WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

FLARE: Smart Emergency Response and Public Safety Platform

Haritha M [#1], Sabeetha M [#2], Ajay A [#3], Akshara Varshini S [#4], Anjana V J [#5], Dinesh R [#6], Divyaprakash A J [#7]

[#1] Mentor, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
[#2] Coordinator, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
[#3] Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
[#4] Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
[#5] Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
[#6] Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
[#7] Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.

ABSTRACT

Personal safety, particularly for women, remains a serious concern in public and private spaces. In an emergency, a victim rarely has the time to make several phone calls, explain a location, or open multiple applications. FLARE: Smart Emergency Response and Public Safety Platform is a web-based system designed to reduce this delay by allowing a user to raise an alert with a single action and to share her live location with trusted people immediately.

The platform is built with HTML5, CSS3 and vanilla JavaScript on the frontend and Node.js with Express.js on the backend. User, contact and alert information is stored in MongoDB through Mongoose, and authentication is handled with JSON Web Tokens and bcrypt password hashing. When the SOS button is pressed, or when the voice command "help" or "emergency" is recognised, the application reads the device location through the browser Geolocation API, captures a still photograph through the device camera as evidence, and sends the location, a map link and the photograph to every saved emergency contact by e-mail (Nodemailer) and by SMS (Twilio).

FLARE also provides an interactive Leaflet and OpenStreetMap map with live tracking, nearby police station and hospital markers with route guidance, a community alert layer that shows active SOS events of other users, and a fake incoming call screen that helps a user leave an uncomfortable situation. The system uses only free and open technologies. Further field testing is required to measure alert delivery time and reliability on different devices and networks.

INTRODUCTION

Crimes and harassment against women in public transport, workplaces, educational campuses and isolated streets continue to be reported widely. Although most people carry a smartphone, a conventional emergency call requires the victim to unlock the phone, dial, and describe where she is.

These steps are difficult under stress and may even alert the attacker. A safety system should therefore work with minimum interaction, share the exact location automatically, and notify people who can respond quickly.

Mobile and web technologies make this possible. Modern browsers expose the Geolocation API for position, the MediaDevices API for the camera, and the Web Speech API for voice recognition. Combined with a server that can send e-mail and SMS, these interfaces can turn an ordinary device into an emergency beacon. Open map services such as OpenStreetMap and Leaflet further allow location information to be displayed without paid map licences.

FLARE (Smart Emergency Response and Public Safety Platform) is designed on these principles. The main objective of the project is to provide a simple, fast and free safety platform in which a registered user can store emergency contacts, trigger an alert by button or voice, share live location and photographic evidence, find nearby help, and make the surrounding community aware of an active emergency. The modular client-server architecture also allows additional safety features to be added without changing the existing components.

LITERATURE SURVEY

Women safety applications have been studied extensively as a way of using ubiquitous mobile devices for personal security. Most existing systems follow a common pattern: the user registers a few trusted contacts, and a panic action sends a message with the current location to those contacts. The effectiveness of such systems depends on how quickly the alert can be raised and how reliably it is delivered.

Location-based services rely on the Global Positioning System and network positioning to determine where a device is. The W3C Geolocation API standardises access to this information in web browsers, which makes browser-based safety applications possible without installing native software, although it requires a secure origin such as HTTPS or localhost.

Several existing applications depend on dedicated hardware such as wearable panic buttons, or on paid services for maps and messaging. These solutions can be expensive and may not be accessible to all users. Open-source mapping through OpenStreetMap, the Leaflet library, and the Open Source Routing Machine (OSRM) provide free alternatives for visualising location and calculating routes.

Token-based authentication using JSON Web Tokens is widely used to protect REST APIs without server-side sessions, while adaptive hashing schemes such as bcrypt protect stored passwords against brute-force attacks. Document databases such as MongoDB store flexible records such as alerts and contacts, and time-to-live indexes allow temporary data to expire automatically.

Community-based reporting is another important idea in public safety. When users can mark unsafe places and see active alerts nearby, the system benefits from collective awareness. FLARE combines these earlier ideas, namely one-touch alerting, multi-channel notification, evidence capture, voice activation, open maps, and community awareness, in a single free web platform.

METHODOLOGY

Overview of Proposed Method

FLARE is a full-stack web application that connects a browser-based frontend with a Node.js and Express.js backend through REST APIs. The frontend provides the user interface for authentication, emergency contacts, the SOS trigger and the map. The backend validates each request, reads or writes MongoDB collections, and sends the e-mail and SMS notifications.

The high-level workflow is:

User → Browser Interface → REST API → Express.js Backend → MongoDB / Nodemailer / Twilio → Response→ Dashboard Display

During an emergency the workflow is: SOS button or voice command → Geolocation → Camera evidence capture → POST /api/sos → Alert stored → E-mail and SMS to emergency contacts → Community alert shown on other users' maps.

The development process consists of requirement analysis, architecture design, database design, backend implementation, frontend implementation, integration of maps and notification services, testing, and evaluation. Each module is tested independently before end-to-end integration.

Dataset Description

FLARE does not use an image dataset or a machine-learning training dataset. Its data is operational data created by users while the application is running. The data is stored in MongoDB in four collections, summarised in Table 1.

Table 1. Data collections used by FLARE

Collection	Main fields
User	name, email (unique), phone, hashed password, createdAt
Contact	user reference, name, phone, email, relation, createdAt
Alert	user reference, latitude, longitude, active flag, base64 evidence image, createdAt (auto-expires after 24 hours)
DangerZone	latitude, longitude, description, reportedBy, createdAt

Data Preprocessing

Data handling in FLARE concerns validating, normalising and securing the information that flows between the browser and the server. During registration the password is never stored directly; it is salted and hashed with bcrypt before being saved. After login the server returns a JSON Web Token valid for five hours, and the browser sends it in the x-auth-token header of each protected request. A middleware verifies the token and rejects requests with a missing or invalid token.

Location data is obtained from the Geolocation API as latitude and longitude values and converted into OpenStreetMap and Google Maps links for e-mail and SMS messages. Phone numbers entered by the user in different formats are normalised to the international E.164 format: spaces, dashes and brackets are removed, a leading zero is dropped, and a ten-digit number is prefixed with the +91 country code. The photograph captured by the camera is drawn on a canvas and encoded as a base64 JPEG at 80% quality before upload, and it is attached to the e-mail as a JPEG file.

Errors such as denied location permission, unavailable camera, missing contacts, invalid credentials, or failed message delivery are handled and reported to the user through on-screen notifications.

Model Architecture

FLARE follows a three-tier client-server architecture consisting of a presentation layer, an application layer and a data layer, supported by external services. Table 2 lists the technologies used.

Table 2. Technology stack of FLARE

Layer	**Technology**
Frontend	HTML5, CSS3 (glassmorphism interface), vanilla JavaScript
Backend	Node.js, Express.js, CORS, dotenv
Database	MongoDB with Mongoose
Security	JSON Web Token, bcryptjs
Maps and routing	Leaflet.js, OpenStreetMap, Leaflet Routing Machine (OSRM)
Notifications	Nodemailer (Gmail SMTP), Twilio SMS
Browser APIs	Geolocation, MediaDevices, Web Speech

The presentation layer contains the login and registration screens, the home dashboard with the SOS button, and the contacts screen. The application layer exposes REST endpoints for authentication, contacts, SOS, alerts and danger zones, listed in Table 3. The data layer stores users, contacts, alerts and danger zones.

Table 3. REST API endpoints

Endpoint	**Method**	**Purpose**
/api/auth/register	POST	Create account, return token
/api/auth/login	POST	Authenticate user
/api/auth/me	GET	Get logged-in profile
/api/contacts	GET, POST	List and add contacts
/api/contacts/:id	PUT, DELET E	Update and remove contact
/api/sos	POST	Trigger SOS, send e-mail and SMS
/api/alerts	GET	List active community alerts
/api/alerts/resolve	POST	Close user's active alert
/api/alerts/evidence	POST	Attach photo to active alert
/api/danger-zones	GET, POST	List and report unsafe areas

Each protected route passes through the authentication middleware. The contact routes also check that the contact belongs to the requesting user before it is changed or deleted. The modular route and model structure makes the system easy to maintain and extend.

Development and Testing Process

FLARE is a software platform rather than a machine-learning system, so its development emphasises implementation, integration, functional testing and debugging instead of model training. The backend and frontend were developed as separate modules and integrated through REST APIs.

First, the Node.js server was configured with Express.js, the MongoDB connection and the environment variables for the server port, database address, token secret, e-mail credentials and Twilio credentials. The authentication and contact routes were created and checked with sample requests. The SOS route was then added to find the user's contacts, store an alert, and send notifications by e-mail and SMS. For Gmail the server uses an application password rather than the account password.

Next, the frontend was developed. The authentication forms, dashboard, contact management and map were implemented and connected to the backend API. Because the Geolocation and camera APIs are available only on secure origins, the frontend is served through a local HTTP server or VS Code Live Server rather than opened directly as a file.

Functional testing follows the user workflow: register an account, log in, add a contact with a personal secondary e-mail address, grant location and camera permission, press SOS, and check that the e-mail with the location link and photograph is received. The voice trigger, fake call, nearby help routing and community alert marker are tested in the same way. Failure cases such as no contacts, denied permissions, and unconfigured e-mail or SMS service are also tested, so that the user receives a clear message.

Evaluation Metrics

The evaluation of FLARE measures how quickly and reliably an emergency can be reported and how well each module functions. The following metrics are used.

1. **SOS Response Time:** The time between pressing the SOS button and receiving the confirmation from the server. It includes location acquisition, evidence capture and notification sending. The application waits up to 8 seconds for the camera so that the alert is not delayed indefinitely.
2. **Alert Delivery Success Rate:** The percentage of e-mails and SMS messages that are delivered to emergency contacts out of the total attempted. The server counts successful and failed deliveries and reports them to the user.
3. **Location Accuracy:** How closely the reported latitude and longitude match the real position of the device. It depends on the device GPS and on network conditions, and high-accuracy mode is requested.
4. **Evidence Capture Success:** Whether the camera permission is granted and the photograph is captured and attached to the alert.
5. **Functional Reliability:** Whether authentication, contacts, voice trigger, fake call, routing and community alerts behave correctly, including error handling.

Implementation Process:

The implementation consists of a frontend of three JavaScript modules and a backend of five route modules. The api.js module wraps all calls to the REST API; the map.js module handles the map, location, SOS and voice logic; and the app.js module handles views, authentication, contacts and the fake call and evidence capture functions.

Authentication and Dashboard

A new user creates an account with name, e-mail, phone number and password, and an existing user signs in with e-mail and password. After success the token is stored in the browser and the main view opens with a Home tab containing the emergency trigger and a Contacts tab. Logging out clears the token and switches off any active voice recognition.

SOS Trigger and Alert Notification

When the SOS button is pressed, the application obtains the position, silently captures a photograph, and sends a request to the /api/sos endpoint. The server stores or updates the user's active alert, sends an e-mail with the location, map links and the photo attachment to each contact having an e-mail address, and sends an SMS with a Google Maps link through Twilio to each contact having a phone number. If some deliveries fail, the server responds with a partial-success message that lists the failures. While the alert is active the location is refreshed every 10 seconds, and tracking stops automatically after 3 minutes, which also closes the active alert.

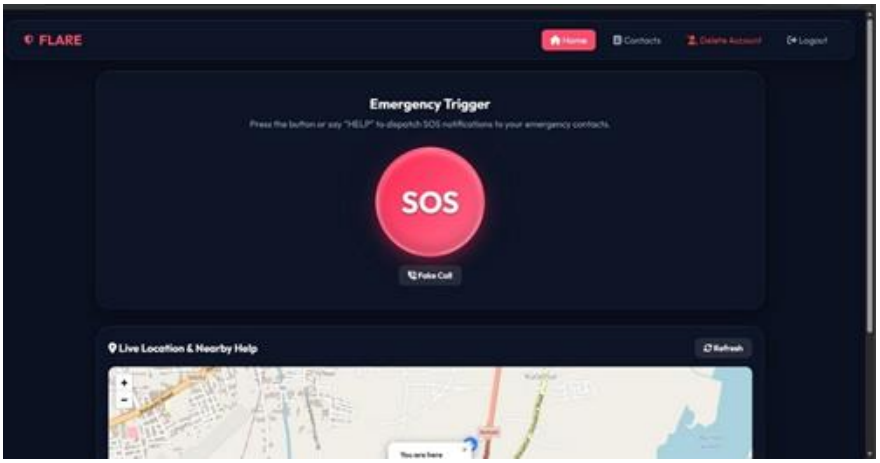


Fig. 1. Home dashboard with the SOS emergency trigger and Fake Call

Voice SOS and Fake Call

Voice SOS is optional. The microphone is switched on only when the user presses the Voice SOS button, and the Web Speech API listens continuously. If the recognised text contains the word "help" or "emergency" the SOS procedure starts automatically. The fake call feature shows a full-screen incoming call with a caller name and plays a ringtone after a delay of three seconds, so that the user can leave an uncomfortable situation naturally.

Live Map, Nearby Help and Safe Route

The map is created with Leaflet and OpenStreetMap tiles. The user is shown as a pulsing marker, and nearby police station and hospital markers are displayed with a Directions button. Choosing it calculates a route with Leaflet Routing Machine and draws the path in green. The nearby help markers are currently generated as demonstration points around the user's location, and integration with a real places data source is planned as future work. Active alerts of other users are fetched every 15 seconds and displayed as red pulsing markers, giving community awareness.

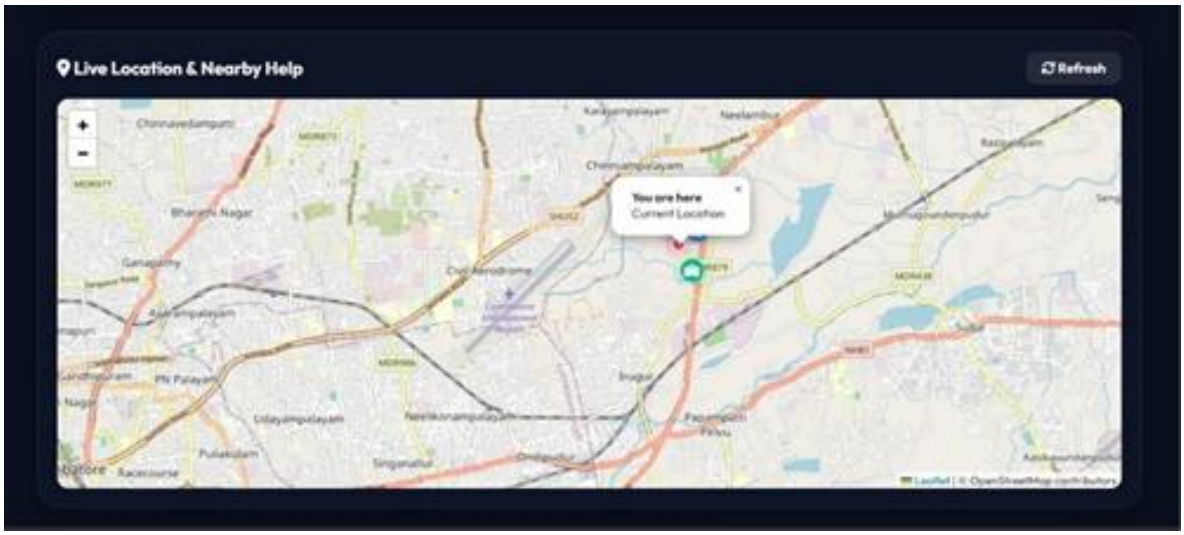


Fig. 2. Live location map with nearby help markers

Emergency Contact Management

The Contacts tab lists the saved contacts as cards showing name, phone, e-mail and relation (Family, Friend, Colleague or Other). New contacts are added through a form and can be removed after a confirmation. The interface also lets the user choose, for each contact, which channels are enabled (WhatsApp, SMS and e-mail), shows the status of the last SMS dispatched, and provides a Delete Account option in the navigation bar. These contacts are the recipients of every SOS alert.

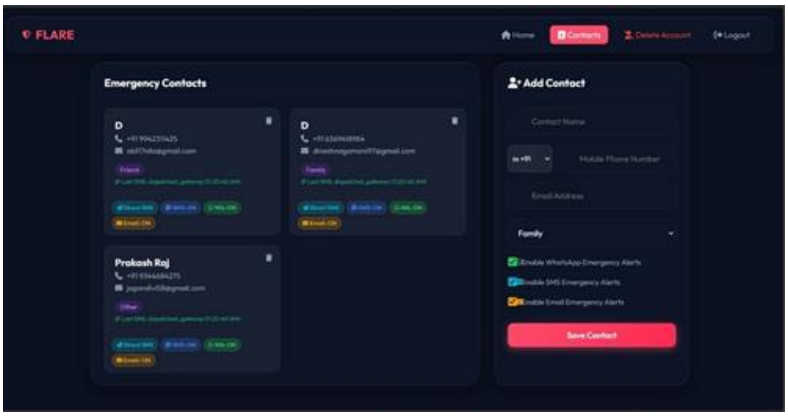


Fig. 3. Emergency contact management

Danger Zone Reporting

The backend includes a DangerZone model and routes by which authenticated users can report an unsafe location with a description, and by which all users can retrieve the reported zones for display on the map. This module forms the basis of crowd-sourced unsafe-area mapping.

Advantages of the Proposed Model

FLARE reduces the time and effort needed to ask for help. A single press of a button or a spoken word triggers a complete alert containing the live location, a map link and a photograph, so contacts receive the information they need to act. Notification through both e-mail and SMS increases the chance that at least one contact sees the alert quickly.

The platform uses free and open technologies, namely Node.js, MongoDB, Leaflet, OpenStreetMap and OSRM, and runs in a normal web browser without installing an application. The additional features of fake call, voice activation, safe route suggestion and community alerts give a more complete safety toolkit than a simple panic button.

Security is considered through hashed passwords, token-based access, per-user data ownership checks, and automatic deletion of alert records after 24 hours. The modular design separates routes, models and interface logic, which makes the system easy to test, maintain and extend with new features.

SOS Response Time

The SOS response time represents the interval between the user pressing the SOS button and the application showing that the emergency contacts have been notified. It consists of the time to obtain the location, the camera capture of at most 8 seconds, and the server time to store the alert and send messages. It can be recorded by repeating the SOS action several times under different network conditions and noting the duration of each attempt. Lower values indicate a faster emergency response.

Alert Delivery Rate

The alert delivery rate shows how many of the e-mail and SMS messages reach the intended contacts. The backend returns, for each SOS, the number of e-mails and SMS messages sent and failed. Measuring this over multiple tests helps to identify problems such as incorrect e-mail credentials, invalid phone numbers or trial-account restrictions of the SMS provider.

Interpretation

FLARE combines a browser interface with a Node.js backend, MongoDB storage, and e-mail and SMS notification. The implemented modules, namely authentication, contact management, SOS alerting with location and photographic evidence, voice trigger, fake call, live map, route guidance and community alerts, together demonstrate the complete emergency workflow. Detailed measurement of response time and delivery reliability under different devices, browsers and networks is required before conclusions about real-world performance can be drawn.

Formula

Alert Delivery Success Rate (%) = (Messages Delivered / Messages Attempted) × 100

For example, if 9 out of 10 messages are delivered, the success rate is 90%. This is an illustrative example, not a measured FLARE result.

Performance Interpretation

The performance of FLARE can be judged by SOS response time, alert delivery success rate, location accuracy, evidence capture success and functional reliability. A short response time and a high delivery rate show that contacts are informed quickly and reliably. Location accuracy depends on the device GPS and the surrounding environment, and camera capture depends on the permission given by the user. Repeated tests in different locations and networks can reveal limitations and guide improvements. Actual measurements must be collected before drawing conclusions about the overall performance of the system.

Table 4. Performance Evaluation of FLARE

S.No.	Evaluation Metric	Purpose	Current Observation
1	SOS Response Time	Measure time from SOS press to confirmation	Requires measurement
2	Alert Delivery Rate	Evaluate e-mail and SMS delivery	Requires repeated testing
3	Location Accuracy	Verify reported position	Requires field testing
4	Evidence Capture	Verify camera photo attached	Implemented; depends on permission
5	Authentication	Verify secure login and token access	Implemented
6	Error Handling	Evaluate behaviour during failures	Messages implemented; needs validation

Result:

The FLARE application provides a single interface through which a user can register, store emergency contacts, trigger an alert, and monitor the live location on a map. When the SOS is activated, the application shows each stage of the process, namely fetching location, capturing evidence and sending alerts, and finally confirms that the emergency contacts have been notified.

The contacts receive an e-mail carrying the name of the user, the latitude and longitude, links to OpenStreetMap and Google Maps, the user's phone number, an emergency ID, buttons for Google Maps and live tracking, and the captured camera evidence, and an SMS carrying a map link. Other logged-in users see the active alert as a red marker on their maps. The user can also start a fake call, use voice activation, and obtain a route to the nearest police station or hospital.

The results show that FLARE offers a unified, low-cost and easy-to-use platform that shortens the time needed to ask for help and supports faster response by trusted contacts.

SOS Alert E-mail:

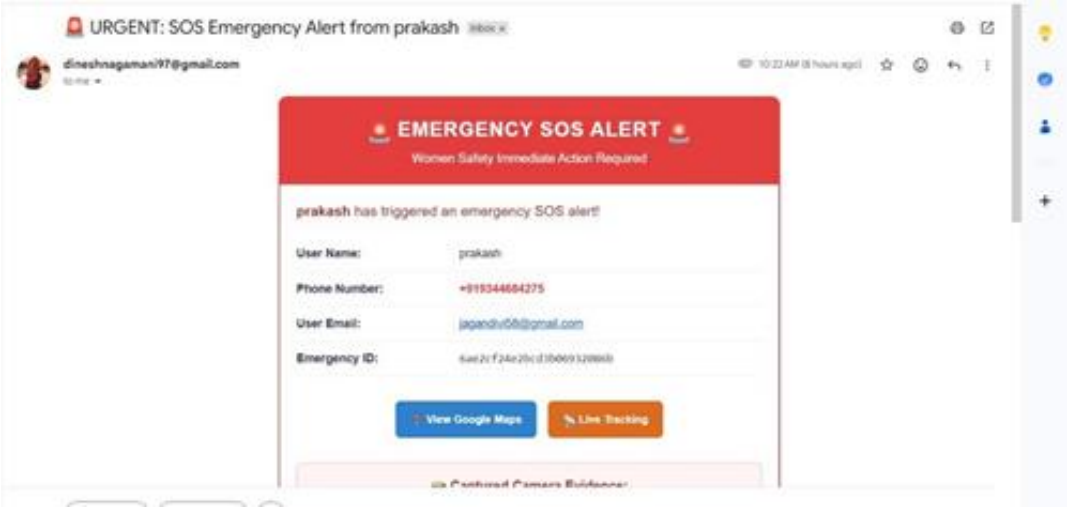


Fig. 4. SOS alert e-mail received by an emergency contact

Conclusion:

This project presents FLARE: Smart Emergency Response and Public Safety Platform, a web-based system designed to help users raise an emergency alert quickly and share their live location with trusted contacts.

The system combines a browser frontend with a Node.js and Express.js backend, MongoDB storage and REST API communication. One-touch and voice-activated SOS, automatic evidence capture, e-mail and SMS notification, an interactive map with safe route suggestion, a fake call, and community alert display are all implemented using free technologies.

Current limitations include the demonstration nature of the nearby help markers, the dependence of e-mail and SMS on external service configuration, the need for a secure (HTTPS or localhost) origin for the camera and location, and the lack of measured performance results. Future enhancements may include integrating the danger zone reporting with the map interface, retrieving real police station and hospital data, a mobile application with background alerting and a hardware or shake-based trigger, audio and video evidence capture, multi-language support, push notifications, and deployment of the backend to a cloud environment with HTTPS.

In conclusion, FLARE aims to improve personal and public safety by combining quick alerting, location sharing, evidence capture and community awareness in one accessible platform. Continued development and systematic testing can evolve it into a dependable emergency response solution.

References

1. Leaflet. *Leaflet: An Open-Source JavaScript Library for Interactive Maps*. Official documentation, leafletjs.com.
2. OpenStreetMap Foundation. *OpenStreetMap*. Open map data and tile service, openstreetmap.org.
3. Luxen, D. and Vetter, C. (2011). *Real-time routing with OpenStreetMap data*. Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems.
4. Node.js Documentation. *Node.js Documentation*. Official documentation for server-side JavaScript development.
5. Express.js Documentation. *Express Web Framework*. Official documentation for building web applications and REST APIs.
6. MongoDB Documentation and Mongoose Documentation. *MongoDB Manual; Mongoose ODM*. Official documentation for document databases and object modelling.
7. Jones, M., Bradley, J. and Sakimura, N. (2015). *JSON Web Token (JWT)*. RFC 7519, Internet Engineering Task Force.
8. Provos, N. and Mazières, D. (1999). *A Future-Adaptable Password Scheme*. Proceedings of the USENIX Annual Technical Conference.

9. Nodemailer Documentation. *Nodemailer: Send Emails from Node.js*. nodemailer.com.

10. Twilio Documentation. *Programmable Messaging (SMS) API*. Official documentation, twilio.com/docs.

11. W3C. *Geolocation API; Media Capture and Streams; Web Speech API*. Web platform specifications for browser location, camera and voice input.

12. Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine.