Application

Description automatically generated with low confidence

**SMART WATER FOUNTAINS USING IoT**



**Prepared By,**

Saravanan N,

Meiyarasan S,

Kaviyarasu G,

Ganeshmoorthy E

**1. Introduction**

The Real-time Water Fountain Monitoring Platform is a web-based application designed to receive, process, and display real-time data from water fountains. The platform aims to monitor the water flow rate and detect malfunctions in water fountains. This report outlines the design, implementation, and features of the platform using web development technologies such as HTML, CSS, and JavaScript.

**2. Project Scope**

The project scope includes:

* Designing a user-friendly web interface to visualize real-time water fountain data.
* Implementing a backend system to receive and process data from water fountain sensors.
* Displaying water flow rate information graphically.
* Sending malfunction alerts to users in real-time.

**3. Technologies Used**

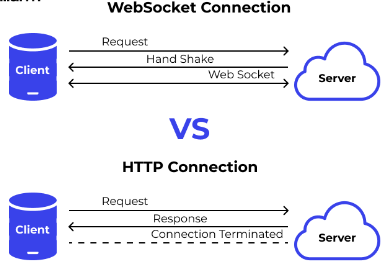
* **Frontend:** HTML, CSS, JavaScript, Bootstrap for responsive design.
* **Backend:** Node.js for server-side scripting, Express.js for routing.
* **Database:** MongoDB for storing fountain data.
* **Real-time Communication:** Socket.io for real-time communication between the server and clients.
* **Web Server:** Express.js
* **Charting Library:** Chart.js for visual representation of data

**4. System Architecture**

The platform follows a client-server architecture. Clients (web browsers) send requests to the server, which processes data from sensors and sends real-time updates back to the clients via WebSocket.

**Client-Side:** HTML, CSS, JavaScript for the user interface

**Server-Side:** Node.js, Express.js for handling requests, Socket.io for real-time communication, MongoDB for data storage.



**5. Features**

**User Authentication:**

Secure login system for authorized access.

**Real-time Data Visualization:**

Graphical representation of water flow rates using charts and graphs.

**Malfunction Alerts:**

Instant notifications to users via email or in-app notifications when malfunctions are detected.

**Historical Data:**

Store and display historical data for analysis and trend monitoring.

**User Profiles:**

Allow users to customize their profiles and set notification preferences.

**6. Implementation**

**Frontend Design:**

Create an intuitive and responsive user interface using HTML, CSS, JavaScript and Bootstrap.

* **HTML (index.html)**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Water Fountain Monitoring System</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

<div id="fountain-data"></div>

</div>

<script src="https://cdn.socket.io/4.3.1/socket.io.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

<script src="script.js"></script>

</body>

</html>

* **CSS (styles.css)**

body {

font-family: Arial, sans-serif;

}

container {

max-width: 800px;

margin: 0 auto;

padding: 20px;

}

canvas {

width: 100%;

height: 400px;

}

* **JavaScript (script.js)**

const socket = io();

socket.on('fountainData', function(data) {

// Update UI with real-time data

// Use Chart.js to plot data on canvas

});

socket.on('malfunctionAlert', function(alert) {

// Display malfunction alert to the user

});

// Code for initializing and updating charts using Chart.js

**Backend Development:** Develop server-side logic using Node.js and Express.js to handle incoming requests and manage data processing.

* **Node.js (server.js)**

const express = require('express');

const http = require('http');

const socketIo = require('socket.io');

const mongoose = require('mongoose');

const app = express();

const server = http.createServer(app);

const io = socketIo(server);

// Connect to MongoDB database

mongoose.connect('mongodb://localhost/fountainData', { useNewUrlParser: true, useUnifiedTopology: true });

const db = mongoose.connection;

// Fountain data schema and model setup using Mongoose

const fountainSchema = new mongoose.Schema({

flowRate: Number,

timestamp: { type: Date, default: Date.now }

});

const Fountain = mongoose.model('Fountain', fountainSchema);

// Socket.io connection event

io.on('connection', (socket) => {

console.log('A user connected');

// Retrieve and emit real-time data from the database

Fountain.find().sort({ timestamp: -1 }).limit(10).exec((err, data) => {

if (err) throw err;

socket.emit('fountainData', data);

});

// Simulated malfunction detection

setInterval(() => {

const randomFlowRate = Math.random() \* 100; // Random flow rate data

if (randomFlowRate < 20) {

socket.emit('malfunctionAlert', 'Fountain malfunction detected!');

}

}, 5000);

socket.on('disconnect', () => {

console.log('User disconnected');

});

});

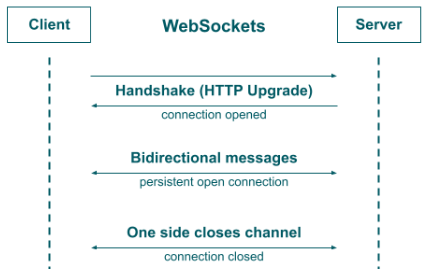
server.listen(3000, () => {

console.log('Server is running on port 3000');

});

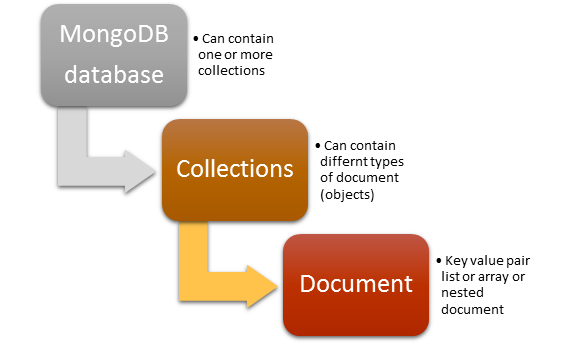
**Real-time Communication:**

Implement WebSocket communication using Socket.io for real-time updates.



**Database Integration:**

Use MongoDB to store and retrieve fountain data efficiently.



**Security:**

Implement encryption and validation mechanisms to ensure data security and integrity.

**Notifications:**

Integrate email or push notification services for alerting users about malfunctions.

**7. Future Enhancements**

**Geolocation Integration:**

Display fountains on a map for easier navigation.

**Machine Learning:**

Implement predictive maintenance using machine learning algorithms to detect potential malfunctions before they occur.

**Mobile Application:**

Develop a mobile app version for on-the-go monitoring and notifications.

**Integration with IoT Devices:**

Connect with IoT sensors for more comprehensive data collection

.

**8. Conclusion:**

The Real-time Water Fountain Monitoring Platform provides an effective solution for monitoring water fountains in real-time. By utilizing web development technologies and real-time communication tools, the platform ensures accurate data visualization and timely alerts, enabling users to respond promptly to fountain malfunctions. With its user-friendly interface and robust features, the platform enhances the efficiency of fountain management and contributes to water conservation efforts.