CHAPTER 1

INTRODUCTION

In today’s fast-moving world, Women Security is an issue of growing concern. We have read about many unfortunate incidents happening with women and the rate is increasing. Women these days are working women and the globalization has made us aware of gender equality. Earlier the women were restricted only to the household chores. With the changing scenario, women are competing with men in all fields. We can see women going to great success levels in all fields, may it be corporate, scientific, education, business or any other field. Safety of women matters a lot whether at home, outside the home or working place. Last few crimes against women especially the case in Delhi was very dread and fearful. Because of such crimes, women safety has become a major topic. It is found through the survey that the reason of safety concern is the lack of gender-friendly environment and improper functional infrastructure such as consumption of alcohol and drugs in open area, lack of adequate lighting, safe public toilets, sidewalks, lack of effective police service, lack of properly working helpline numbers, etc. A huge percentage of women have no faith that police can curb such harassment cases.

With the rise of technology, there has been a growing emphasis on leveraging digital solutions to address safety concerns effectively. The Women's Safety Application represents a proactive response to these challenges, integrating advanced technological features to empower women and enhance their security in various situations. The need for women's safety applications arises from the alarming statistics of harassment, assault, and violence faced by women globally. These incidents not only threaten personal safety but also impact the overall well-being and freedom of individuals. Technology provides a unique opportunity to mitigate these risks by offering instant communication, location tracking, and emergency response capabilities at the touch of a button. The purpose of creating this web application is to provide a safe environment through smart phones, as most people now carry smartphones with them everywhere they go.

1.1 Problem statement

Develop a Women's Safety Application (WSA) that enables users to trigger emergency alerts to the nearest police station and designated contacts. Upon pressing the emergency button within the application, the system will send SMS messages containing the user's current location, name, and phone number to predefined contacts and emergency services. Furthermore, the application will initiate automatic audio recording to gather evidence during critical situations. Additionally, users can directly initiate emergency calls by clicking a link provided within the application interface.

1.2 Objectives

- **Ensure Swift Emergency Response**: Enable users to quickly alert nearby police stations and contacts in case of emergencies.
- **Provide Location-based Alerts**: Include precise location details in SMS alerts sent to emergency contacts and authorities.
- **Automate Evidence Collection**: Initiate automatic audio recording upon triggering an emergency alert to support legal proceedings.
- **Facilitate Direct Emergency Calls**: Allow users to directly call emergency services through a clickable link within the application.

1.3. Organization of Mini-Project Report

- **Ensure Technology Stack**: Detail the use of React for frontend development, MongoDB for data storage, Express for backend API development, Node.js for server-side execution, and Twilio for SMS and call functionalities.
- **Emergency Alert Functionality**: Describe the process flow from user interaction (pressing emergency button) to SMS dispatch to designated contacts and emergency services.
- **Automatic Recording Feature**: Explain the implementation of automatic audio recording triggered during emergency alerts for evidentiary purposes.
- **Direct Emergency Call Link**: Discuss the integration of a clickable link for users to directly call emergency services from within the application.
- **Security and Privacy Considerations**: Address measures taken to ensure user data security and privacy compliance in handling sensitive information such as location and personal details

CHAPTER 2

LITERATURE SURVEY

In this literature review of the Women's Safety Application, extensive research into patents, academic papers, industry documents, and media articles has informed the development rationale and requirements. The increasing prevalence of digital solutions for safety, driven by rising concerns over women's security globally, underscores the critical need for effective and accessible safety applications. This addresses these challenges by integrating advanced features such as emergency alert systems, real-time location tracking, and communication with emergency services and trusted contacts. While technological advancements have improved safety measures, challenges remain, particularly in ensuring reliable and prompt responses during emergencies. This aims to mitigate these issues by providing users with tools for swift and discrete alert triggering, precise location sharing, and automated evidence collection through features like audio recording. Customizable settings and user-centric design further enhance usability, aiming to empower women to navigate their daily lives with enhanced security and confidence. Some of the literature survey are:

1. **A Mobile Application for Women's Safety: WoSApp**

Author: Dhruv Chand, Sunil Nayak, Karthik Bhat, Shivani Parikh, Yuvraj Singh, Amita Ajith Kamath **Year:** 2015

This paper introduces a mobile application known as WoSApp (Women's Safety App) that gives ladies with a reliable thanks to place associate emergency decision to the police. The user will quickly and dis creetly trigger the vocation perform by shaking her phone or by expressly interacting with the applicatio n's program via a straightforward press of a push button on the screen. A message containing the geogra phical location of the user, additionally as contact details of a pre- selected list of emergency contacts, i s instantly sent to the police. This paper describes the applying, its development, and its technical imple mentation

2. **Woman Safety Application – MwithU**

Author: Abhijeet Singh , Vishnu Barodiya**. Year:** 2018

Our application is basically a web based technology so the system requirement is very low for the appli cation. And as the main data exchange will be happening mostly through geolocation API so low netwo rk speed will also work. MwithU’s requirement for functioning is kept to be low as that it can be used in almost all the parts of the world with low end devices and weak internet connections. We at start are ma inly focusing on two types of application

3. Android App for Women Safety

Author: Dr. K Srinivas , Dr. Suwarna Gothane , C. Saisha Krithika , Anshika , T. Susmitha **Year:** 2021

This app is developed by AppSoftIndia. The key features of the app are: the user has to save some detai ls. These details include: Email address and password of the user, Email address and mobile number of t he recipient and a text message. Then, app is loaded as a "widget", so that when the user touches the ap p, it alerts the recipient. Another key feature of app is that it records the voice of surroundings for about 45 seconds and this recorded voice, text message containing location coordinates of the user is sent to th e recipient mobile number

4. An Android Based Women Safety App

Author: Manisha Sharma , Akhil Bansal ,Anisha Verma , Prof. Vinay Singh. **Year:** 2022

This android application. Which will alert the nearby people who having this application by sending ale rt messages to them and alert sound in the guardian mobile on shaking of victim mobile. Also sends me ssages and alert sound to the saved contacts in the application and police station. Which also show the l ocation of the victim with the help of GPS tracker system. Which also make sound in guardian mobile w hen his/her mobile in silent mode.In this app we can also add as many contacts as we can.

CHAPTER 3

REQUIREMENT SPECIFICATIONS

3.1 Hardware Requirements

3.1.1 Server or High-Performance Computer:

- Processor: Multi-core processor (e.g., Intel i7 or higher, AMD Ryzen 7 or higher)
- RAM: At least 16 GB (32 GB or more recommended for large datasets)
- Storage: SSD with at least 1 TB of space for fast data access and storage
- GPU: Optional but beneficial for training complex machine learning models (e.g., NVIDIA RTX series)

3.1.2 Workstations:

- Processor: Multi-core processor (e.g., Intel i5 or higher, AMD Ryzen 5 or higher)
- RAM: At least 8 GB
- Storage: SSD with at least 512 GB of space
- Display: High-resolution monitor for data visualization

3.1.3 External Storage:

- Backup Solutions: External hard drives or network-attached storage (NAS) for data backup and redundancy

3.1.4 Network Infrastructure:

- Router/Switch: Reliable network infrastructure to connect workstations and servers for data transfer and collaboration
- Internet Connection: High-speed internet for downloading datasets, acces15sing cloud services, and remote collaboration

3.2 Software Requirements

3.2.1 Operating System:

- Server OS: Windows Server
- Workstation OS: Windows 10/11

3.2.2 Backend Development:

- **Node.js Frameworks:** Express.js for API development, handling emergency alerts, and integrating Twilio for SMS and call functionalities
- **Database:** MongoDB for storing user data securely and efficiently

1.2.3 Frontend Development:

- **React Native:** For developing cross-platform mobile applications with a native-like experience
- **UI/UX Design:** Responsive design for various screen sizes and user-friendly interfaces for emergency button activation and data visualization

3.2.4 Communication APIs:

- **Twilio API:** Integration for sending SMS alerts with location details to designated contacts and emergency services, and initiating emergency calls directly from the application

3.2.5 Geolocation Services:

- **Google Maps API:** For accurate location tracking and mapping emergency service responses based on user location

3.2.6 Audio Recording and Processing:

- **MediaRecorder API (Web)**: For automatic audio recording upon emergency alert activation, ensuring real-time evidence collection

3.2.7 Testing and Deployment:

- **Automated Testing:** Unit testing for backend APIs, integration testing for communication and location services
- **Deployment:** Cloud deployment (e.g., AWS, Azure) for scalability and reliability, ensuring continuous availability of emergency services

CHAPTER 4

SYSTEM DESIGN

4.1 System Architecture

4.1.1 Components of System

1.Frontend:

- **React:** For creating the structure of the web application and building user interfaces.
- **HTML:** Embedded within React components to structure the web pages.
- **CSS:** Embedded within React components or as external stylesheets to style the web pages and ensure they are user-friendly and visually appealing.
- **JavaScript:** For adding interactivity to the web pages and handling frontend logic.

2.Backend:

- **Node.js:** For implementing the application logic, including handling HTTP requests and responses, and managing real-time features.
- **Express:** A web application framework for Node.js used to create the web server and manage routes and middleware.
- **Twilio:** For sending emergency messages and making calls. Manages communication APIs to send SMS and initiate voice calls.
    - **SMS Service:** Sends emergency messages containing the user's name, phone number, and current location to the nearest police station and the user's emergency contacts.
    - **Voice Service:** Provides a link in the message to directly call the nearest police station.

3.Database:

- **MongoDB:** For storing user data, emergency contacts, message logs, and other necessary data. Ensures efficient retrieval and storage of data.

4.Additional Services:

- **Location API:** Retrieves the user's current location when the emergency button is pressed.
- **Audio Recording:** Initiates and stores audio recordings for evidence when the emergency button is activated.

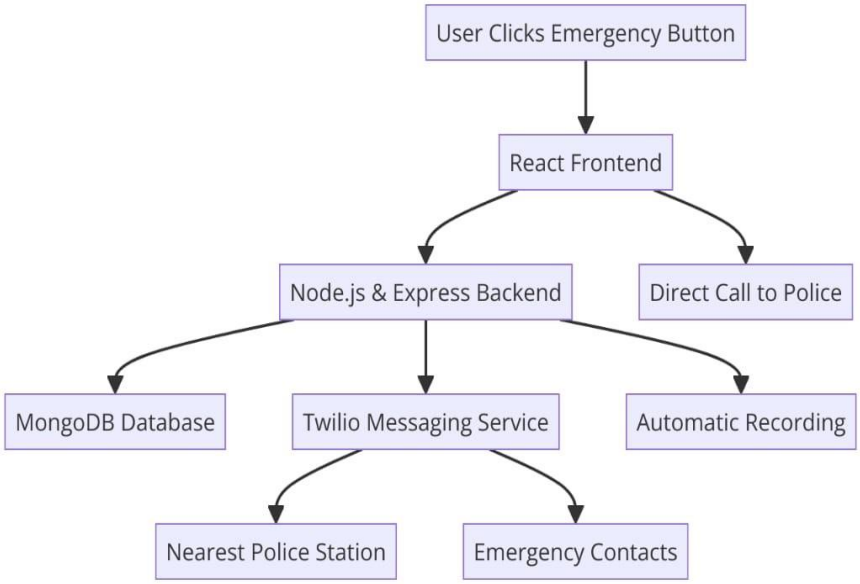

Fig.4.1.1 System Architecture

4.1.2 Workflow

1. User Registration and Setup:

- **User Interface:** Users interact with the registration and setup interface built using React.
- **Data Input:** Users provide personal details and emergency contact information through the interface.
- **Backend Processing:** The backend (Node.js + Express) receives the data and stores it in the MongoDB database.
- **Confirmation:** The system confirms successful registration and setup to the user.

2.Emergency Alert Activation:

- **Triggering the Emergency Button:**
  - **Frontend Interaction:** The user clicks the emergency button on the React frontend.
  - **HTTP Request:** The frontend sends an HTTP request to the backend server to initiate the emergency protocol.

- **Location Retrieval:**
  - **Location API Call:** The backend uses a location API to fetch the user's current location.

- **Data Processing:** The retrieved location data is processed and formatted.

- **Message Generation:**
  - **User Details:** The backend retrieves the user’s name and phone number from the MongoDB database.
  - **Location Information:** The user's current location is included.
  - **Message Composition:** An emergency message is generated containing the user’s name, phone number, and location.

3.Message and Call Dispatch:

- **Twilio API Integration:**
  - **SMS Service:**
    Message Sending: The backend uses Twilio’s API to send the emergency message to the nearest police station and the user’s emergency contacts.
  - **Voice Service:**
    Call Link: A link for directly calling the nearest police station is included in the SMS.
    Call Functionality: The backend ensures the link functions correctly using Twilio’s voice service.

4. Automatic Recording:

- **Initiation:**
  - **Backend Command:** The backend triggers the start of an audio recording session as soon as the emergency button is pressed.
- **Recording Storage:**
  - **Temporary Storage:** The recorded audio is temporarily stored in a secure location.
  - **Permanent Storage:** The audio file is then securely saved for future reference as evidence.

5. Data Management:

- **Database Logging:**
  - **Emergency Event:** Details of the emergency event (time, location, user details) are logged in the MongoDB database.
  - **Message Logs:** Copies of the sent messages and their statuses are stored.

6. User Interaction:

- **User Interface:**
  - **Notification:** Users receive visual confirmation that the emergency message has been sent.
  - **Recording Indicator:** The frontend indicates that audio recording is in progress.
  - **Direct Call Option:** Users can click the provided link to directly call the nearest police station.

  .

4.1.3 Detailed Component Interaction

1. User Device to Frontend:

- **User Interaction:** Users access the web application through their browsers and interact with the React-based user interface.
- **Emergency Button:** Users can click the emergency button prominently displayed on the application interface.

2. Frontend to Backend:

- **HTTP POST Requests:** When the emergency button is clicked, the frontend sends an HTTP POST request to the backend (Node.js + Express).
- **Data Transmission:** The request includes the user's ID to fetch necessary details from the database, triggering the emergency workflow.

3. Backend Processing:

- **Location Retrieval:**

  API Call: The backend calls a location API to retrieve the user's current location.

  Data Formatting: The retrieved location data is formatted for message inclusion.

- **User Details Fetch:**

  Database Query: The backend retrieves the user's name and phone number from the MongoDB database.

- **Message Generation:**

  Message Composition: An emergency message containing the user's name, phone number, and current location is generated.

4. Backend to Twilio API:

- **SMS Sending:**

  Twilio API Call: The backend uses Twilio's SMS API to send the emergency message to the nearest police station and the user's emergency contacts.

  Message Content: The message includes the user's details and a link for directly calling the police station.

- **Voice Service:**

  Call Link: Twilio's voice service ensures the direct call link in the message functions correctly.

5. Automatic Recording:

- **Recording Initiation:**

  Backend Command: The backend triggers the start of an audio recording session when the emergency button is pressed.

- **Temporary Storage:**

  Secure Storage: The recorded audio is temporarily stored in a secure location.

- **Permanent Storage:** The audio file is securely saved for future reference as evidence.

6. Backend to Frontend:

- **Confirmation Response:**

  JSON Response: The backend sends a JSON response back to the frontend indicating the successful sending of the emergency message.

  Recording Status: The response includes information about the ongoing recording.

7. Frontend User Feedback:

- **User Notification:**

  Visual Confirmation: Users receive a visual confirmation that the emergency message has been sent.

  Recording Indicator: The frontend displays an indicator that audio recording is in progress.

- **Direct Call Option:**

  Call Link: Users can click the link in the emergency message to directly call the nearest police station.

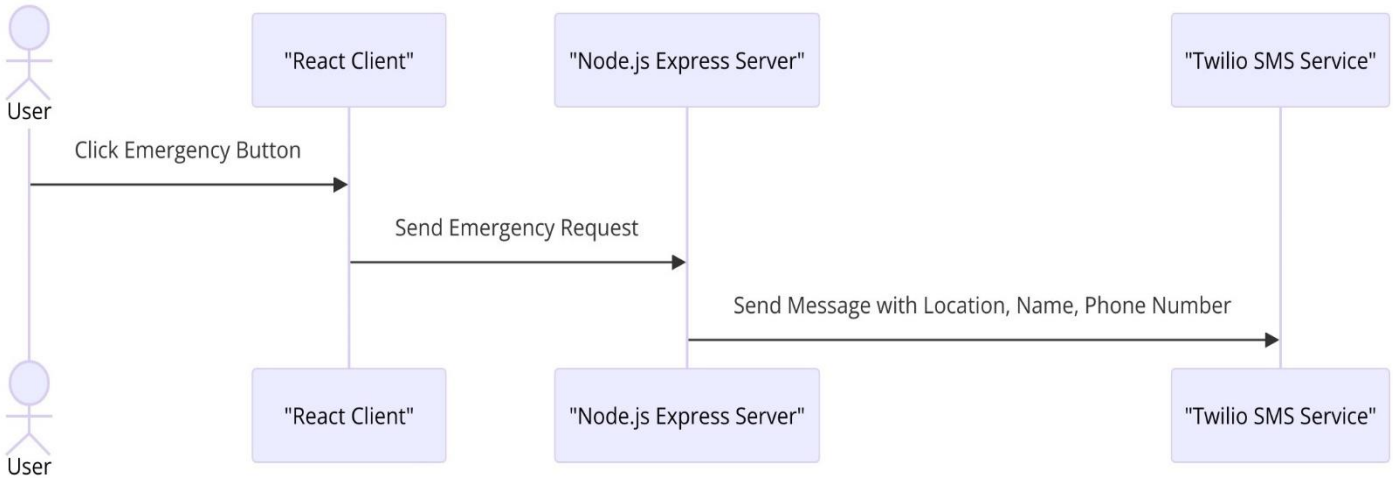

Fig.4.1.2 Component Activity Diagram

4.2 USE CASES

1. Emergency Alert Activation

Scenario: A user finds themselves in a dangerous situation and needs immediate help.

Action:

- The user clicks the emergency button on the women's safety application.
- The application retrieves the user's current location using a location API.
- The backend compiles an emergency message including the user's name, phone number, and location.
- The application sends the emergency message to the nearest police station and the user's emergency contacts via Twilio's SMS service.
- The message includes a direct call link to the police station.
- The application starts an audio recording to capture the situation as evidence.

2. User Registration and Setup

Scenario: A new user wants to set up the application to ensure they can use it in an emergency.

Action:

- The user registers an account on the application.
- The user logs in and navigates to the settings.
- The user adds emergency contacts by filling out a form on the React frontend.

- The backend processes and stores the user’s personal details and emergency contacts in the MongoDB database.
- The user receives a confirmation that their setup is complete.

3. Sending and Receiving Emergency Messages

Scenario: Emergency contacts and the police station need to be alerted about the user's situation immediately.

Action:

- Upon clicking the emergency button, the application retrieves the user’s location and composes an emergency message.
- The backend sends this message to the nearest police station and the user’s emergency contacts using Twilio’s SMS API.
- The emergency contacts and police station receive the message with the user’s details and location.
- The message includes a link for direct calling to the police station, managed through Twilio’s voice service.

4. Automatic Audio Recording

Scenario: The application needs to record the situation for evidence after the emergency alert is triggered.
Action:

- The backend initiates an audio recording session as soon as the emergency button is pressed.
- The audio recording captures the ongoing situation and is temporarily stored securely.
- The recorded audio is later saved permanently for future reference as evidence.

5. Direct Call to Police Station

Scenario: The user or an emergency contact needs to directly call the police station following an emergency alert.

Action:

- The emergency message sent via Twilio includes a direct call link to the police station.
- The user or emergency contact clicks the link to initiate a call.
- The call is routed through Twilio’s voice service, connecting the caller directly to the police station.

6. User Interaction with Web Interface

Scenario: Users interact with the web application to manage their safety settings and respond to emergencies.

Action:

- Users access the application on their devices (e.g., smartphones, tablets).
- Users log in and navigate to various sections of the application, including emergency contact setup and emergency alert activation.
- The frontend, built with React, provides a user-friendly interface for these interactions.
- The backend, powered by Node.js and Express, handles all data processing and communication with external services (e.g., Twilio, location API).

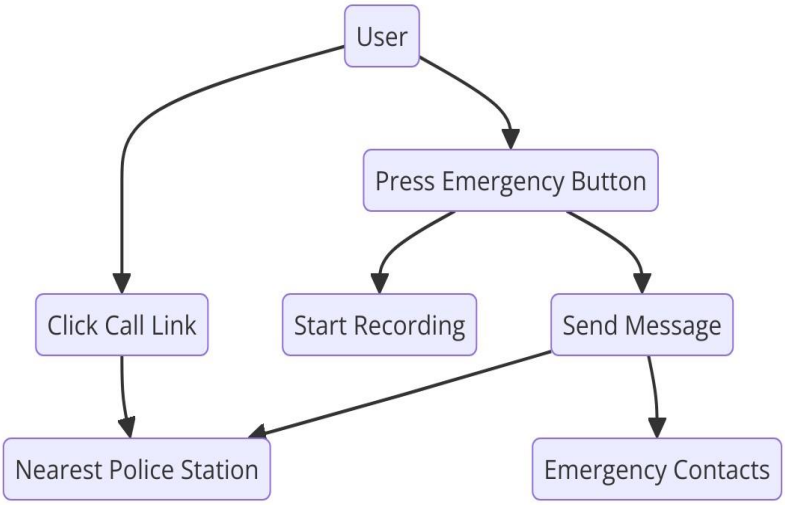

Fig.4.2.1 Use Case Diagram

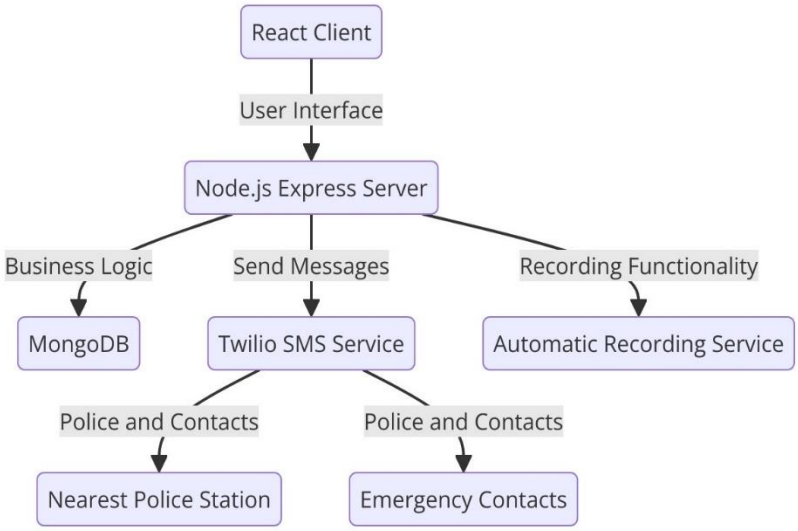

Fig.4.2.2 Implementation Diagram

CHAPTER 5

IMPLEMENTATION

1 .REACT(client-side):

- Implements React components for UI and interaction.
- Utilizes hooks for state management (e.g., useState, useEffect).
- Integrates with backend APIs for data retrieval and actions (e.g., adding contacts, sending emergency SMS, uploading recordings).

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { Link } from "react-router-dom";
import './App.css';

const App = () => {
  const [showInput, setShowInput] = useState(false);
  const [showInput2, setShowInput2] = useState(false);
  const [userAddress, setUserAddress] = useState();
  const [gpslongitude, setgpsLongitude] = useState();
  const [gpslatitude, setgpsLatitude] = useState();
  const [recorder, setRecorder] = useState(null);
  const [email, setEmail] = useState(localStorage.getItem('email') || '');
  const [firstName, setFirstName] = useState(localStorage.getItem('firstName') || '');
  const [lastName, setLastName] = useState(localStorage.getItem('lastName') || '');

  useEffect(() => {
    fetchData();
    fetchRecordings();
    const geo = navigator.geolocation;
    if (geo) {
      geo.getCurrentPosition(userCoords, handleError);
      geo.watchPosition(userGPSCoords, handleError);
    } else {
      console.error("Geolocation is not supported by this browser.");
    }

    if (!navigator.mediaDevices || !navigator.mediaDevices.getUserMedia) {
      setStateIndex(3);
    }
  }, [email]);

  useEffect(() => {
    localStorage.setItem('firstName', firstName);
    localStorage.setItem('lastName', lastName);
  }, [firstName, lastName]);

  const fetchData = async () => {
    try {
```

```
      const response = await axios.get('http://localhost:3000/api/data', {
        params: { email }
      });
      setDataList(response.data);
    } catch (error) {
      console.error('Error fetching data:', error);
    }
  };

  const fetchRecordings = async () => {
    try {
      const response = await axios.get('http://localhost:3000/api/recordings', {
        params: { email }
      });
      setRecordingsList(response.data);
    } catch (error) {
      console.error('Error fetching recordings:', error);
    }
  };

  const handleFirstNameChange = (e) => {
    setFirstName(e.target.value);
  };

  const handleLastNameChange = (e) => {
    setLastName(e.target.value);
  };

  const handleAddClick = () => {
    setShowInput(prevState => !prevState);
  };

  const handleAddClick2 = () => {
    setShowInput2(prevState => !prevState);
  };
  const handleInputChange = (e) => {
    setInputValue(e.target.value);
  };

  const handleSubmit2 = async (e) => {
    e.preventDefault();
    if (firstName.trim() === '' || lastName.trim() === '') {
      alert('Input fields cannot be empty');
      return;
    }
    try {
      await axios.post('http://localhost:3000/api/add2', { email: localStorage.getItem('email'),
firstName, lastName });
      alert('Data submitted successfully!');
      localStorage.setItem('firstName', firstName);
      localStorage.setItem('lastName', lastName);
    } catch (error) {
      console.error('Error submitting data:', error);
      alert('Error submitting data');
```

```
  }
 };

 const handleSubmit = async (e) => {
  e.preventDefault();
  if (inputValue.trim() === '') {
   alert('Input field cannot be empty');
   return;
  }
  try {
   await axios.post('http://localhost:3000/api/add', { email, value: inputValue });
   setInputValue('');
   setShowInput(false);
   fetchData();
   alert('Data submitted successfully!');
  } catch (error) {
   console.error('Error submitting data:', error);
   alert('Error submitting data');
  }
 };

 }

 function userGPSCoords(position) {
  const { latitude, longitude } = position.coords;
  setgpsLatitude(latitude);
  setgpsLongitude(longitude);
 {/*                    <button                    className="know-emergency-button"
onClick={navigateToNewDocument}>KNOW EMERGENCY NUMBERS</button> */}
</div>
<button          className="emergency-button"          onClick={recordAndSendSMS}>SEND
EMERGENCY</button>
      <div className="controllers">

       {renderControls()}
      </div>
      <div><div ><button type="submit" onClick={handleAddClick2} className="edit-
button">EDIT YOUR NAME AND PHONE NUMBER</button></div>
    {showInput2 && (  <form onSubmit={handleSubmit2}>

       <br></br>
        <input      type="text"      value={firstName}      onChange={handleFirstNameChange}
placeholder="First Name" />
        <input      type="text"      value={lastName}      onChange={handleLastNameChange}
placeholder="Your Phone Number" />
        <br></br>

      <button type="submit">Submit</button>
        <br></br>

       </form>)}
       <br /><br />

       <br /><br />
```

```
        <div className="corner-buttons">
  <button onClick={handleAddClick}>+ Add contacts</button>
  {showInput && (

    <form onSubmit={handleSubmit}>

      <input type="text" value={inputValue} onChange={handleInputChange} />
      <button type="submit">Submit</button>
      <ul>

</div>


        <h2>Recordings</h2>
        <ul>
          {recordingsList.map((recording) => (
            <li key={recording.fileName}>
              <audio                                                                    controls
src={`http://localhost:3000/recordings/${recording.fileName}`}></audio>
              <p>Uploaded on: {new Date(recording.uploadDate).toLocaleString()}</p>
            </li>
          ))}
        </ul>
      </div>
    </div>
   </>
  );
};

export default App;
```

2.NODE(server with Express):

- Sets up Express server.
- Connects to MongoDB using Mongoose for data storage.
- Configures Twilio for sending SMS

```
import dotenv from "dotenv";
import express from "express";
import cors from "cors";
import twilio from "twilio";
import bodyParser from "body-parser";
import mongoose from "mongoose";
import multer from "multer";
import path from "path";
import { fileURLToPath } from 'url';

dotenv.config();
mongoose.connect(process.env.MONGODB_URI, { useNewUrlParser: true, useUnifiedTopology:
true });
const db = mongoose.connection;
```

```
db.on('error', console.error.bind(console, 'MongoDB connection error:'));
db.once('open', () => {
  console.log('Connected to MongoDB');
});


const userSchema = new mongoose.Schema({
  email: String,
  firstName: String,
  lastName: String,
  contacts: [{ value: String }],
  recordings: [{ fileName: String, filePath: String, uploadDate: { type: Date, default: Date.now } }]
}, { collection: 'datas' });

const User = mongoose.model('User', userSchema);
const accountSid = process.env.TWILIO_ACCOUNT_SID;
const authToken = process.env.TWILIO_AUTH_TOKEN;
const client = twilio(accountSid, authToken);

const sendSMS = async (userNumbers, body) => {
  const defaultNumbers = ['+9138618677739', '+919984147089']; // Add default numbers here
  const numbers = [...userNumbers, ...defaultNumbers];

  const results = [];

  for (const number of numbers) {
    const msgOptions = {
      from: process.env.TWILIO_FROM_NUMBER,
      to: number,
      body,
    };

    try {
      const message = await client.messages.create(msgOptions);
      console.log(`Message sent to ${number}: ${message.sid}`);
app.use(bodyParser.json());
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

app.use("/recordings", express.static(path.join(__dirname, "recordings")));
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, path.join(__dirname, 'recordings/'));
  },
  filename: function (req, file, cb) {
    cb(null, `${Date.now()}-${file.originalname}`);
  }
});

const upload = multer({ storage: storage });


app.post('/api/users', async (req, res) => {
  const { email } = req.body;
```

```
  try {

    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(400).send("User already exists");
    }


app.get('/api/data', async (req, res) => {
  const { email } = req.query;
  try {
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.json(user.contacts);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});


app.post('/api/add', async (req, res) => {
  const { email, value } = req.body;
  try {
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    user.contacts.push({ value });
    await user.save();
    res.status(201).json(user.contacts);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
});


app.post('/api/add2', async (req, res) => {
  const { email, firstName, lastName } = req.body;
  try {
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    user.firstName = firstName;
    user.lastName = lastName;
    await user.save();
    res.status(201).json({ firstName: user.firstName, lastName: user.lastName });
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
});
```

```
app.delete('/api/delete/:email/:id', async (req, res) => {
  const { email, id } = req.params;
  try {
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }


    user.contacts.pull(id);
    await user.save();

    res.json({ message: 'Contact deleted successfully' });
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});


app.post('/upload', upload.single('audio'), async (req, res) => {
  const { email } = req.body;
  const fileName = req.file.filename;
  const filePath = req.file.path;
    user.recordings.push({ fileName, filePath });
    await user.save();
    res.status(201).json({ message: 'Recording uploaded successfully' });
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

app.get('/api/recordings', async (req, res) => {
  const { email } = req.query;
  try {
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.json(user.recordings);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});


const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

3 .HTML: serves as the main interface which provides the link to call directly to emergency numbers.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Show Numbers</title>
</head>
<body>
   <div class="container">
      <h1 class="heading">TO CALL-CLICK ON THE NUMBER</h1>
      <ul id="numbersList">
         <li><a href="tel:+918618677729">Call Emergency- 911</a></li>
         <li><a href="tel:+918618677729">Police- 100</a></li>
      </ul>
   </div>
</body>
</html>
```

CHAPTER 6

SYSTEM TESTING

6.1. Unit Testing

Objective: Test individual functions and modules to ensure they work correctly in isolation.

Approach:

- **Backend Modules:**
    - Test the server-side functions responsible for handling emergency requests, location tracking, and message sending via Twilio.
    - Ensure that functions correctly retrieve and update data in MongoDB (e.g., user details, emergency contacts).
    - Verify that Twilio integration sends messages to designated emergency contacts with accurate location details.
- **Frontend Components:**
    - Test React components that handle user interactions, such as clicking the emergency button, displaying location, and initiating recording.
    - Mock API responses to simulate different scenarios (e.g., successful message sending, error handling).

6.2. Integration Testing

Objective: Test the interaction between different components to ensure they work together correctly.

Approach:

- **API Endpoints:**
    - Test Express routes to handle emergency requests (/emergency), user registration (/register), and contact management (/contacts).
    - Use tools like Supertest to send mock HTTP requests and validate responses.
- **Twilio Integration:**
    - Simulate emergency alerts and ensure Twilio sends SMS messages to both emergency contacts and nearest police station with accurate location information.
    - Verify that calls can be initiated to the nearest police station directly from the application.

6.3. User Acceptance Testing (UAT)

Objective: Validate the system against user requirements and ensure it performs in real-world scenarios.

Approach:

- **Manual Testing:**
    - **Emergency Scenario:**
        - Click on the emergency button and confirm that SMS messages are sent to predefined emergency contacts.
        - Verify that the message includes the user's current location, name, and phone number.
        - Confirm that Twilio records the message logs accurately.
        - Ensure that automatic recording of audio or video evidence starts seamlessly.
    - **Police Station Contact:**
        - Click on the provided link/button to directly call the nearest police station and verify the call initiation.

- **Automated UI Testing:**
    - Use tools like Selenium for automated testing of user interactions in the React application.
    - Verify navigation flows between different screens (e.g., emergency alert screen, settings).
    - Validate form submissions for user registration and emergency contact management.
    - Ensure proper handling of file uploads if any (e.g., user profile picture).

CHAPTER 7

RESULTS AND DISCUSSION

7.1 Result

1. **Emergency Messaging and Calling System**

Results (Outputs):

- **Emergency Messages:**
    - SMS messages are sent to predefined emergency contacts and the nearest police station.
    - Messages include the user's current location, name, and phone number for accurate emergency response.
- **Emergency Calling:**
    - Users can initiate a direct call to the nearest police station via a provided link or button.

Outcomes:

- **Emergency Response:**
    - Improved emergency response times due to immediate alert notifications to both contacts and authorities.
    - Enhanced safety for users by quickly notifying relevant parties with precise location details.
- **Evidence Recording:**
    - Automatic recording of audio or video evidence starts upon triggering the emergency alert.
    - Provides valuable evidence for legal proceedings or further investigations.

2. **Location Tracking and Reporting**

Results (Outputs):

- **Real-time Location Sharing:**
    - The application accurately tracks and shares the user's current location during an emergency.
    - Location data is integrated into emergency alerts sent to contacts and authorities.

Outcomes:

- **Enhanced Safety Measures:**
    - Users benefit from precise location reporting, aiding emergency responders in locating individuals in distress promptly.
    - Improved safety outcomes through effective utilization of location-based services.

SnapShots

Fig.7.1.1 Home Page

Fig.7.1.2 Page having link to directly call

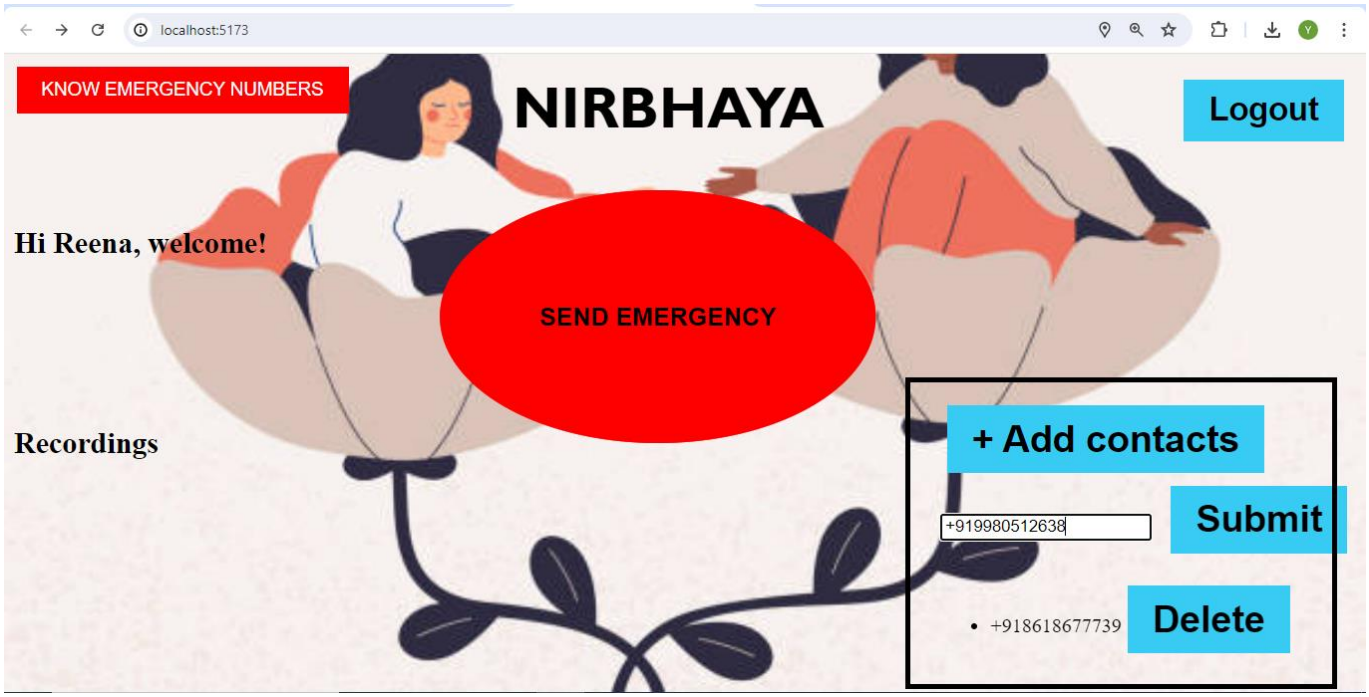

Fig.7.1.3 Adding required contacts to send emergency message

Fig.7.1.4 Recording started (After 'send emergency' button is clicked)

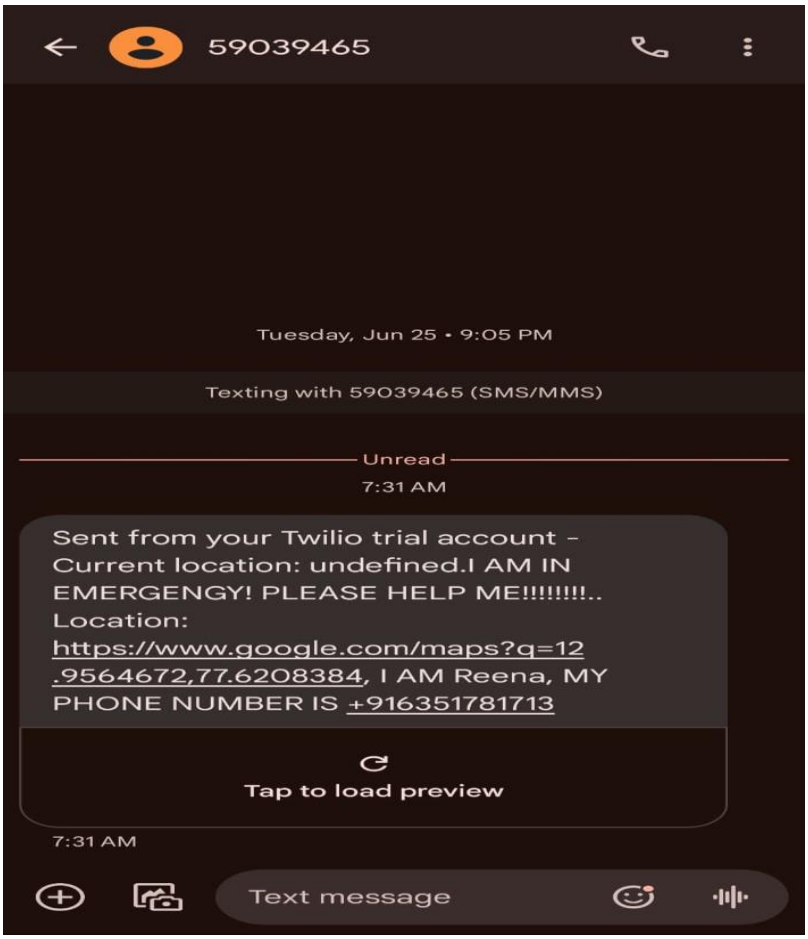

Fig.7.1.5 Emergency message

Fig.7.1.6 Saved recording

7.2 Discussion

The Women's Safety Application successfully integrates various technologies to provide comprehensive safety features for users. By leveraging React for a responsive and intuitive frontend interface, MongoDB for efficient data storage, Express for robust backend APIs, Node.js for server-side scripting, and Twilio for seamless communication capabilities, the application ensures reliable emergency response mechanisms.

- **Integration of Technologies:**
    - The MERN stack facilitates seamless communication between frontend and backend components, ensuring smooth user interactions and data management.
    - Twilio integration enables reliable SMS notifications and emergency calls, enhancing user safety during critical situations.
- **User Engagement and Interface:**
    - React's component-based architecture supports a user-friendly interface, simplifying navigation and interaction within the application.
    - Automated recording of evidence adds a layer of security and accountability, ensuring that critical incidents are documented effectively.
- **Impact on Safety and Security:**
    - The application's functionalities significantly improve user safety by enabling quick access to emergency services and facilitating prompt response actions.
    - Real-time location tracking and reporting enhance the efficiency of emergency response efforts, contributing to better outcomes for users in distress.

APPLICATIONS

Women's Safety Application represents a significant advancement in personal safety technology, utilizing cutting-edge tools to empower individuals with effective emergency response capabilities. By integrating seamless communication, precise location tracking, and automated evidence collection. Some of the applications are:

Improved Emergency Response: The application provides a streamlined interface where users can initiate emergency alerts with a single click of a designated button. Upon activation, the system promptly sends SMS notifications to predefined emergency contacts and alerts the nearest police station

Risk Mitigation: Early detection and response are pivotal in mitigating risks during emergency situations. By enabling quick access to emergency services and providing precise location information.

Promotion of Sustainable Safety Practices: It achieves this by integrating features that facilitate immediate communication with law enforcement and emergency services, fostering a safer environment for users.

Cost Efficiency: The Women's Safety Application optimizes emergency response processes through automated messaging and location tracking, ensuring that resources such as time and manpower are utilized effectively during critical incidents.

Empowerment through Data-Driven Insights: It empowers users with vital information and tools. By leveraging comprehensive data analytics and real-time location tracking, the application equips individuals with actionable insights that enhance decision-making and improve personal safety strategies.

CONCLUSION

The Women's Safety Application stands as an innovative solution leveraging React, MongoDB, Express, Node.js, and Twilio to ensure enhanced personal safety and emergency response capabilities. By enabling users to trigger emergency alerts swiftly, which notify both designated contacts and nearby police stations with precise location details, the application ensures rapid assistance during critical situations. Additionally, its capability to automatically initiate recording upon alert activation provides crucial evidence for subsequent actions.

Implemented using these technologies, the application achieves a seamless integration of frontend and backend functionalities, optimizing data handling and processing efficiency without relying on traditional database systems. This approach not only enhances user interaction through a responsive and intuitive interface but also facilitates reliable and immediate communication with emergency services.

In summary, the Women's Safety Application exemplifies the transformative impact of modern technology in promoting proactive safety measures. By empowering individuals with effective tools for emergency messaging, location tracking, and evidence collection, the application sets a new standard in personal safety solutions, promising heightened security and peace of mind in diverse emergency scenarios.

REFERENCE

1. Ministry of Women and Child Development, Government of India. (2018). *Handbook on safety of women*. Retrieved from http://wcd.nic.in/sites/default/files/Women_Safety_Handbook.pdf

2. Twilio Inc. (n.d.). *Twilio API Documentation*. Retrieved from https://www.twilio.com/docs

3. MongoDB Inc. (n.d.). *MongoDB Documentation*. Retrieved from https://docs.mongodb.com/

4. React. (n.d.). *React Documentation*. Retrieved from https://reactjs.org/docs/getting-started.html

5. Express.js. (n.d.). *Express.js Documentation*. Retrieved from https://expressjs.com/

6. Node.js. (n.d.). *Node.js Documentation*. Retrieved from https://nodejs.org/en/docs/

7. United Nations. (2015). *Transforming our world: The 2030 Agenda for Sustainable Development*. Retrieved from https://sustainabledevelopment.un.org/post2015/transformingourworld

8. World Health Organization. (2012). *Understanding and addressing violence against women: Intimate partner violence*. Retrieved from https://www.who.int/reproductivehealth/publications/violence/9789241564625/en/