Project Tittle: smart water fountains

Phase 4: Development part 2

- Topic:
- continue building the project by
- performing different activities like feature
- engineering, model training, evaluation etc as
- per the instructions in the project.

SMART WATER FOUNTAINS

- INTRODUCTION
- Smart water fountains represent a cutting-edge innovation in the
- field of water management and technology. These fountains go beyond
- traditional water dispensers by incorporating advanced features and
- technologies to provide efficient, sustainable, and user-friendly
- solutions for various applications. The primary objective of smart water
- fountains is to optimize water usage, enhance user experience, and
- promote environmental sustainability.

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.

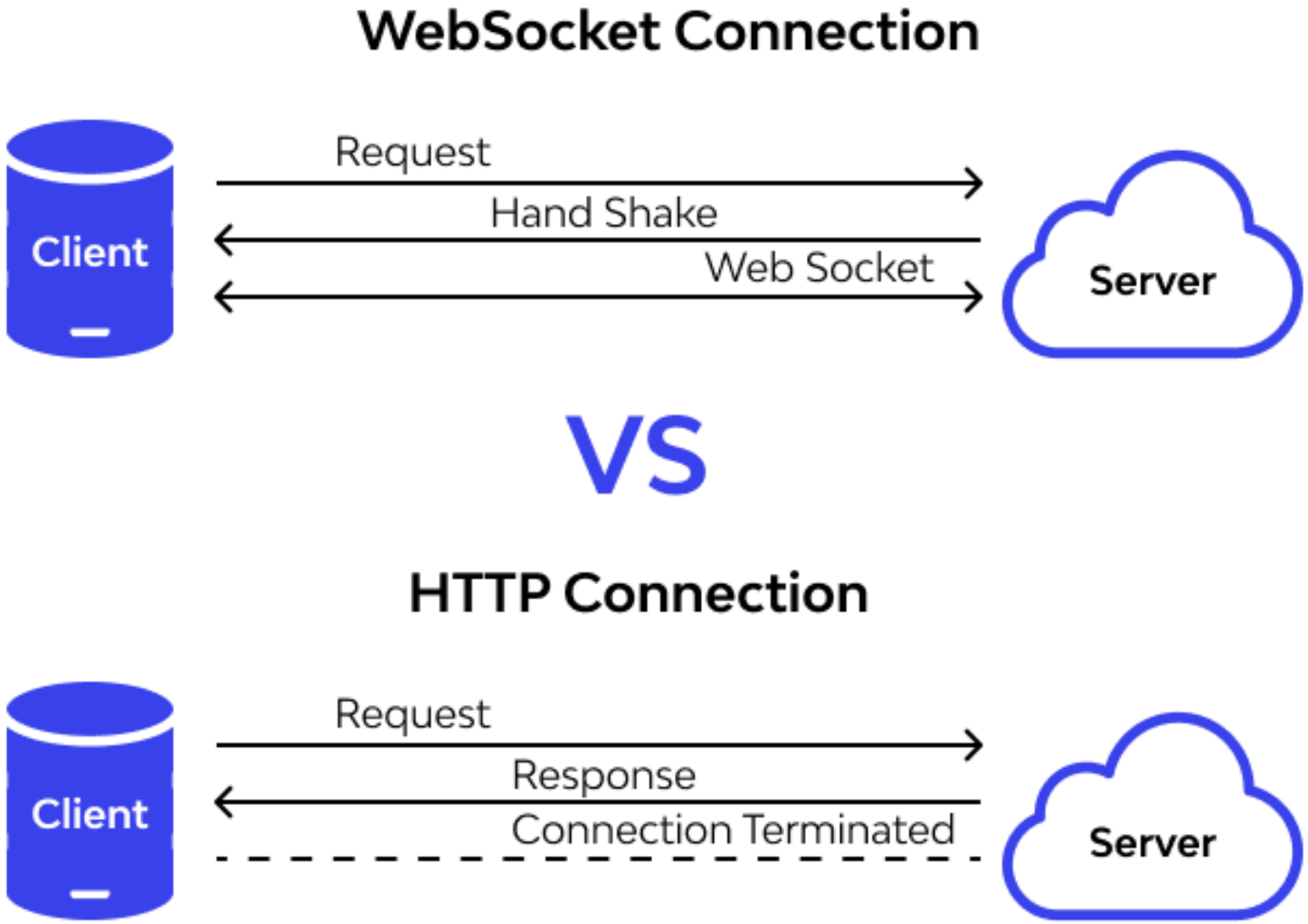
Technologies Used

- **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

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.

Web Socket vs HTTP



Implementation

- *Frontend Design:*
- **HTML (index.html)**

```
<!DOCTYPE html>
<html>
<head>
    <title>Water Fountain Status</title>
    <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
    <h1>Water Fountain Status</h1>
```

```
    <div class="status">
        <div id="flow-rate">
            <h2>Water Flow Rate</h2>
            <p id="flow-rate-value">Loading...</p>
        </div>
        <div id="malfunction">
            <h2>Malfunction  Alerts</h2>
            <ul id="malfunction-list">
                <li>Loading...</li>
```

```
            </ul>
        </div>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

CSS (styles.css)

```
body {
    font-family: Arial,
sans-serif;
    text-align: center;
    background-color:
#f0f0f0;
    padding: 20px;
}
```

```
h1 {
    color: #333;
}

.status {
    display: flex;
    justify-content:
space-around;
    margin: 20px;
}

#flow-rate,
#malfunction {
    background-color:
#fff;
```

```
    padding: 20px;
    border: 1px solid #ccc;
    border-radius: 5px;
    box-shadow: 0 4px 8px 0 rgba(0,
0, 0, 0.2);
}

#malfunction-list {
    list-style-type: none;
    padding: 0;
}
```

JavaScript (script.js)

```
// Simulated real-time data (replace with actual data
source)
let waterFlowRate = 0;
let malfunctionAlerts = ["No alerts"];

// Function to update the display with real-time data
function updateStatus() {
    // Fetch real-time data here, e.g., using AJAX or
WebSockets
    // Update waterFlowRate and malfunctionAlerts with
the fetched data
    // For now, simulate data updates
    waterFlowRate = Math.random() * 10; // Replace with
real data
    malfunctionAlerts = ["No alerts"]; // Replace with real
data
```

```
    // Update the HTML elements
    document.getElementById("flow-rate-
value").textContent = waterFlowRate.toFixed(2) + "
GPM";
    const malfunctionList =
document.getElementById("malfunction-list");
    malfunctionList.innerHTML = ""; // Clear previous
alerts
    malfunctionAlerts.forEach((alert) => {
        const listItem = document.createElement("li");
        listItem.textContent = alert;
        malfunctionList.appendChild(listItem);
    });
}

// Update data every 5 seconds (adjust the interval as
needed)
setInterval(updateStatus, 5000);
```

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(&#39;express&#39;);
const http = require(&#39;http&#39;);
const socketIo = require(&#39;socket.io&#39;);
const mongoose = require(&#39;mongoose&#39;);
const app = express();
const server = http.createServer(app);
const io = socketIo(server);
// Connect to MongoDB database
mongoose.connect(&#39;mongodb://localhost/fountainData&#39;, {
```

```
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(&#39;Fountain&#39;, fountainSchema);
// Socket.io connection event
io.on(&#39;connection&#39;, (socket) =&gt; {
console.log(&#39;A user connected&#39;);
```

```
// Retrieve and emit real-time data from the database
Fountain.find().sort({ timestamp: -1 }).limit(10).exec((err, data)
=&gt; {
if (err) throw err;
socket.emit(&#39;fountainData&#39;, data);
});
// Simulated malfunction detection
setInterval(() =&gt; {
const randomFlowRate = Math.random() * 100; // Random
flow rate
data
if (randomFlowRate &lt; 20) {
socket.emit(&#39;malfunctionAlert&#39;, &#39;Fountain
malfunction detected!&#39;);
}
```

```
}, 5000);
socket.on(&#39;disconnect&#39;, () =&gt; {
console.log(&#39;User disconnected&#39;);
});
});
server.listen(3000, () =&gt; {
console.log(&#39;Server is running on port 3000&#39;);
});
```

Database Integration

Automatically uploading data from your local software to your website

Security:

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

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

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.