Water Management System

# Introduction

The Water Consumption Monitoring Website is the development of a web-based platform aimed at efficiently tracking and managing water consumption in public places, contributing to a sustainable future.

# Design Phase

## User Requirements

This section highlights the user requirements, including the target audience, their needs, and the importance of water conservation. It discusses specific user stories and use cases that guided the project.

As it is a website we want it to view real-time water consumption data for different locations to stay informed about water usage.

**Criteria:**

The homepage should be able to get real-time water consumption data.

Users can select different locations to view data for specific areas.

The data should update automatically without the need for manual refreshing.

## System Architecture

Describes the system architecture, the utilization of IoT sensors, data-sharing platform, and web development technologies (HTML, CSS, JavaScript, and frameworks). It details the flow of data from the sensors to the user interface.

The system architecture of the Water Consumption Monitoring website consists of the following components:

**IoT Sensors:**

These sensors are used to measure water consumption.

**Data-Sharing Platform:**

This platform collects data from the sensors and stores it in a MongoDB database.

**Web Application:**

The user interface is built using HTML, CSS, and JavaScript (with the React framework). It communicates with the data-sharing platform to retrieve and display data.

**MongoDB Database:**

Historical water consumption data is stored in a MongoDB database.

**Data Flow:**

1. IoT sensors collect water consumption data and send it to the data-sharing platform.

2. The data-sharing platform stores the data in the MongoDB database.

3. The web application fetches data from the database and displays it to users in real-time.

## Database Design

The MongoDB database system's design

// MongoDB Schema

const consumptionSchema = new mongoose.Schema({

    date: Date,

    location: String,

    consumption: Number,

});

// Model

const Consumption = mongoose.model('Consumption', consumptionSchema);

## User Interface Design

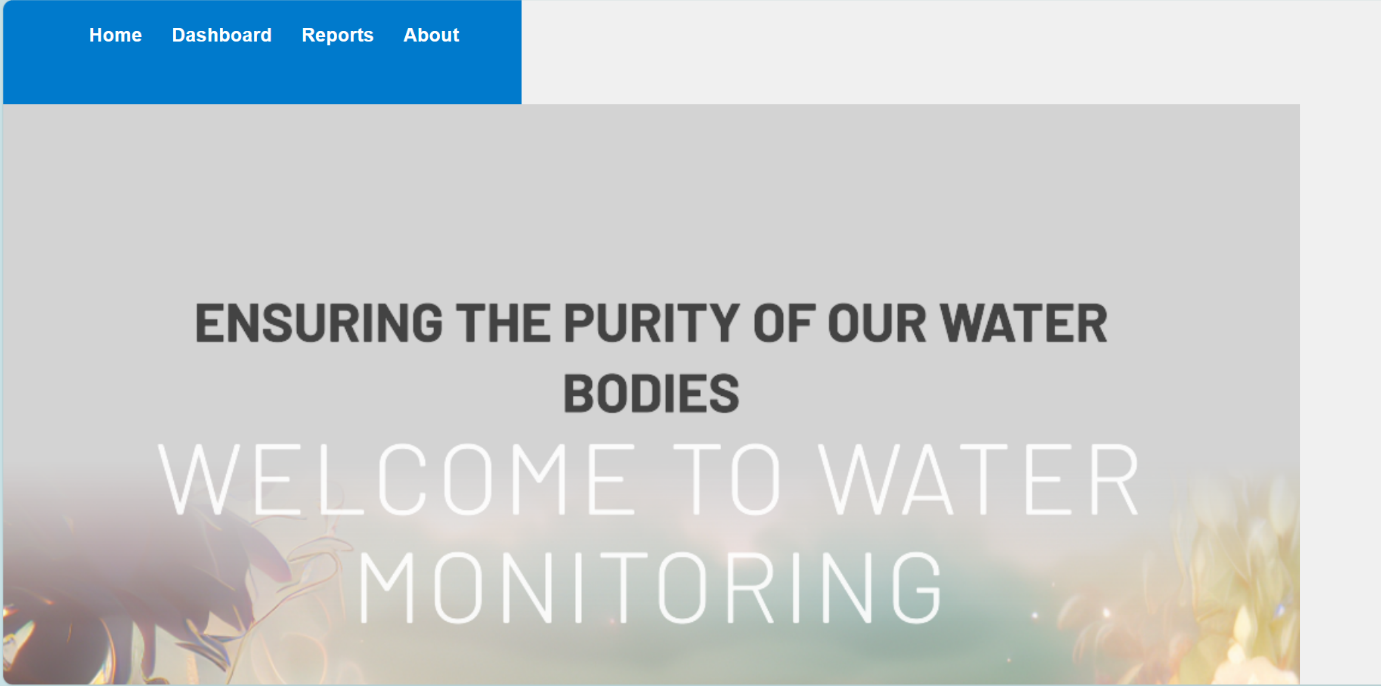
The design principles and considerations for the user interface

We designed the user interface using HTML, CSS, and JavaScript.

For a responsive and dynamic user interface, we used the React.js framework.

The design prioritizes a clean and intuitive layout for easy navigation.

Real-time data updates are displayed using Chart.js for data visualization.



# Implementation Phase

## Development Tools

The list of tools and technologies used for development, such as Node.js, Express, Mongoose, and additional libraries.

**Node.js:** Used for server-side development.

**Express.js:** Utilized to create a RESTful API for data communication.

**Mongoose:** ODM library for MongoDB database interaction.

**React.js:** Chosen for building the interactive and responsive user interface.

## Server-Side Implementation

The server-side implementation in Node.js, Using configuration of APIs for communication with IoT sensors and the user interface.

 // Node.js server setup

const express = require('express');

const app = express();

const port = 3000;

// Define API endpoints

app.get('/api/historical-data', (req, res) => {

    // Retrieve historical data from the MongoDB database and send it to the client.

    // Implement error handling as needed.

});

app.listen(port, () => {

    console.log(`Server is running on port ${port}`);

});

Connecting to the MongoDB database in Node.js server

// MongoDB connection setup

const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost:27017/water\_consumption', {

    useNewUrlParser: true,

    useUnifiedTopology: true,

})

    .then(() => {

        console.log('Connected to MongoDB');

    })

    .catch((err) => {

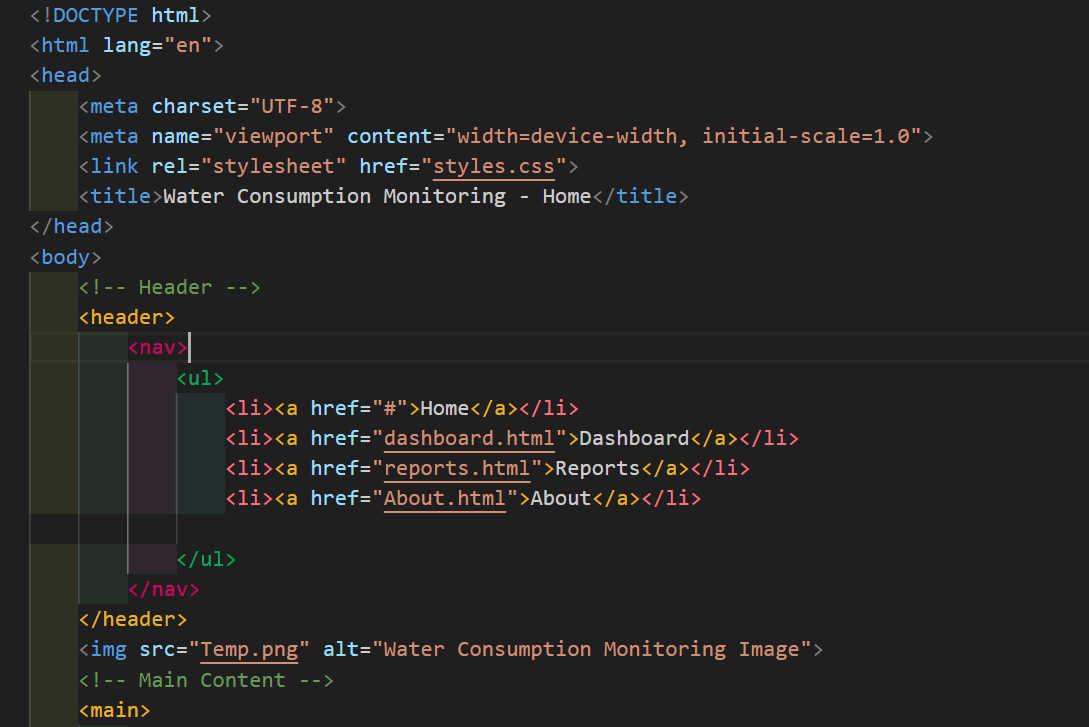
        console.error('MongoDB connection error:', err);

    });

## Front-End Implementation

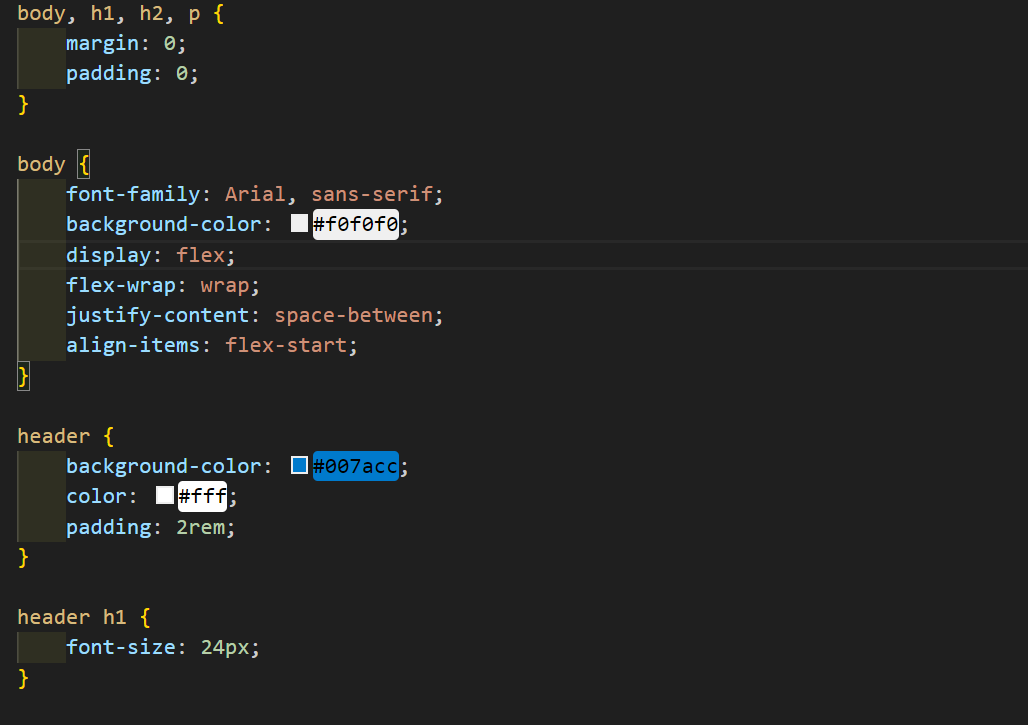
The development of the user interface, covering real-time data visualization, historical data charts, and interactive features.

Home.html

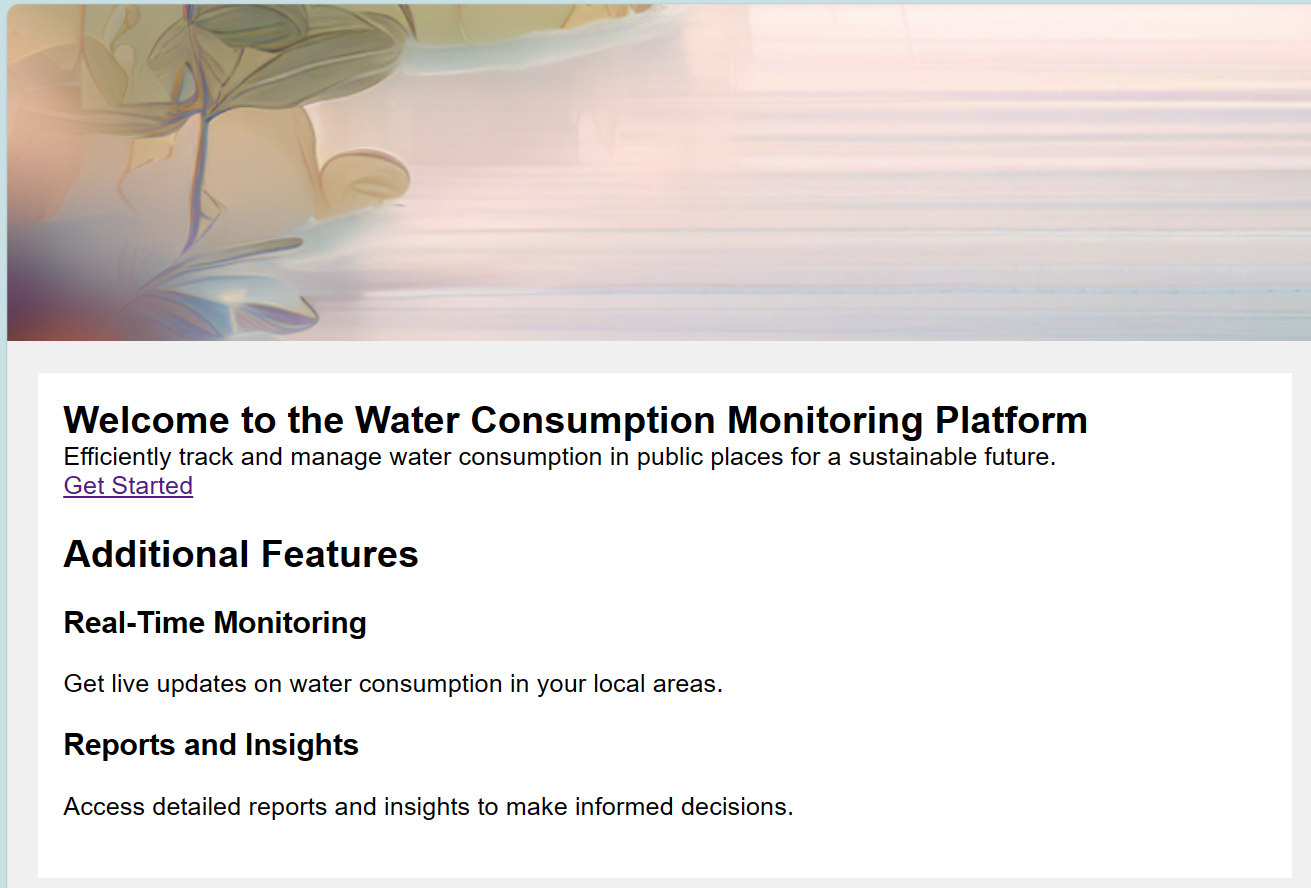


Stylesheet

The code snippet of our stylesheet



Our Homepage



Fetch and display real-time Water Consumption

// React component to fetch and display real-time water consumption data

import React, { useState, useEffect } from 'react';

function RealTimeData() {

    const [data, setData] = useState([]);

    useEffect(() => {

        //real-time data from the server using WebSocket

        // Update the state with the fetched data.

    }, []);

    return (

        <div>

            <h2>Real-Time Water Consumption Data</h2>

            <ul>

                {data.map((item, index) => (

                    <li key={index}>{item.location}: {item.consumption} liters</li>

                ))}

            </ul>

        </div>

    );

}

export default RealTimeData;

Real-Time Data Fetching Using WebSocket

// WebSocket client setup using the WebSocket API

const socket = new WebSocket('ws://your-server-url');

socket.addEventListener('open', (event) => {

    console.log('WebSocket connection opened.');

});

socket.addEventListener('message', (event) => {

    const data = JSON.parse(event.data);

    // Update the UI with real-time data received from the server.

});

socket.addEventListener('close', (event) => {

    console.log('WebSocket connection closed.');

});

# Testing and Quality Assurance

Describes how the website's functionality and performance by Unit Testing

Unit Testing using a testing framework like Mocha and Chai

// Using Mocha and Chai for testing

const chai = require('chai');

const expect = chai.expect;

const request = require('supertest');

const app = require('../app'); // Import your Express app

describe('API Endpoint Testing', () => {

    it('should return historical data', (done) => {

        request(app)

            .get('/api/historical-data')

            .expect(200)

            .end((err, res) => {

                if (err) return done(err);

                expect(res.body).to.be.an('array');

                done();

            });

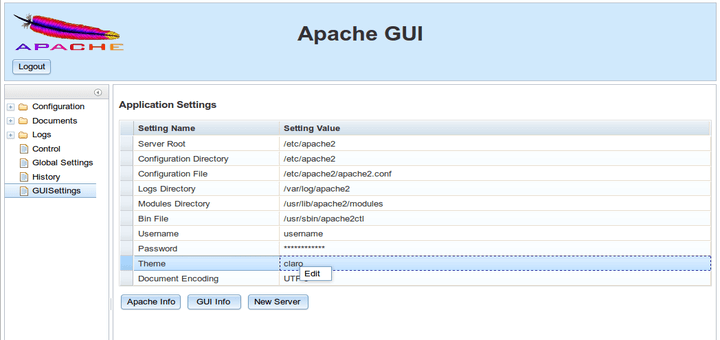
    });

});

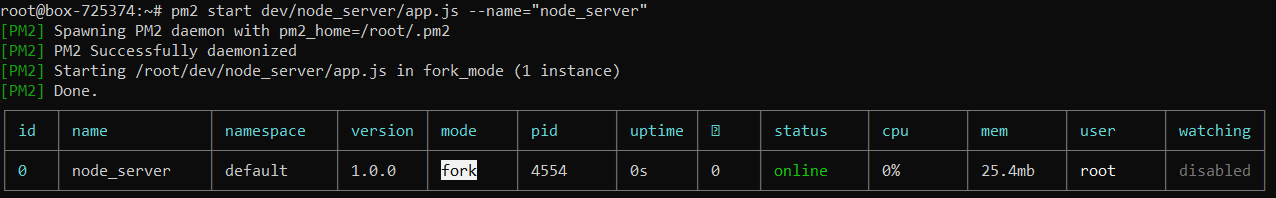
# Deployment

Steps for deploying the web, including setting up a web server and hosting

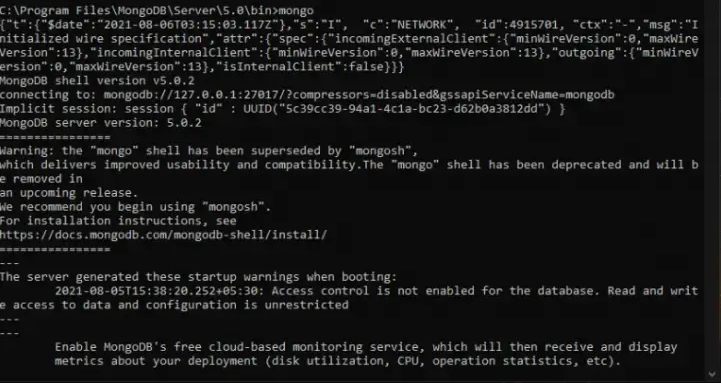
1. Set up the Apache web server, on hosting environment.



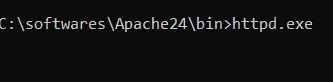
1. Configure the server to point to Node.js application.



1. Ensure that MongoDB is properly configured and running.

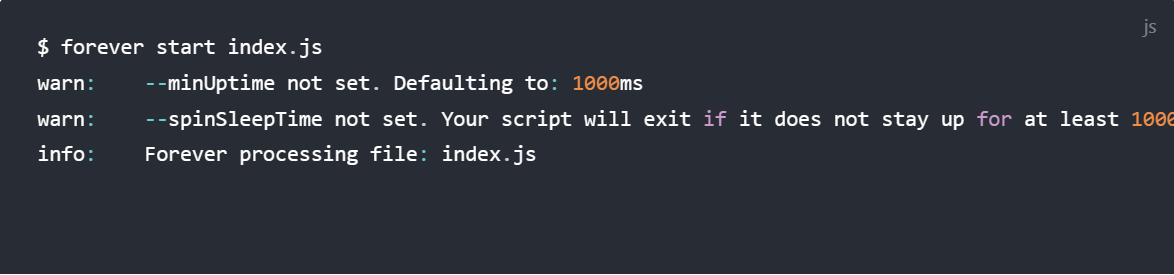


1. Deploy the web application code to the server.



5. Configure environment variables and settings for production.

6. Start the Node.js application using a process manager to keep it running.



7. Monitor server and application logs for any issues.

# Conclusion

This document outlines the design and implementation of the Water Consumption Monitoring website. It provides a comprehensive overview of the project's structure, components, and code. The goal of our team is to make the Water Consumption Monitoring website more accessible and maintainable So we will be able to gather and update real time data easily and efficiently, from the water management system.