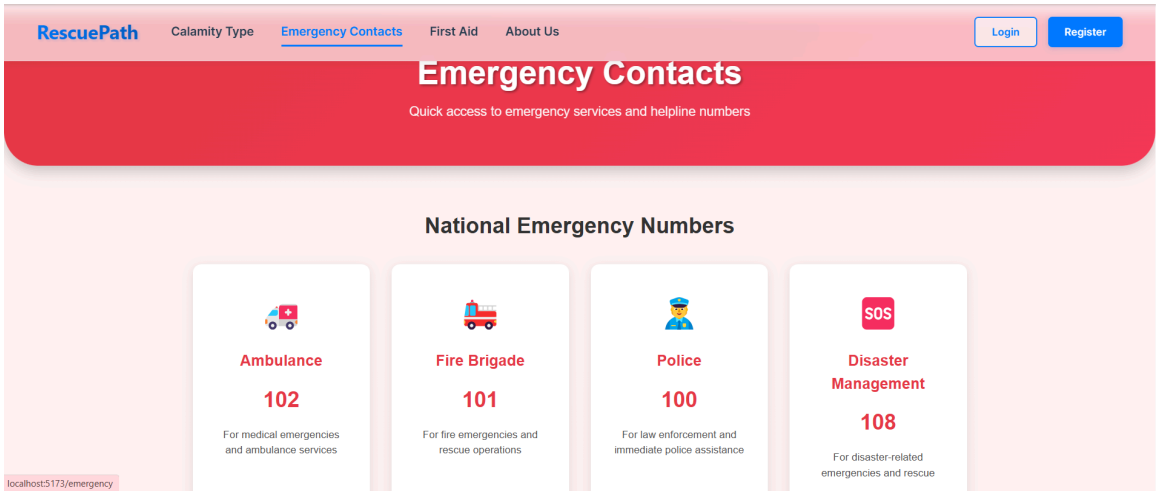

Figure 10: Emergency Contacts Page

First Aid Page:

This section is dedicated to life-saving medical procedures and is titled “Emergency First Aid Guide.” Below the header is the “Common Emergency Procedures” section, laid out in clean card-based UI blocks for CPR, Choking, Burns, etc.

Each card offers a short explanation (e.g., “Step-by-step guide to performing CPR on adults, children, and infants”) and a “Learn More” button. Clicking this button leads users to detailed, instructional content, which may include step-based procedures or embedded educational media.

This module is particularly helpful during situations where immediate medical action is needed before professional help arrives.

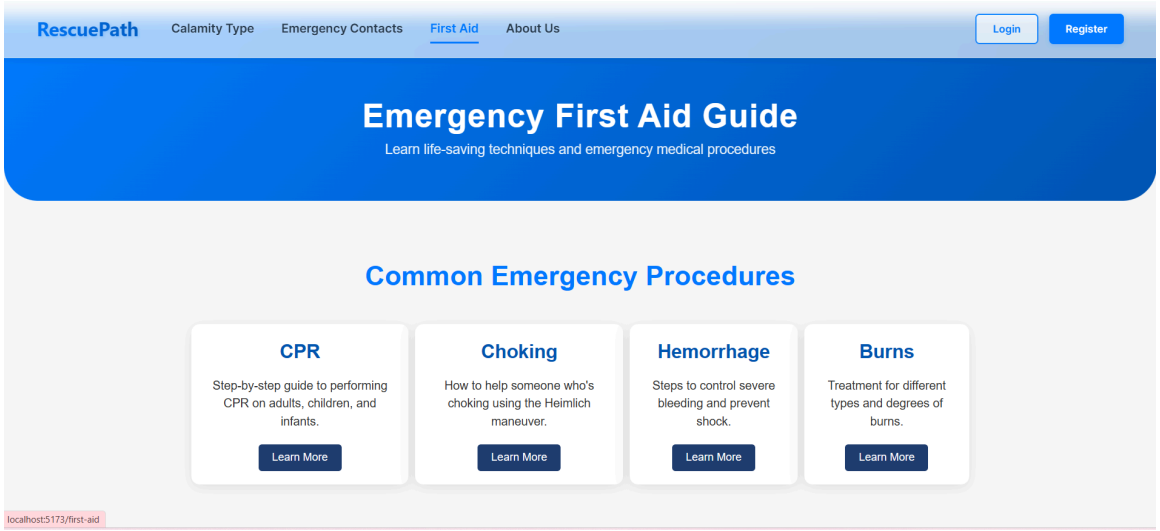

Figure 11: First Aid Page

Emergency Response Videos:
Users can view curated YouTube videos across various emergency topics, available in multiple languages including English, Hindi, and Tamil. This ensures that preparedness content is accessible to a wider audience.

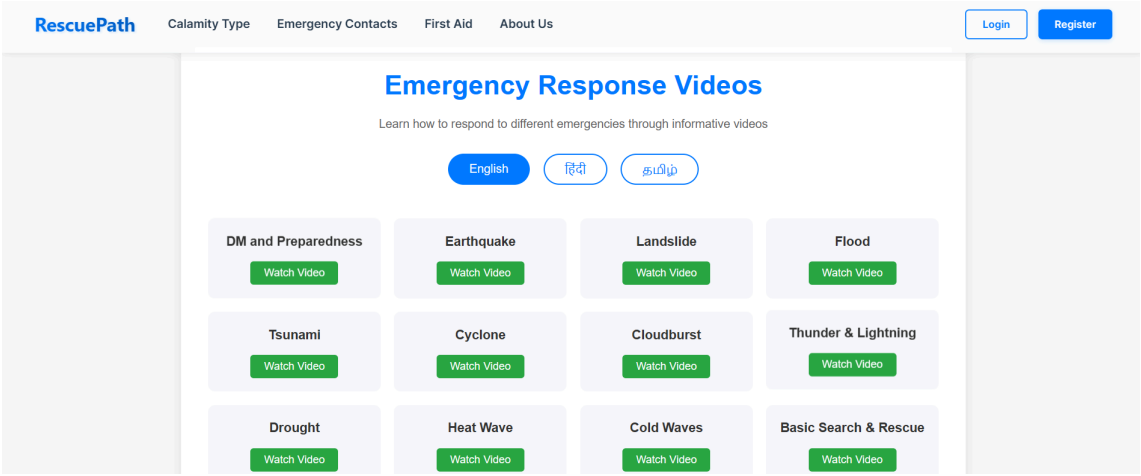

Figure 12: Emergency Response Videos

Footer and Navigation:
Throughout the platform, users can navigate easily using the sticky top navigation bar. The footer provides links to privacy policy, terms and conditions, and includes a tagline reinforcing the mission of RescuePath: "Empowering communities with life-saving information and resources."

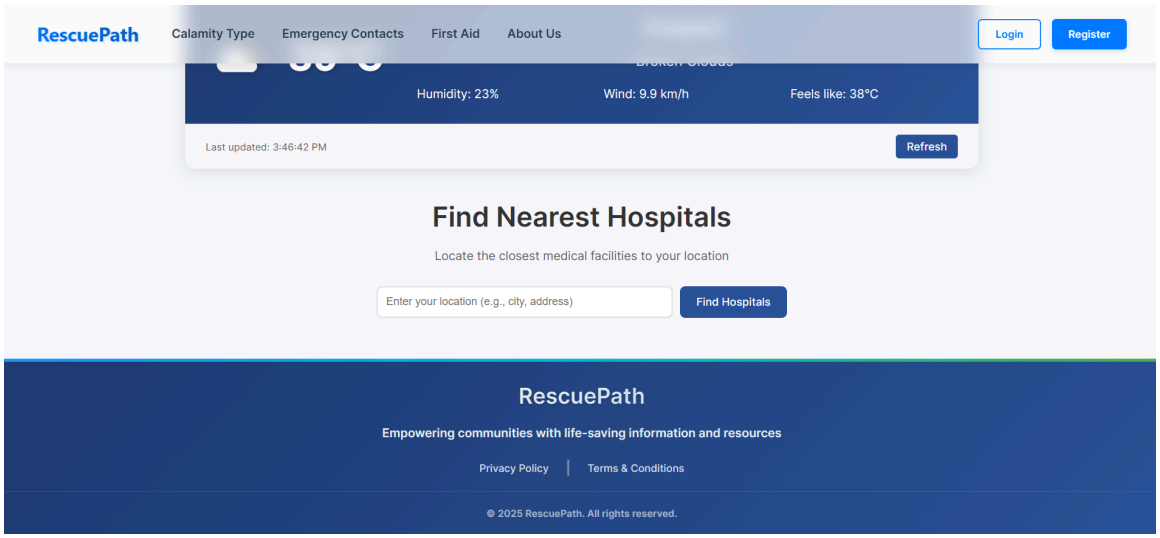

Figure 13: Footer and Navigation

7.3.1 Nearest Hospital Feature Demonstration

One of the core features of RescuePath is the "Find Nearest Hospitals" module, which helps users quickly locate nearby medical facilities based on their current or manually entered location. This feature integrates the OpenStreetMap/Overpass API to retrieve hospital data and display it meaningfully.

Here's how the demonstration works:

1. The user navigates to the "Find Nearest Hospitals" section of the homepage.

2. In the input field provided, the user types the name of their city. For example, entering "Vellore" triggers the system to fetch hospital data related to that location.

3. Upon clicking the "Find Hospitals" button, the application queries the OpenStreetMap API. Within seconds, a list of hospitals in or around Vellore is displayed as individual cards. Each card contains:
   →The name of the hospital
   → A "Get Directions" button

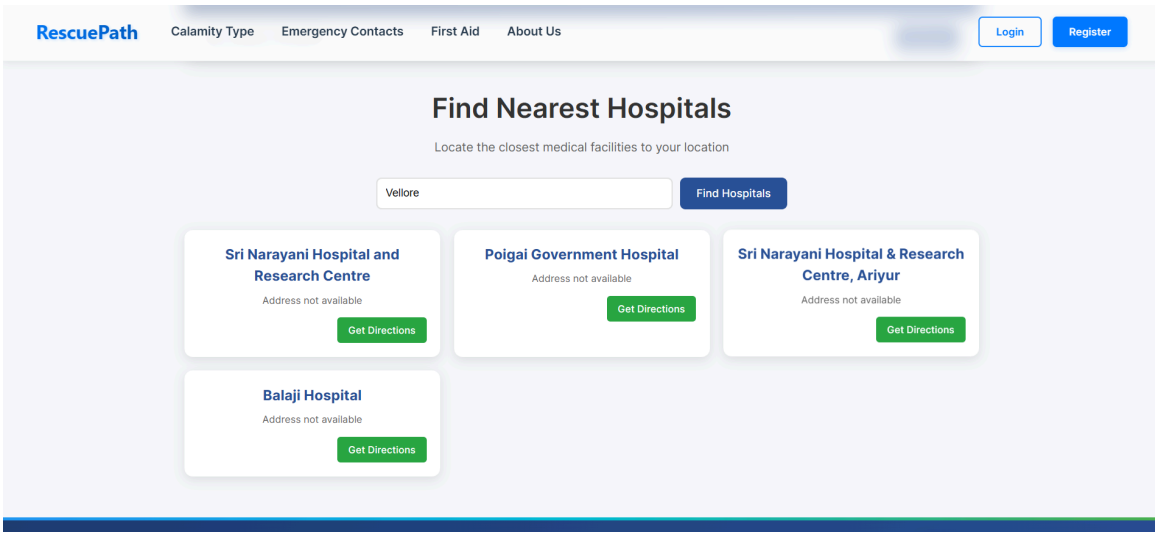

Figure 14: Navigating Hospitals in Vellore

4. When the user clicks on "Get Directions" for a specific hospital (e.g., Sri Narayani Hospital), a new tab or window opens with an OpenStreetMap view. This view displays:
   → A marker for the selected hospital
   → The route from the user's city to the hospital
   → Turn-by-turn directions

→ Estimated distance and travel time

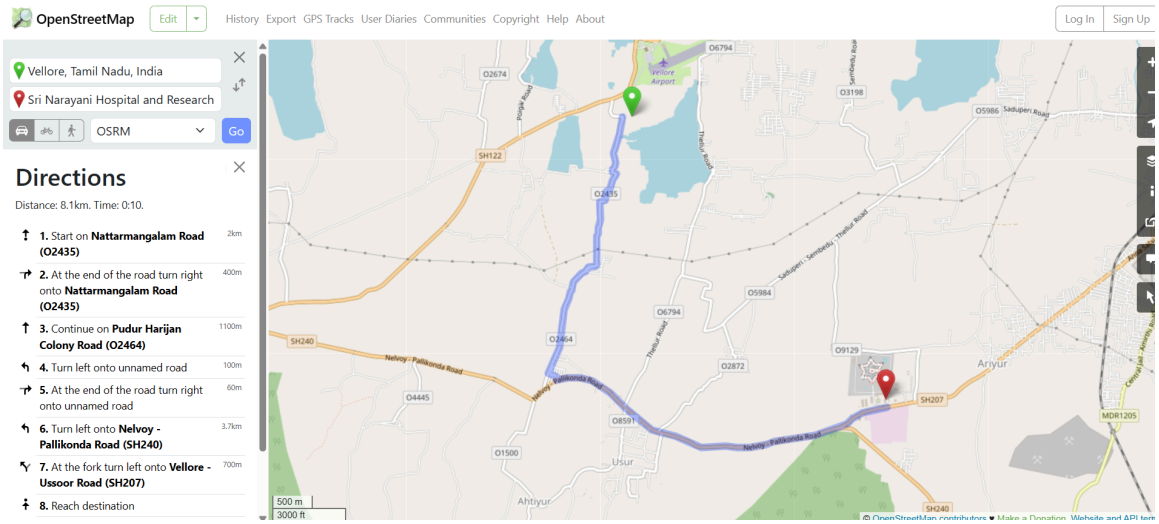

Figure 15: Directions to the chosen hospital

5. This real-time integration enables users, especially those unfamiliar with local geography, to make quick and informed decisions during emergencies.

This feature is especially valuable during disasters such as floods, earthquakes, or fires, when nearby hospitals may be the first point of safety or assistance. Its responsive, map-based visualization makes the guidance both actionable and easy to understand.

7.3.2 Earthquake Safe Zone Finder

1. The user visits the RescuePath homepage, where they see dropdown options for "Calamity Type" and a search bar for entering their current location.
2. The user selects "Earthquake" as the disaster type.
3. In the search field, they enter a location—e.g., “RK Puram New Delhi”.

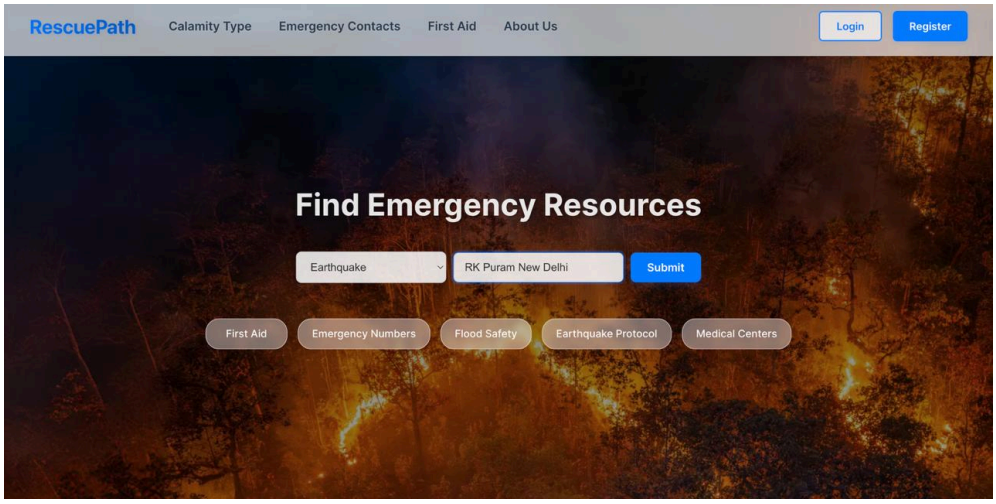

Figure 16: Safe Zone Finder

4. On clicking the “Submit” button:
   - The application queries Nominatim API (nominatim.openstreetmap.org) to geocode the address and get exact coordinates.
   - These coordinates are then passed to the Overpass API to find nearby open areas like parks, stadiums, and grounds.
   - OSRM (Open Source Routing Machine) is used to calculate and generate a route from the user's location to the identified safe open area.

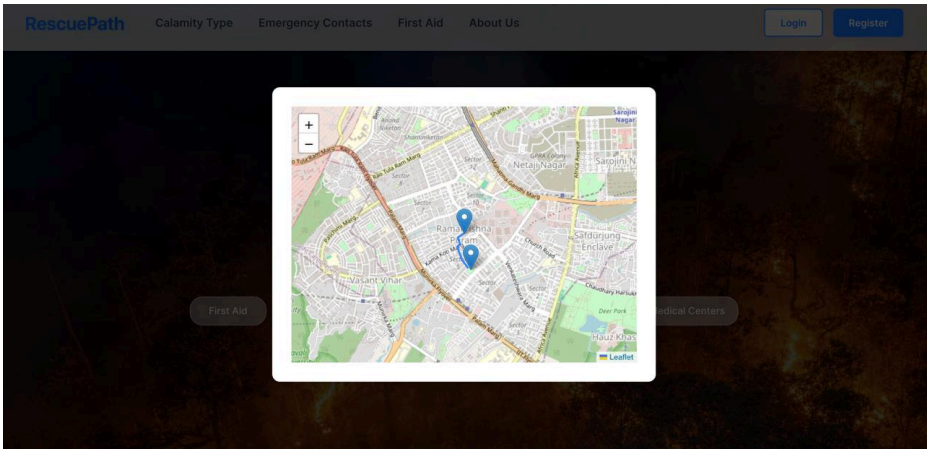

Figure 17: Map displaying the safe path

5. The application displays a map (using Leaflet) marking both the user's input location and the nearest open safe zone.

6. A clear path is drawn on the map with step-by-step walking directions, guiding the user safely to the open ground.

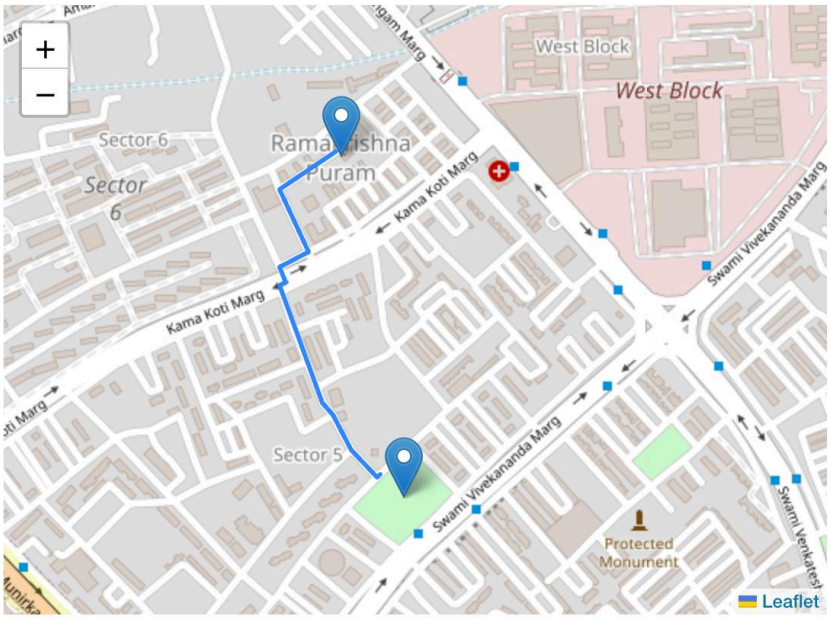

Figure 18: Directions to the closest open area

Chapter - 7

7. RESULTS AND DISCUSSIONS

The RescuePath project has been developed with the aim of providing users with quick and easy access to essential disaster management resources, including real-time weather alerts, hospital locations, and first aid instructions. The development of the application utilized React for frontend development, integrating external APIs such as OpenWeatherMap and OpenStreetMap to provide dynamic, real-time data. The primary objective of the project was to ensure a seamless user experience, enabling fast access to emergency resources during critical times.

7.1 KEY RESULTS

1. Real-Time Emergency Support:

Users can quickly access disaster-specific resources, such as earthquake protocols or flood safety guidelines, by simply selecting a disaster type and entering their location. The app integrates Nominatim and Overpass APIs to identify safe open zones (e.g., parks and stadiums) and uses OSRM for routing in earthquake scenarios.

2. Accurate Weather and Hospital Data:

The application pulls live weather data from the OpenWeatherMap API, including temperature, wind speed, and humidity, offering timely situational awareness. Additionally, OpenStreetMap-based queries return nearby hospitals, complete with directions using map visualizations.

3. Seamless User Experience:

RescuePath features a clean and intuitive interface built with React. The routing is handled client-side using React Router, ensuring fast navigation without full page reloads. Features like emergency contact cards and first-aid tiles are visually organized for quick access.

4. Interactive Demonstrations:

Use-case flows, such as typing “Vellore” to view nearby hospitals or selecting “Earthquake” and “RK Puram” to see directions to open grounds, showcase the platform’s functional integration with third-party geospatial services.

5. Mobile and Cross-Browser Compatibility:

The responsive layout ensures usability across mobile, tablet, and desktop environments. Performance audits using Lighthouse yielded favorable scores in accessibility and mobile responsiveness.

Overall, the system provides a low-latency, high-utility solution for urban and semi-urban populations during disaster events. The real-time decision-making support makes RescuePath a practical tool in emergency preparedness and response.

7.2 COST ANALYSIS:

1. Development Costs:

    - The project utilized open-source tools like React, Jest, Cypress, and OpenStreetMap, significantly reducing initial development costs.

    - The OpenWeatherMap API provides free tier access but comes with limitations in terms of request volume. If the project needs to scale and handle higher request volumes, a paid subscription will be necessary, which could add to operational costs.

    - Hosting and Deployment: The deployment of the project on platforms like Netlify or Heroku provides free-tier options, but scaling the application with higher traffic volumes would incur hosting costs. This would typically involve a paid plan, which may range from $10 to $100/month based on usage.

2. Operational Costs:

    - As the application scales, the operational costs would depend primarily on the API usage (for OpenWeatherMap and OpenStreetMap) and server maintenance. A paid subscription for OpenWeatherMap API could cost between $40 to $100/month for moderate usage.

    - If the application supports a large user base, hosting costs would also rise. Cloud hosting services like AWS, Google Cloud, or Microsoft Azure may be more suitable for scalability but come at a higher cost.

3. Maintenance and Updates:

    - Maintenance costs would involve periodic updates for bug fixes, compatibility with newer versions of libraries, and optimizations based on user feedback.

    - Since the project uses modern technologies, the maintenance effort is likely to be manageable with periodic updates to dependencies and performance improvements.

4. Additional Costs:

- The app could integrate more APIs, such as location-based services or emergency alerts from government bodies. Some of these APIs may require premium access, which could add extra costs.

- Design updates, new features (like multilingual support or advanced analytics), and user training for stakeholders could further increase costs.

Chapter - 8

8. CONCLUSION AND FUTURE WORKS

8.1 CONCLUSION

The RescuePath web application addresses a critical need for a reliable, accessible, and real-time disaster management support system, particularly in regions where emergency preparedness is often reactive rather than proactive. Designed as a responsive, user-centric platform, RescuePath provides individuals with quick access to life-saving information during natural calamities. By integrating data from multiple trusted APIs—such as OpenWeatherMap, OpenStreetMap, Overpass, OSRM, and Nominatim—the application enables users to locate hospitals, receive real-time weather alerts, and identify nearby safe zones, all through a seamless and intuitive interface.

A major highlight of the system is its ability to support decision-making during critical situations. For instance, users can search for medical facilities in their vicinity during emergencies by simply typing the name of their city. They are instantly provided with a list of hospitals and an option to generate directions to reach them safely. Similarly, in the case of an earthquake, users can input their location and view open spaces like parks or stadiums that are safe to evacuate to, supported by a navigable map route. These features directly align with national disaster guidelines, making the tool practically useful and policy-compliant.

RescuePath's frontend is developed using React.js, with dynamic routing handled by React Router. The backend services are built with Node.js and Express, while data fetching is managed using Axios. The app is fully responsive and performs reliably across mobile, tablet, and desktop devices. Lightweight components and modular design ensure a consistent experience and easy future maintenance.

One of the strongest aspects of this project is its cost-efficiency. No commercial tools or paid APIs were used during the development phase. Instead, the project relied entirely on free-tier services and open-source technologies, making it a scalable, low-barrier solution for community-level or institutional deployment.

In summary, RescuePath has successfully delivered a Minimum Viable Product (MVP) that is functional, visually clear, and practical in real-world disaster scenarios. Its ability to deliver contextual guidance in real time makes it a valuable addition to digital disaster response ecosystems.

8.2 FUTURE WORKS:

While the current system meets the essential goals of disaster information delivery and emergency navigation, several enhancements could be implemented to extend its capabilities and improve user engagement. These include:

1. Expansion of Supported Calamities
   At present, the platform focuses primarily on floods and earthquakes. In future iterations, additional calamities such as wildfires, landslides, droughts, cyclones, and even pandemics could be included. Each disaster type would be mapped to a unique set of emergency protocols, government helplines, safe zones, and educational content.

2. Intelligent Geofencing and Alerts
   The platform could incorporate real-time geofencing to monitor user movement and push alerts when a user is entering or nearing a high-risk area. This would be especially useful for situations like flash floods, chemical leaks, or forest fires. Users could also be given early warnings and evacuation routes based on live data.

3. Integration with Local Authority Networks
   RescuePath could be linked with municipal and state-level emergency response systems. This would allow authenticated alerts, advisories, and official communication (e.g., from NDRF or local disaster control rooms) to be broadcasted directly through the app. Two-way communication could even allow users to report incidents or request aid.

4. Safe Zone and Shelter Routing for All Calamities
   While the earthquake module currently displays open grounds and parks, similar functionality could be expanded to suggest shelters, relief camps, or government-designated assembly points for other disasters. These routes could factor in roadblocks, elevation, or real-time traffic and safety conditions using enhanced map layers.

5. Multilingual Support
   Given India's linguistic diversity, supporting regional languages (Tamil, Hindi, Telugu, Bengali, etc.) would make the platform more inclusive. Language selection could be offered at the beginning of the user journey, with translated content for disaster protocols, video instructions, and contact directories.

6. Offline Access to Emergency Info
   During severe disasters, internet access may be unavailable. RescuePath could allow users to download local emergency guides, hospital lists, and contact numbers offline for quick access during network failures. PWA (Progressive Web App) capabilities and local storage could facilitate this.

7. Community-Contributed Information
   RescuePath could implement a feature where verified users can contribute updates—like reporting flooded roads, sharing local shelter availability, or flagging medical supply shortages. A moderation layer would be required to ensure reliability and prevent misuse.

8. AI and Predictive Analytics
   Leveraging historical weather and disaster data, the system could adopt basic machine learning models to forecast risk zones, expected rainfall or flooding severity, and spread of disease outbreaks. These insights could be used to generate early preparedness alerts and help authorities allocate resources more effectively.

9. Accessibility Features
   Enhancements such as voice navigation, font size adjustment, color contrast modes, and screen reader compatibility would make RescuePath usable by people with disabilities—especially the elderly, visually impaired, or those with cognitive challenges.

9. REFERENCE

1. Moussa, L., & Mohan, M. (2024). *An analysis of geospatial technologies for risk and natural disaster management*. *ResearchGate*. Retrieved from https://www.researchgate.net/publication/259640124

2. Kuo, Y.-H., Leung, J. M.Y., Meng, H. M., & Tsoi, K. K.F. (2016). *A real-time decision support tool for disaster response: A mathematical programming approach*. *IEEE BigData Congress*, 639–642. Retrieved from https://www.researchgate.net/publication/302209363

3. Sumon, M. F. I., Khan, M. A., & Rahman, A. (2023). *Machine learning for real-time disaster response and recovery in the U.S. International Journal of Machine Learning Research in Cybersecurity and Artificial Intelligence*, 14(1). Retrieved from https://ijmlrcai.com/index.php/Journal/index

4. Chen, S.-C., & Alonso, S. L. M. (n.d.). *Multimodal data analytics and integration for emergency response and disaster management*. *Smart & Connected Communities Virtual Organization*. Retrieved from https://sccvo.org/projects/multimodal-data-analytics-and-integration-emergency-response-and-disaster-management

5. *Leveraging disruptive technologies for faster and more efficient disaster response management*. *Sustainability*, 16(23), 10730. Retrieved from https://www.mdpi.com/2071-1050/16/23/10730

6. *A comprehensive review of geospatial technology applications in earthquake preparedness, emergency management, and damage assessment*. *MDPI Remote Sensing*, 15(7), 1939. Retrieved from https://www.mdpi.com/2072-4292/15/7/1939

7. Kuo, Y.-H., Leung, J. M.Y., Meng, H. M., & Tsoi, K. K.F. (2016). *A real-time decision support tool for disaster response: A mathematical programming approach*. *IEEE BigData Congress*, 639–642. Retrieved from https://www.researchgate.net/publication/302209363

8. Chen, S.-C., & Alonso, S. L. M. (n.d.). *Multimodal data analytics and integration*

for emergency response and disaster management. *Smart & Connected Communities Virtual Organization*. Retrieved from https://sccvo.org/projects/multimodal-data-analytics-and-integration-emergency-response-and-disaster-management

9. *Leveraging disruptive technologies for faster and more efficient disaster response management*. *Sustainability*, 16(23), 10730. Retrieved from https://www.mdpi.com/2071-1050/16/23/10730

10. *A comprehensive review of geospatial technology applications in earthquake preparedness, emergency management, and damage assessment*. *MDPI Remote Sensing*, 15(7), 1939. Retrieved from https://www.mdpi.com/2072-4292/15/7/1939

11. Shafapourtehrany, M., Batur, M., Shabani, F., Pradhan, B., Kalantar, B., & Özener, H. (2023). A comprehensive review of geospatial technology applications in earthquake preparedness, emergency management, and damage assessment. *Remote Sensing, 15*(7), 1939. Retrieved from https://www.mdpi.com/2072-4292/15/7/1939

12. Calle Müller, C., Lagos, L., & Elzomor, M. (2024). Leveraging disruptive technologies for faster and more efficient disaster response management. *Sustainability, 16*(23), 10730. Retrieved from https://www.mdpi.com/2071-1050/16/23/10730

13. Sumon, M. F. I., Khan, M. A., & Rahman, A. (2023). Machine learning for real-time disaster response and recovery in the U.S. *International Journal of Machine Learning Research in Cybersecurity and Artificial Intelligence, 14*(1). Retrieved from https://ijmlrcai.com/index.php/Journal/index

14. Kuo, Y.-H., Leung, J. M. Y., Meng, H. M., & Tsoi, K. K. F. (2016). A real-time decision support tool for disaster response: A mathematical programming approach. *IEEE BigData Congress*, 639–642. Retrieved from https://www.researchgate.net/publication/302209363

15. Chen, S.-C., & Alonso, S. L. M. (n.d.). Multimodal data analytics and integration for emergency response and disaster management. *Smart & Connected Communities Virtual Organization*. Retrieved from https://sccvo.org/projects/multimodal-data-analytics-and-integration-emergency-response-and-disaster-management

16. Shafapourtehrany, M., Batur, M., Shabani, F., Pradhan, B., Kalantar, B., & Özener, H. (2023). A comprehensive review of geospatial technology applications

in earthquake preparedness, emergency management, and damage assessment. *Remote Sensing, 15*(7), 1939. Retrieved from https://www.mdpi.com/2072-4292/15/7/1939

17. Calle Müller, C., Lagos, L., & Elzomor, M. (2024). Leveraging disruptive technologies for faster and more efficient disaster response management. *Sustainability, 16*(23), 10730. Retrieved from https://www.mdpi.com/2071-1050/16/23/10730

18. Sumon, M. F. I., Khan, M. A., & Rahman, A. (2023). Machine learning for real-time disaster response and recovery in the U.S. *International Journal of Machine Learning Research in Cybersecurity and Artificial Intelligence, 14*(1). Retrieved from https://ijmlrcai.com/index.php/Journal/index

19. Kuo, Y.-H., Leung, J. M. Y., Meng, H. M., & Tsoi, K. K. F. (2016). A real-time decision support tool for disaster response: A mathematical programming approach. *IEEE BigData Congress*, 639–642. Retrieved from https://www.researchgate.net/publication/302209363

20. Chen, S.-C., & Alonso, S. L. M. (n.d.). Multimodal data analytics and integration for emergency response and disaster management. *Smart & Connected Communities Virtual Organization*. Retrieved from https://sccvo.org/projects/multimodal-data-analytics-and-integration-emergency-response-and-disaster-management

21. Shafapourtehrany, M., Batur, M., Shabani, F., Pradhan, B., Kalantar, B., & Özener, H. (2023). A comprehensive review of geospatial technology applications in earthquake preparedness, emergency management, and damage assessment. *Remote Sensing, 15*(7), 1939. Retrieved from https://www.mdpi.com/2072-4292/15/7/1939

22. Calle Müller, C., Lagos, L., & Elzomor, M. (2024). Leveraging disruptive technologies for faster and more efficient disaster response management. *Sustainability, 16*(23), 10730. Retrieved from https://www.mdpi.com/2071-1050/16/23/10730

23. Sumon, M. F. I., Khan, M. A., & Rahman, A. (2023). Machine learning for real-time disaster response and recovery in the U.S. *International Journal of Machine Learning Research in Cybersecurity and Artificial Intelligence, 14*(1). Retrieved from https://ijmlrcai.com/index.php/Journal/index

24. Kuo, Y.-H., Leung, J. M. Y., Meng, H. M., & Tsoi, K. K. F. (2016). A real-time decision support tool for disaster response: A mathematical programming approach. *IEEE BigData Congress*, 639–642. Retrieved from https://www.researchgate.net/publication/302209363

25. Chen, S.-C., & Alonso, S. L. M. (n.d.). Multimodal data analytics and integration for emergency response and disaster management. *Smart & Connected Communities Virtual Organization*. Retrieved from https://sccvo.org/projects/multimodal-data-analytics-and-integration-emergency-response-and-disaster-management

APPENDIX A - SAMPLE CODE

App.jsx

```
import { BrowserRouter as Router, Routes, Route, Link, useNavigate } from
'react-router-dom';

import './App.css';

import { useState } from 'react';


import wildfire from './assets/wildfire.avif';

import WeatherAlerts from './components/WeatherAlerts';

import NearestHospital from './components/NearestHospital';

import EscapeRouteMap from './components/EscapeRouteMap';

import Footer from './components/Footer';

import ScrollToTop from './components/ScrollToTop';

import Modal from './components/Modal'; // 🆕 Import the modal


import FirstAid from './pages/FirstAid';

import Emergency from './pages/Emergency';

import CalamityTypes from './pages/CalamityTypes';

import AboutUs from './pages/AboutUs';


function App() {

  return (

    <Router>

      <ScrollToTop />
```

```
<div className="app">
  <nav className="topbar">
    <div className="left-section">
      <Link to="/" className="logo-text">RescuePath</Link>
      <div className="nav-links">
        <Link to="/calamity-types" className="nav-link">Calamity Type</Link>
        <Link to="/emergency" className="nav-link">Emergency Contacts</Link>
        <Link to="/first-aid" className="nav-link">First Aid</Link>
        <Link to="/about" className="nav-link">About Us</Link>
      </div>
    </div>
    <div className="auth-buttons">
      <button>Login</button>
      <button>Register</button>
    </div>
  </nav>

  <Routes>
    <Route path="/" element={<HomeContent />} />
    <Route path="/first-aid" element={<FirstAid />} />
    <Route path="/emergency" element={<Emergency />} />
    <Route path="/calamity-types" element={<CalamityTypes />} />
    <Route path="/about" element={<AboutUs />} />
  </Routes>
```

```
      <Footer />
    </div>
  </Router>
 );
}

function HomeContent() {
 const navigate = useNavigate();
 const [disaster, setDisaster] = useState('');
 const [location, setLocation] = useState('');
 const [routeData, setRouteData] = useState(null);
 const [showModal, setShowModal] = useState(false); // 🆕

 const handleTagClick = (path, section) => {
  if (path === '/first-aid' && section) {
   navigate(path, { state: { scrollTo: section } });
  } else if (section === 'hospitals') {
   const element = document.getElementById('hospital-section');
   if (element) {
    element.scrollIntoView({ behavior: 'smooth' });
   }
  } else {
   navigate(path);
  }
 };
```

```
  const handleSubmit = async () => {
    if (!location || !disaster) {
      alert("Please enter both location and disaster type.");
      return;
    }

    try {
      const res = await
fetch(`http://localhost:8000/get-escape-route?location=${location}&calamity=${disaster
}`);
      if (!res.ok) throw new Error("Failed to fetch route");

      const data = await res.json();
      setRouteData(data);  // Save route data
      setShowModal(true);  // Show modal 🆕
    } catch (error) {
      console.error("Error fetching route:", error);
      alert("Could not fetch escape route. Try again later.");
    }
  };

  return (
    <>
      <div className="search-container" style={{ backgroundImage: `url(${wildfire})`
}}>
```

```
<h1 className="search-heading">Find Emergency Resources</h1>
<div className="search-wrapper">
  <select
    value={disaster}
    onChange={(e) => setDisaster(e.target.value)}
    className="disaster-dropdown"
  >
    <option value="">Select Disaster Type</option>
    <option value="earthquake">Earthquake</option>
    <option value="floods">Floods</option>
    <option value="tsunami">Tsunami</option>
    <option value="wildfire">Wildfire</option>
  </select>
  <input
    type="text"
    value={location}
    onChange={(e) => setLocation(e.target.value)}
    placeholder="Enter location..."
    className="location-search"
  />
  <button className="submit-button" onClick={handleSubmit}>Submit</button>
</div>

<div className="popular-searches">
  <span className="search-tag" onClick={() => handleTagClick('/first-aid')}>First
```

```
Aid</span>

        <span className="search-tag" onClick={() =>
handleTagClick('/emergency')}>Emergency Numbers</span>

        <span className="search-tag" onClick={() =>
handleTagClick('/calamity-types')}>Flood Safety</span>

        <span className="search-tag" onClick={() =>
handleTagClick('/calamity-types')}>Earthquake Protocol</span>

        <span className="search-tag" onClick={() => handleTagClick(null,
'hospitals')}>Medical Centers</span>

      </div>

    </div>


    {/* 🔲 Modal for Escape Route */}

    <Modal isOpen={showModal} onClose={() => setShowModal(false)}>

      {routeData && <EscapeRouteMap routeData={routeData} />}

    </Modal>


    <WeatherAlerts />

    <div id="hospital-section">

      <NearestHospital />

    </div>

  </>

 );

}


export default App;
```

Main.py

```
from fastapi import FastAPI, HTTPException, Query
from fastapi.middleware.cors import CORSMiddleware
from utils.geocode import geocode_location
from utils.overpass_query import find_nearest_open_space
from utils.routing import get_route

app = FastAPI(title="RescuePath Escape Route API")

# Add CORS middleware
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # In production, set this to your frontend's URL
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

@app.get("/get-escape-route")
async def escape_route(location: str = Query(...), calamity: str = Query(...)):
    print(f"[API CALL] Received request - Location: {location}, Calamity: {calamity}")

    if calamity.lower() != "earthquake":
```

```
    raise HTTPException(400, "Only 'earthquake' is supported for now")

try:
    origin = await geocode_location(location)
    print(f"[GEOCODE] Origin coordinates: {origin}")
except Exception as e:
    print(f"[ERROR] Geocoding failed: {e}")
    raise HTTPException(400, f"Geocoding failed: {e}")

try:
    dest = await find_nearest_open_space(origin["lat"], origin["lon"])
    print(f"[DESTINATION] Nearest open space: {dest}")
except Exception as e:
    print(f"[ERROR] Finding nearest open space failed: {e}")
    raise HTTPException(404, f"No safe zone found: {e}")

try:
    route = await get_route(origin, dest)
    print(f"[ROUTE] Route data length: {len(route)} steps")
except Exception as e:
    print(f"[ERROR] Route calculation failed: {e}")
    raise HTTPException(500, f"Routing failed: {e}")

return {
    "origin": origin,
```

```
    "destination": {
        "name": dest["name"],
        "lat": dest["lat"],
        "lon": dest["lon"]
    },
    "route": route
}
```

GITHUB LINK: *https://github.com/Gaurrav7Yadav/Rescue_Path*