Work Instruction

PoC/Working Demo

|  |  |
| --- | --- |
| **Author:** | **Praveen Kumar Madhava Rao** |
| **Owner:** | **Praveen Kumar Madhava Rao** |
| **Contributors:** |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **Version:** | **1.0** |
| **Date:** | **17/02/2024** |

**Table of Contents**

[**Preface** 3](#_Toc159348151)

[**Introduction:** 4](#_Toc159348152)

[**Prerequisites**: 4](#_Toc159348153)

[**Structure and layout:** 4](#_Toc159348154)

[**Handling User Interactions and Updating Visual Elements:** 4](#_Toc159348155)

[**Making Web Interface Responsive:** 5](#_Toc159348156)

[**Simulating EcoSync Data:** 5](#_Toc159348157)

[**Setup a Basic Backend Server:**  **Setup Node.js and Express.js:** 6](#_Toc159348158)

[**Create APIs for Simulating EcoSync Data** 7](#_Toc159348159)

[**Run the Backend Server:** 7](#_Toc159348160)

[**Simulate EcoSync Data:** 7](#_Toc159348161)

[**Update Route and Endpoint:** 8](#_Toc159348162)

[**Test the Integration:** 8](#_Toc159348163)

[**Interactivity with JavaScript:** 9](#_Toc159348164)

[**Adding Visual Elements:** 9](#_Toc159348165)

[Testing and Deployment: 10](#_Toc159348166)

[Testing: 10](#_Toc159348167)

[Deployment: 10](#_Toc159348168)

[Conclusion: 10](#_Toc159348169)

# **Preface**

The purpose of this document is to provide clear and concise instructions for performing the following:

* Integration: Integrate EcoSync with existing infrastructure for seamless data collection and analysis.
* Setup: Guide participants through the setup process of EcoSync within their environment.
* Customization: Demonstrate the customization options available in EcoSync for tailoring ESG metrics monitoring to specific organizational needs.
* Interactivity: Showcase the interactive features of EcoSync, allowing users to explore and interpret ESG data effectively.

# **Introduction:**

This work instruction offers step-by-step guidance for implementing EcoSync within your organization's infrastructure, facilitating real-time monitoring and management of Environmental, Social, and Governance (ESG) metrics. The objective is to empower organizations to make informed decisions, drive sustainability initiatives, and optimize environmental impact effectively.

# **Prerequisites**:

1. Basic Knowledge of HTML, CSS, and JavaScript:
2. Node.js and Express.js installed on the system.
3. Development environment set up with a new project directory.
4. Familiarity with simulating EcoSync data using mock values for ESG metrics.

# **Structure and layout:**

* Develop a basic web interface using HTML, CSS, and JavaScript.
* Create static visual elements like buttons, sliders, and charts to represent ESG metrics.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>EcoSync Demo</title>

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

</head>

<body>

<header>

<h1>EcoSync Demo</h1>

</header>

<main>

<section id="metrics">

<h2>ESG Metrics</h2>

<!-- Visual elements like charts or tables will be dynamically added here -->

</section>

<section id="controls">

<h2>Controls</h2>

<!-- User interaction elements like buttons or dropdown menus will be added here -->

</section>

</main>

<footer>

<p>&copy; 2024 EcoSync Inc.</p>

</footer>

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

</body>

</html>

<!

# **Handling User Interactions and Updating Visual Elements:**

JavaScript is used to handle user interactions and update visual elements dynamically. Here's an JavaScript code for handling a button click event and updating a visual element:

// Select the button element

const button = document.querySelector('#update-button');

// Add an event listener for the button click event

button.addEventListener('click', () => {

// Select the visual element to be updated

const visualElement = document.querySelector('#chart');

// Update the visual element

visualElement.textContent = 'Updated chart data';

});

**Making Web Interface Responsive:**CSS is used to make the web interface responsive by adjusting styles based on the device's screen size. Here's an example of CSS code for making the header responsive:

# Example suggestion messagerint("Suggestion: Consider optimizing the heavy\_cpu\_task function to reduce CPU load

header {

background-color: #333;

color: #fff;

padding: 10px;

text-align: center;

}

@media screen and (max-width: 600px) {

header {

font-size: 16px;

}

}

Top of Form

Bottom of Form

# **Simulating EcoSync Data:**

* You can simulate EcoSync data by generating random ESG metric values or using static data. Here's an example of JavaScript code for generating random metric values:

// Generate a random number between 0 and 100 for energy consumption

const energyConsumption = Math.floor(Math.random() \* 101);

// Generate a random number between 0 and 50 for carbon emissions

const carbonEmissions = Math.floor(Math.random() \* 51);

// Generate a random number between 0 and 200 for server utilization

const serverUtilization = Math.floor(Math.random() \* 201);

// Log the simulated metric values

console.log('Energy Consumption:', energyConsumption);

console.log('Carbon Emissions:', carbonEmissions);

console.log('Server Utilization:', serverUtilization);

# **Setup a Basic Backend Server:** **Setup Node.js and Express.js:**

* First, make sure you have Node.js installed on your computer. You can download it from the official website: [https://nodejs.org/](https://nodejs.org/" \t "_new)
* Once Node.js is installed, create a new directory for your project and navigate to it in your terminal or command prompt.
* Run the following command to initialize a new Node.js project and create a package.json file:  
   *npm init -y*
* Install Express.js by running the following command:  
  *npm install express*

**Create the Backend Server:**

* Create a new file named server.js in your project directory.
* Add the following code to server.js to set up a basic Express.js server:

// Import required modules

const express = require('express');

const app = express();

const port = 3000;

// Define routes

app.get('/', (req, res) => {

res.send('Welcome to the EcoSync API!');

});

// Start the server

app.listen(port, () => {

console.log(`Server is running on http://localhost:${port}`);

});

* This code sets up a basic Express.js server that listens on port 3000 and responds with "Welcome to the EcoSync API!" when you visit [http://localhost:3000](http://localhost:3000/" \t "_new) in your web browser.

# 

# **Create APIs for Simulating EcoSync Data**

* Add the following code to server.js to create simple APIs for simulating EcoSync data:

// Simulated ESG metrics data

const simulatedData = {

energyConsumption: Math.floor(Math.random() \* 101), // Random number between 0 and 100

carbonEmissions: Math.floor(Math.random() \* 51), // Random number between 0 and 50

serverUtilization: Math.floor(Math.random() \* 201) // Random number between 0 and 200

};

// API endpoint to get simulated ESG metrics data

app.get('/metrics', (req, res) => {

res.json(simulatedData);

});

* This code defines a route /metrics that returns simulated ESG metrics data when accessed. The data includes random values for energy consumption, carbon emissions, and server utilization.

# **Run the Backend Server:**

* Save your changes to server.js.
* Open your terminal or command prompt and navigate to your project directory.
* Run the following command to start the backend server: *node server.js*  
   *Your backend server is now running, and you can access the APIs you created by visiting <http://localhost:3000/metrics> in your web browser or making HTTP requests to that endpoint programmatically*.

# **Simulate EcoSync Data:**

* Modify the backend server to generate simulated EcoSync data in the same format as your actual ESG metrics.
* You can simulate data for metrics like energy consumption, carbon emissions, server utilization, etc., similar to what we did in the previous step.

# **Update Route and Endpoint:**

// Simulate EcoSync data

const ecoSyncData = {

energyConsumption: Math.floor(Math.random() \* 101), // Random number between 0 and 100

carbonEmissions: Math.floor(Math.random() \* 51), // Random number between 0 and 50

serverUtilization: Math.floor(Math.random() \* 201) // Random number between 0 and 200

};

// API endpoint to get simulated EcoSync data

app.get('/ecosync', (req, res) => {

res.json(ecoSyncData);

});

* Change the route and endpoint to reflect the EcoSync integration. For example, you can use **/ecosync** instead of **/metrics**.

// API endpoint to get simulated EcoSync data

app.get('/ecosync', (req, res) => {

res.json(ecoSyncData);

});

# **Test the Integration:**

* Start your backend server and test the integration by accessing the **/ecosync** endpoint in your browser or making HTTP requests to it programmatically.
* Verify that the simulated EcoSync data is being served correctly and matches the expected format.  
    
    
    
    
   **Basic Customization Options:**

$ node server.js

Server is running on http://localhost:3000

* + You can provide users with options to customize which ESG metrics they want to view or the time period for data visualization. For this example, let's add dropdown menus for selecting different metrics.

<section id="controls">

<h2>Controls</h2>

<label for="metric-select">Select Metric:</label>

<select id="metric-select">

<option value="energy">Energy Consumption</option>

<option value="carbon">Carbon Emissions</option>

<option value="utilization">Server Utilization</option>

</select>

</section>



## **Interactivity with JavaScript:**

* + You can use JavaScript to handle user interactions with the dropdown menus and update the visual elements (e.g., charts) accordingly.

// Select the dropdown menu element

const metricSelect = document.getElementById('metric-select');

// Add an event listener for changes to the dropdown menu

metricSelect.addEventListener('change', () => {

// Get the selected metric value

const selectedMetric = metricSelect.value;

// Call a function to update the visual elements based on the selected metric

updateVisualElements(selectedMetric);

});

// Function to update the visual elements based on the selected metric

function updateVisualElements(metric) {

// Depending on the selected metric, update the chart or other visual elements

switch (metric) {

case 'energy':

// Update chart for energy consumption

break;

case 'carbon':

// Update chart for carbon emissions

break;

case 'utilization':

// Update chart for server utilization

break;

default:

// Handle invalid metric selection

break;

}

}

# **Adding Visual Elements:**

You can create visual elements (e.g., charts) using libraries like Chart.js or D3.js and update them dynamically based on user selections.  
You can create visual elements (e.g., charts) using libraries like Chart.js or D3.js and update them dynamically based on user selections.

html

// Function to update the chart based on the selected metric

function updateChart(metric) {

// Create or update the chart based on the selected metric

// Example using Chart.js:

const ctx = document.getElementById('chart').getContext('2d');

const chart = new Chart(ctx, {

type: 'bar',

data: {

labels: ['January', 'February', 'March', 'April', 'May', 'June', 'July'],

datasets: [{

label: metric,

data: [10, 20, 30, 40, 50, 60, 70],

backgroundColor: 'rgba(255, 99, 132, 0.2)',

borderColor: 'rgba(255, 99, 132, 1)',

borderWidth: 1

}]

},

options: {

scales: {

y: {

beginAtZero: true

}

}

}

});

}

javascript

<section id="metrics">

<h2>ESG Metrics</h2>

<canvas id="chart"></canvas>

</section>

# Testing and Deployment:

# Testing:

* Perform thorough testing of the web application to validate its functionality and behavior.
* Check for any bugs or errors in the code and address them accordingly.
* Verify that user interactions, such as button clicks or dropdown selections, work as intended.
* Ensure that the web interface responds appropriately to different screen sizes and devices.
* Test the overall user experience to guarantee seamless navigation and usability.

Deployment:

* Once the testing phase is complete and all issues have been resolved, proceed with the deployment of the web application.
* Choose a suitable deployment environment, such as a local server or a hosting platform.
* For local deployment, set up a server environment on your own computer using tools like Node.js or Python's SimpleHTTPServer.
* Alternatively, deploy the application to a hosting platform such as GitHub Pages, Netlify, or Heroku for wider accessibility.
* Ensure that the deployment process is smooth and that the web application is accessible to users via the internet.

# Conclusion:

This work instruction provides a detailed guide for simulating integration with EcoSync, offering a glimpse into its capabilities for monitoring ESG metrics. By generating simulated data and serving it through the backend server, users can demonstrate the functionality of fetching ESG metrics data from EcoSync. While this demonstration serves as a foundational example, there is ample room for further customization and enhancement. Please feel free to explore additional features and improvements to advance the solution. Remember, this is only a sample solution design demonstrating the capabilities of EcoSync, and customization is encouraged to meet our project needs.