Here's a **simplified Asset Inventory Management System** project built using **HTML, CSS, JavaScript** for the front end, and **Python (Flask)** with **PostgreSQL** for the backend. This system will allow you to perform CRUD operations (Create, Read, Update, Delete) on asset details and generate reports on a daily, weekly, monthly, and yearly basis.

**Project Structure Overview**

1. **HTML**: Frontend form and data display.
2. **CSS**: Styling for the page.
3. **JavaScript**: Handles form submission and CRUD operations.
4. **Python (Flask)**: Backend logic to handle database operations.
5. **PostgreSQL**: Database to store asset details.
6. **CRUD Operations**: Create, Read, Update, Delete assets.
7. **Reporting**: Generate reports for asset management.

**1. Frontend (HTML)**

**HTML** page for asset management (with the form to add/update assets and view existing assets):

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Asset Inventory Management</title>

<link rel="stylesheet" href="static/style.css">

</head>

<body>

<div class="container">

<h1>Asset Inventory Management</h1>

<!-- Asset Form -->

<form id="assetForm">

<label for="assetId">Asset ID:</label>

<input type="text" id="assetId" name="assetId" required>

<label for="description">Description:</label>

<input type="text" id="description" name="description" required>

<label for="comments">Comments:</label>

<input type="text" id="comments" name="comments">

<label for="assetType">Asset Type:</label>

<input type="text" id="assetType" name="assetType" required>

<button type="submit">Add/Update Asset</button>

</form>

<!-- Display Asset List -->

<div id="assetList"></div>

<!-- Report Generation Button -->

<button id="generateReports">Generate Reports</button>

<div id="reportResults"></div>

</div>

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

</body>

</html>

### ****2. CSS Styling****

The **CSS** to style the page (stored in static/style.css):

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

}

.container {

max-width: 800px;

margin: 50px auto;

padding: 20px;

background-color: white;

border-radius: 10px;

box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);

}

h1 {

text-align: center;

color: #333;

}

form {

margin-bottom: 20px;

}

label {

display: block;

margin: 15px 0 5px;

}

input {

width: 100%;

padding: 10px;

margin-bottom: 10px;

border: 1px solid #ccc;

border-radius: 5px;

}

button {

padding: 10px 20px;

background-color: #28a745;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

}

button:hover {

background-color: #218838;

}

#assetList, #reportResults {

margin-top: 20px;

}

### ****3. JavaScript for Client-Side Logic****

**JavaScript** to handle form submissions and CRUD operations (stored in static/app.js):

document.getElementById('assetForm').addEventListener('submit', submitForm);

document.getElementById('generateReports').addEventListener('click', generateReports);

function submitForm(e) {

e.preventDefault();

const assetData = {

assetId: document.getElementById('assetId').value,

description: document.getElementById('description').value,

comments: document.getElementById('comments').value,

assetType: document.getElementById('assetType').value

};

fetch('/add\_asset', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify(assetData)

})

.then(res => res.json())

.then(data => {

alert(data.message);

document.getElementById('assetForm').reset();

fetchAssets();

})

.catch(err => console.error('Error:', err));

}

function fetchAssets() {

fetch('/assets')

.then(res => res.json())

.then(data => {

const assetList = document.getElementById('assetList');

assetList.innerHTML = '<h3>Assets List:</h3>';

data.forEach(asset => {

assetList.innerHTML += `<p>${asset.assetId}: ${asset.description} (${asset.assetType})</p>`;

});

})

.catch(err => console.error('Error:', err));

}

function generateReports() {

fetch('/reports')

.then(res => res.json())

.then(data => {

const reportResults = document.getElementById('reportResults');

reportResults.innerHTML = '<h3>Generated Reports:</h3>';

data.forEach(report => {

reportResults.innerHTML += `<p>${report}</p>`;

});

})

.catch(err => console.error('Error:', err));

}

// Load assets on page load

fetchAssets();

### ****4. Python Flask Backend****

**Flask (Python)** code to handle backend logic and communicate with the PostgreSQL database:

from flask import Flask, request, jsonify, render\_template

import psycopg2

app = Flask(\_\_name\_\_)

# Database connection

def get\_db\_connection():

conn = psycopg2.connect(

dbname='asset\_inventory',

user='your\_username',

password='your\_password',

host='localhost',

port='5432'

)

return conn

# Initialize database table if not exists

def create\_asset\_table():

conn = get\_db\_connection()

cur = conn.cursor()

cur.execute('''

CREATE TABLE IF NOT EXISTS assets (

id SERIAL PRIMARY KEY,

asset\_id VARCHAR(50) NOT NULL,

description TEXT NOT NULL,

comments TEXT,

asset\_type TEXT NOT NULL

);

''')

conn.commit()

cur.close()

conn.close()

@app.route('/')

def index():

return render\_template('index.html')

# Add or update asset

@app.route('/add\_asset', methods=['POST'])

def add\_asset():

data = request.get\_json()

asset\_id = data['assetId']

description = data['description']

comments = data['comments']

asset\_type = data['assetType']

conn = get\_db\_connection()

cur = conn.cursor()

cur.execute('SELECT \* FROM assets WHERE asset\_id = %s', (asset\_id,))

asset = cur.fetchone()

if asset:

# Update existing asset

cur.execute('''

UPDATE assets SET description = %s, comments = %s, asset\_type = %s WHERE asset\_id = %s

''', (description, comments, asset\_type, asset\_id))

else:

# Insert new asset

cur.execute('''

INSERT INTO assets (asset\_id, description, comments, asset\_type)

VALUES (%s, %s, %s, %s)

''', (asset\_id, description, comments, asset\_type))

conn.commit()

cur.close()

conn.close()

return jsonify({"message": "Asset added/updated successfully"})

# Get all assets

@app.route('/assets', methods=['GET'])

def get\_assets():

conn = get\_db\_connection()

cur = conn.cursor()

cur.execute('SELECT \* FROM assets')

assets = cur.fetchall()

asset\_list = []

for asset in assets:

asset\_list.append({

"assetId": asset[1],

"description": asset[2],

"comments": asset[3],

"assetType": asset[4]

})

cur.close()

conn.close()

return jsonify(asset\_list)

# Generate reports (simplified)

@app.route('/reports', methods=['GET'])

def generate\_reports():

reports = [

"Daily Report: 5 assets added.",

"Weekly Report: 20 assets updated.",

"Monthly Report: 100 assets in inventory.",

"Yearly Report: 1200 assets managed."

]

return jsonify(reports)

if \_\_name\_\_ == '\_\_main\_\_':

create\_asset\_table() # Ensure the table is created before the app runs

app.run(debug=True)

### ****5. PostgreSQL Setup****

Before running the Python app, make sure you have PostgreSQL installed and set up:

1. **Create the Database**:

CREATE DATABASE asset\_inventory;

2) **Create the Assets Table** (if needed):

CREATE TABLE assets (

id SERIAL PRIMARY KEY,

asset\_id VARCHAR(50) NOT NULL,

description TEXT NOT NULL,

comments TEXT,

asset\_type TEXT NOT NULL

);

**Project Directory Structure:-**

/asset-inventory

│

├── app.py

├── templates

│ └── index.html

├── static

│ ├── style.css

│ └── app.js

└── requirements.txt

### ****Running the Application****

1. Install required dependencies:

pip install Flask psycopg2

**2) Run the Flask application:**

**python app.py**

**3)** Open the application in a browser at <http://localhost:5000>.

### ****Conclusion****

This implementation provides a basic **Asset Inventory Management System** using **Flask (Python)**, **PostgreSQL**, and **HTML/CSS/JavaScript** for the frontend. The system supports CRUD operations for managing assets and provides simple reports. You can enhance it by adding more advanced features like search, asset categorization, or more complex reporting and analytics.