SIMATS ENGINEERING

Engineer to Excel

SIMATS

Saveetha Institute of Medical And Technical Sciences

(Declared as Deemed to be University under Section 3 of UGC Act 1956)

Department of Computer Science and Engineering

CSA4002 –Management Information Systems for Green Energy

MIS for Construction Project Management

Indhumathi V (192124215)

Date of Submission: 27-3-2025

Abstract

The project focuses on developing a Management Information System (MIS) for Construction Project Management to improve project tracking and resource allocation. The system tracks project timelines, material usage, and workforce allocation while generating real-time progress reports and budget tracking. It helps reduce project delays and improve cost variance to below 10%, enhancing overall resource management. The solution is implemented using Flask, Chart.js, and SQLite, providing an interactive dashboard for real-time data visualization and decision-making.

•**Problem:**

Inefficient project tracking and cost overruns.

•**Purpose:**

Develop a system to track project timelines, material usage, and workforce allocation.

•**Outcome:**

Improved resource management, reduced project delays, and real-time budget tracking.

Table of Contents

Introduction

- **Background:**
  Construction projects face delays and cost overruns due to poor tracking.
- **Objective:**
  Build an MIS to track and manage project progress.
- **Significance:**
  Enhance efficiency and cost control.
- **Scope:**
  10+ projects with real-time monitoring.
- **Methodology:**
  Flask (Python), Chart.js, HTML, CSS, and SQLite.

Problem Identification and Analysis

- **Problem:**
  Poor resource allocation and budget overruns.
- **Evidence:**
  High variance (>10%) in budget and schedule slippage.
- **Stakeholders:**
  Project Managers, Engineers, Workforce.
- **Findings:**
  Lack of real-time visibility in project progress and costs.

Solution Design and Implementation

•**Design:**

Flask-based web app with data visualization using Chart.js.

•**Tools:**

Flask, SQLite, Chart.js, HTML, CSS, JavaScript.

•**Solution:**

- Real-time data entry
- Dynamic chart updates
- Resource and budget prediction

Results and Recommendations

•**Results:**
•
✓▯ Improved resource allocation
✓▯ Reduced project delays (<10%)
✓▯ Accurate real-time budget tracking

•**Challenges:**
Data sync issues during updates.

•**Recommendations:**
Improve prediction model accuracy and UI.

CODING-1

1. **Backend (Flask)** – app.py

 Developed using Flask framework.Handles API requests for data fetching, updating, and processing.Manages CSV-based project data storage and updates in real-time.Key

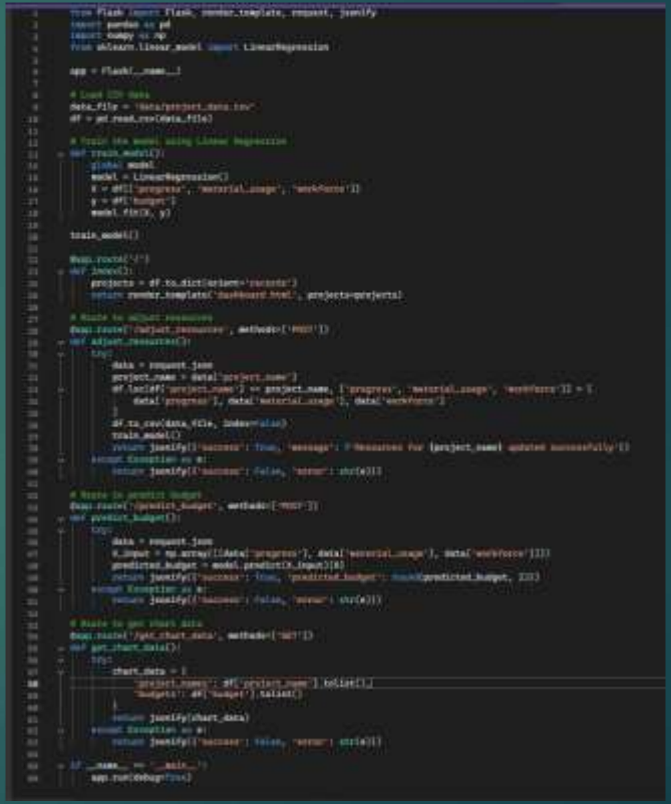
Functionalities:

✓ Load project data from CSV.

✓ Adjust resources based on user input.

✓ Predict budget using dynamic calculations.

✓ Return real-time data for chart updates.



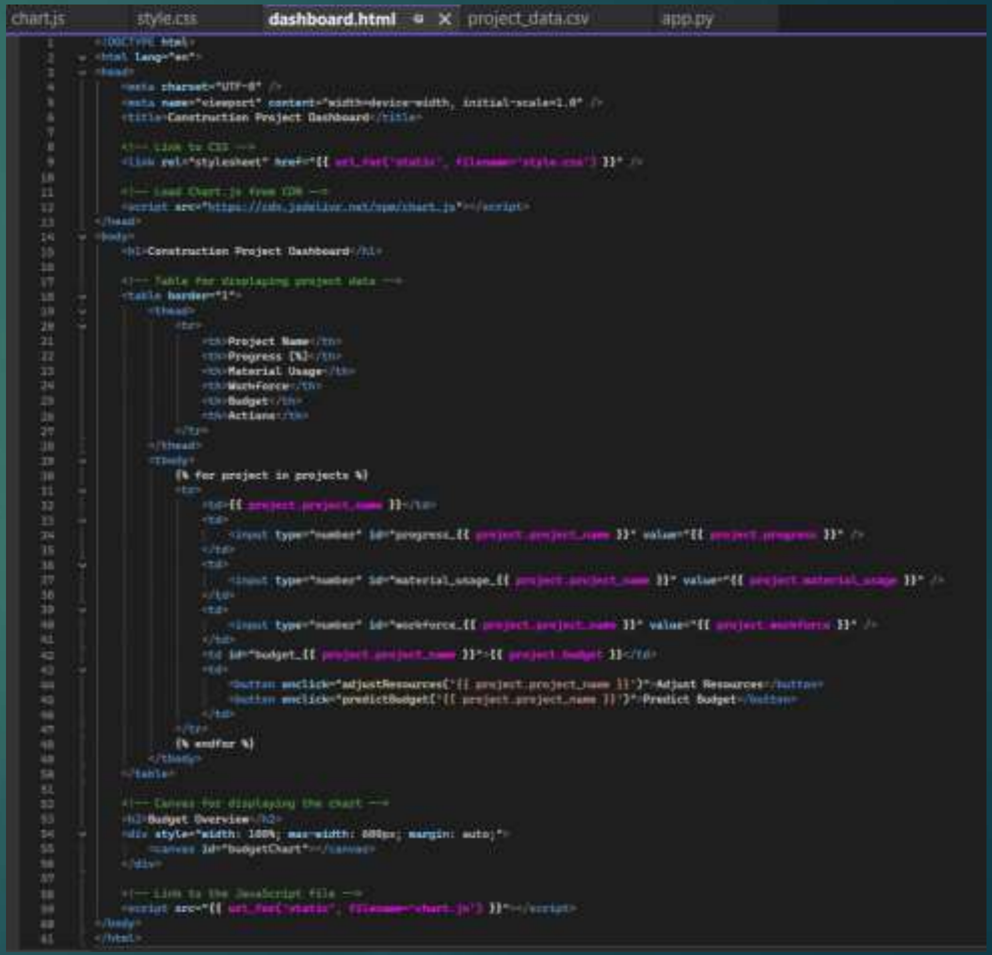
```python
from flask import Flask, render_template, request, jsonify
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression

app = Flask(__name__)

# Load CSV data
data_file = 'data/project_data.csv'
df = pd.read_csv(data_file)

# Train the model using Linear Regression
def train_model():
    global model
    model = LinearRegression()
    X = df[['progress', 'material_usage', 'workforce']]
    y = df['budget']
    model.fit(X, y)

train_model()

@app.route('/')
def index():
    projects = df.to_dict(orient='records')
    return render_template('dashboard.html', projects=projects)

# Route to adjust resources
@app.route('/adjust_resources', methods=['POST'])
def adjust_resources():
    try:
        data = request.json
        project_name = data['project_name']
        df.loc[df['project_name'] == project_name, ['progress', 'material_usage', 'workforce']] = [
            data['progress'], data['material_usage'], data['workforce']
        ]
        df.to_csv(data_file, index=False)
        train_model()
        return jsonify({'success': True, 'message': f'Resources for {project_name} updated successfully'})
    except Exception as e:
        return jsonify({'success': False, 'error': str(e)})

# Route to predict budget
@app.route('/predict_budget', methods=['POST'])
def predict_budget():
    try:
        data = request.json
        X_input = np.array([[data['progress'], data['material_usage'], data['workforce']]])
        predicted_budget = model.predict(X_input)[0]
        return jsonify({'success': True, 'predicted_budget': round(predicted_budget, 2)})
    except Exception as e:
        return jsonify({'success': False, 'error': str(e)})

# Route to get chart data
@app.route('/get_chart_data', methods=['GET'])
def get_chart_data():
    try:
        chart_data = {
            'project_names': df['project_name'].tolist(),
            'budgets': df['budget'].tolist()
        }
        return jsonify(chart_data)
    except Exception as e:
        return jsonify({'success': False, 'error': str(e)})

if __name__ == '__main__':
    app.run(debug=True)
```

CODING-2

2. Frontend – dashboard.html
Built with HTML + CSS + JavaScript.Uses Chart.js for real-time chart updates.Includes user interaction for adjusting data and predicting budget.

Key Features:

✓ Editable table for real-time input.
✓ Action buttons to adjust resources and predict budget.
✓ Real-time data updates reflected in the chart.



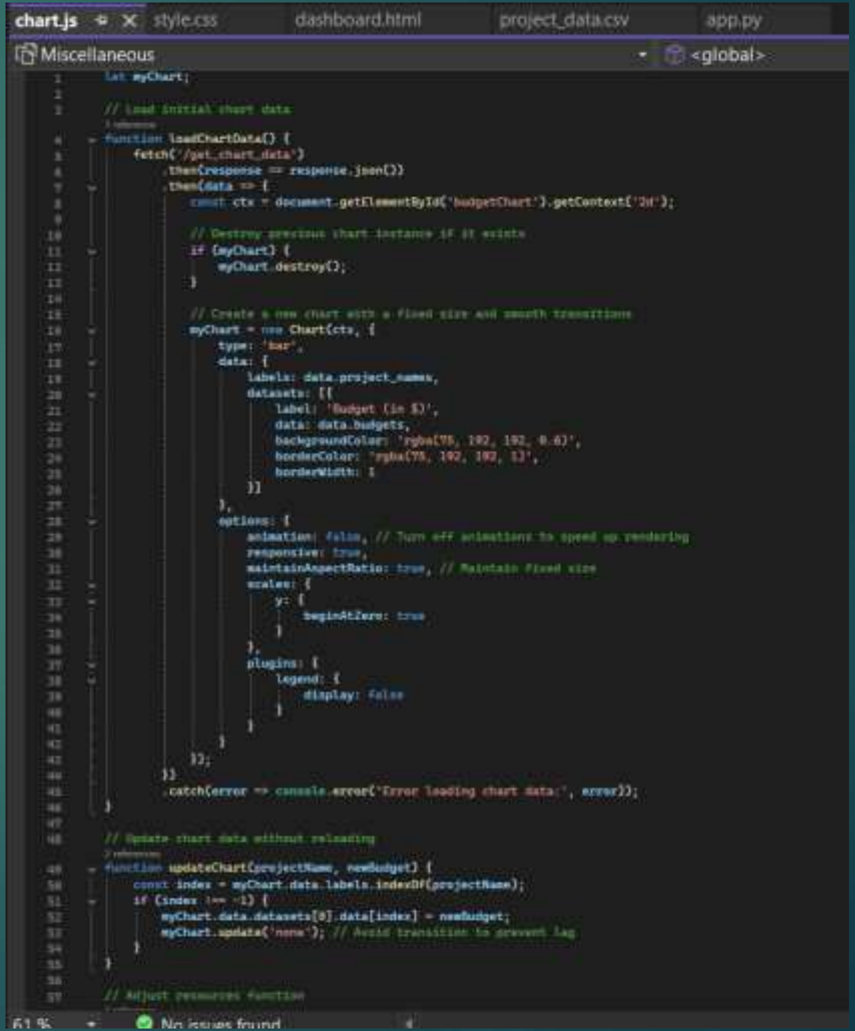
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Construction Project Dashboard</title>

    <!-- Link to CSS -->
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}" />

    <!-- Load Chart.js from CDN -->
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
    <h1>Construction Project Dashboard</h1>

    <!-- Table for displaying project data -->
    <table border="1">
        <thead>
            <tr>
                <th>Project Name</th>
                <th>Progress (%)</th>
                <th>Material Usage</th>
                <th>WorkForce</th>
                <th>Budget</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody>
            {% for project in projects %}
            <tr>
                <td>{{ project.project_name }}</td>
                <td>
                    <input type="number" id="progress_{{ project.project_name }}" value="{{ project.progress }}" />
                </td>
                <td>
                    <input type="number" id="material_usage_{{ project.project_name }}" value="{{ project.material_usage }}" />
                </td>
                <td>
                    <input type="number" id="workforce_{{ project.project_name }}" value="{{ project.workforce }}" />
                </td>
                <td id="budget_{{ project.project_name }}">{{ project.budget }}</td>
                <td>
                    <button onclick="adjustResources('{{ project.project_name }}')">Adjust Resources</button>
                    <button onclick="predictBudget('{{ project.project_name }}')">Predict Budget</button>
                </td>
            </tr>
            {% endfor %}
        </tbody>
    </table>

    <!-- Canvas for displaying the chart -->
    <h2>Budget Overview</h2>
    <div style="width: 100%; max-width: 800px; margin: auto;">
        <canvas id="budgetChart"></canvas>
    </div>

    <!-- Link to the JavaScript file -->
    <script src="{{ url_for('static', filename='chart.js') }}"></script>
</body>
</html>
```

CODING-3

3. Chart Integration – chart.js
Uses Chart.js for real-time visualization.Dynamically updates when table data is adjusted.

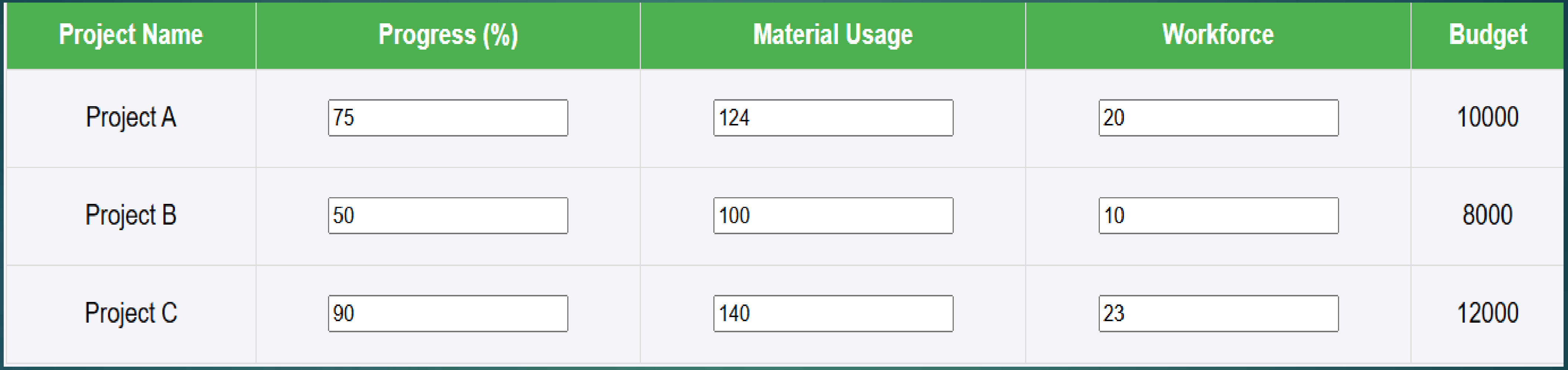
Key Functionalities:

✓ Load data from Flask backend
✓ Update chart directly without reload
✓ Smooth transition for real-time effect



```javascript
let myChart;

// Load initial chart data
function loadChartData() {
    fetch('/get_chart_data')
        .then(response => response.json())
        .then(data => {
            const ctx = document.getElementById('budgetChart').getContext('2d');

            // Destroy previous chart instance if it exists
            if (myChart) {
                myChart.destroy();
            }

            // Create a new chart with a fixed size and smooth transitions
            myChart = new Chart(ctx, {
                type: 'bar',
                data: {
                    labels: data.project_names,
                    datasets: [{
                        label: 'Budget (in $)',
                        data: data.budgets,
                        backgroundColor: 'rgba(75, 192, 192, 0.6)',
                        borderColor: 'rgba(75, 192, 192, 1)',
                        borderWidth: 1
                    }]
                },
                options: {
                    animation: false, // Turn off animations to speed up rendering
                    responsive: true,
                    maintainAspectRatio: true, // Maintain fixed size
                    scales: {
                        y: {
                            beginAtZero: true
                        }
                    },
                    plugins: {
                        legend: {
                            display: false
                        }
                    }
                }
            });
        })
        .catch(error => console.error('Error loading chart data:', error));
}

// Update chart data without reloading
function updateChart(projectName, newBudget) {
    const index = myChart.data.labels.indexOf(projectName);
    if (index !== -1) {
        myChart.data.datasets[0].data[index] = newBudget;
        myChart.update('none'); // Avoid transition to prevent lag
    }
}

// Adjust resources function
```

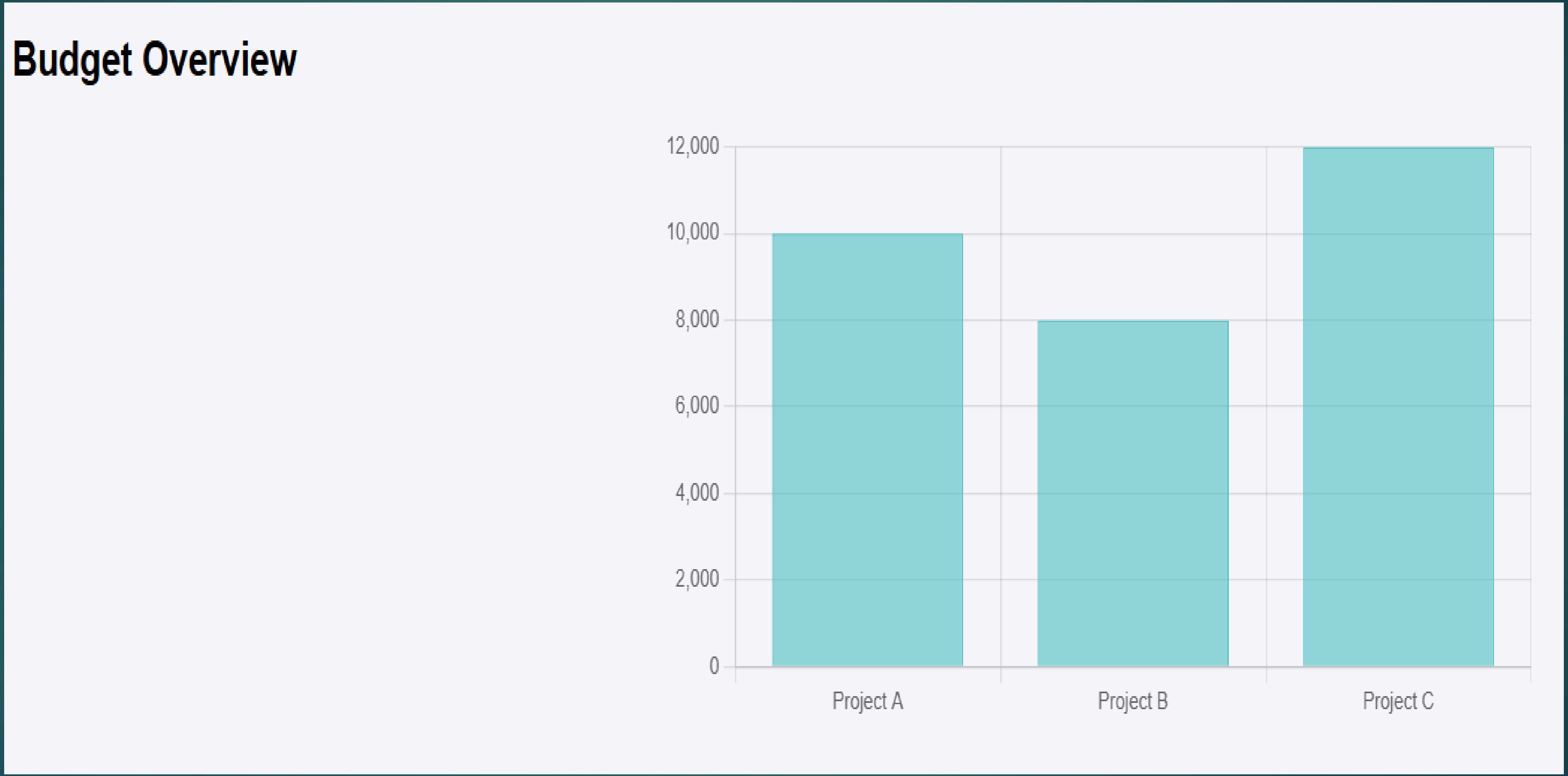
OUTPUT
DASHBOARD OVERVIEW



Project Name	Progress (%)	Material Usage	Workforce	Budget
Project A	75	124	20	10000
Project B	50	100	10	8000
Project C	90	140	23	12000

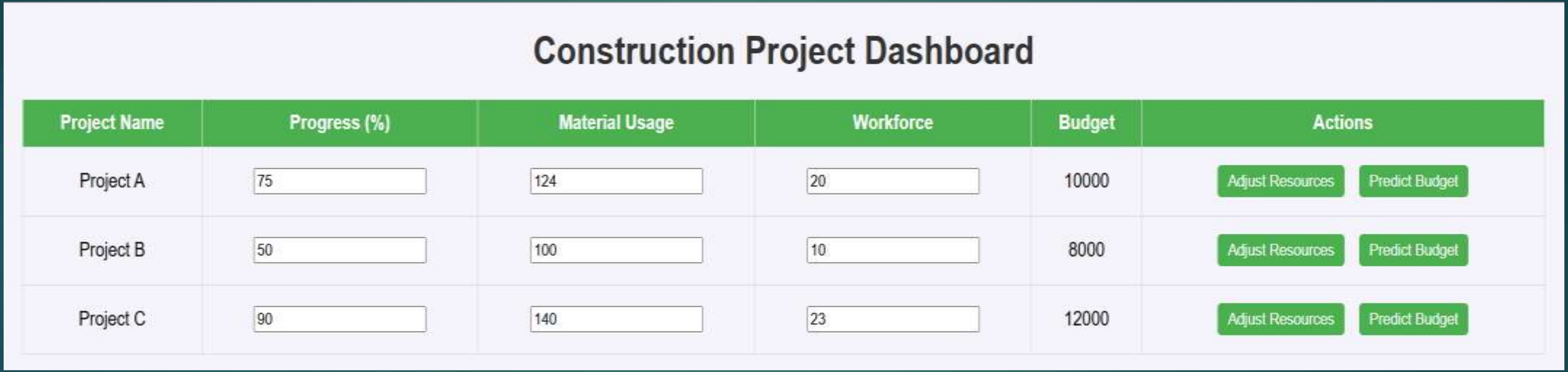
OUTPUT

CHART



OUTPUT

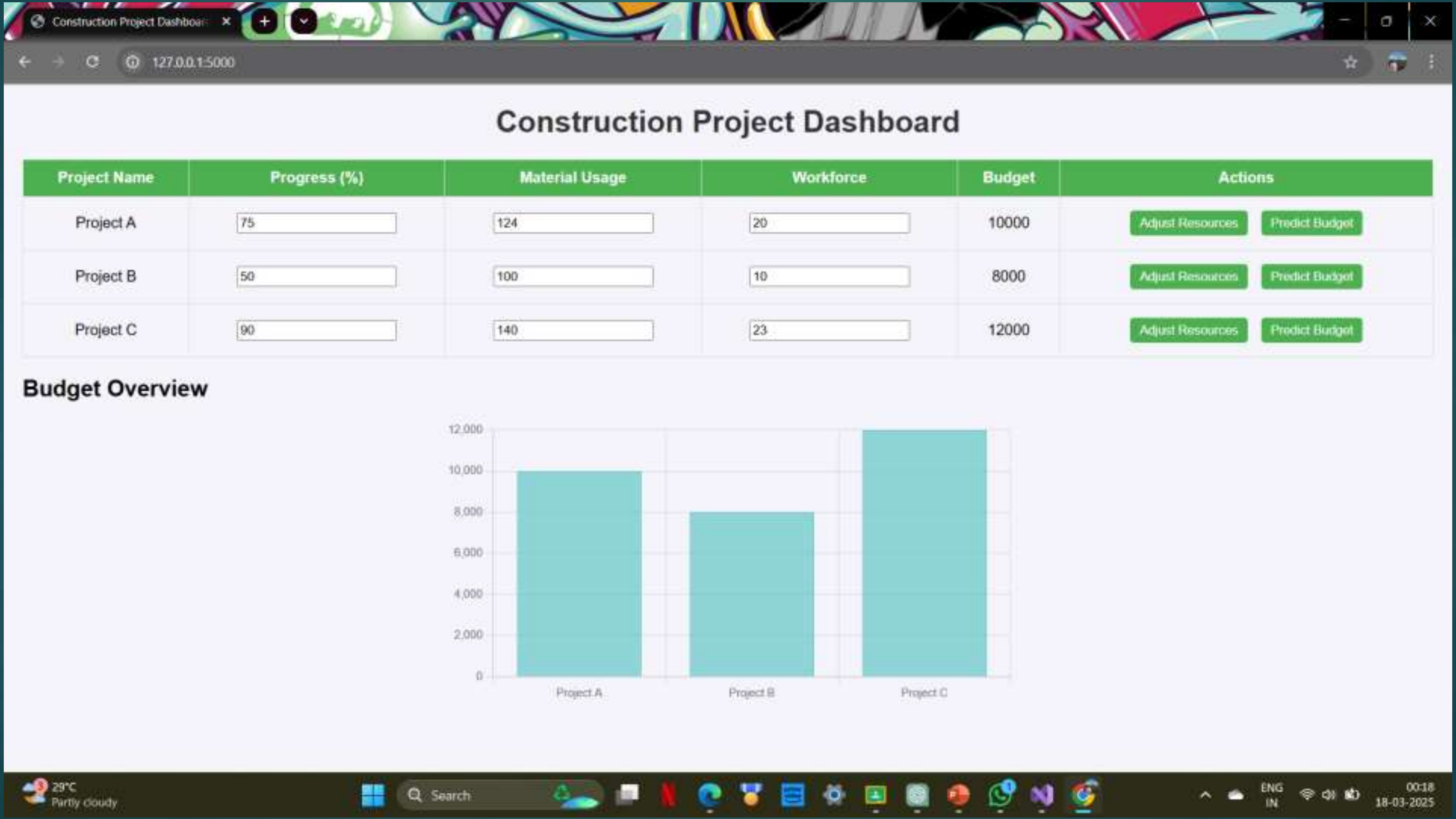
ADJUSTMENT



Construction Project Dashboard

Project Name	Progress (%)	Material Usage	Workforce	Budget	Actions
Project A	75	124	20	10000	Adjust Resources Predict Budget
Project B	50	100	10	8000	Adjust Resources Predict Budget
Project C	90	140	23	12000	Adjust Resources Predict Budget

OUTPUT
FINAL



Construction Project Dashboard

Project Name	Progress (%)	Material Usage	Workforce	Budget	Actions
Project A	75	124	20	10000	Adjust Resources Predict Budget
Project B	50	100	10	8000	Adjust Resources Predict Budget
Project C	90	140	23	12000	Adjust Resources Predict Budget

Budget Overview

Reflection on Learning and Development

- **Learning:**
  Flask, REST API, Chart.js, and real-time updates.
- **Challenges:**
  Chart re-rendering and API sync.
- **Skills:**
  Backend and frontend integration, data visualization.
- **Industry Insight:**
  Importance of real-time MIS for large-scale projects.

Conclusion

•**Findings:**
Real-time MIS improved cost tracking and project efficiency.

•**Importance:**
Enhanced decision-making for project managers.

•**Future:**
Add AI-based predictions and automated resource adjustment.

References

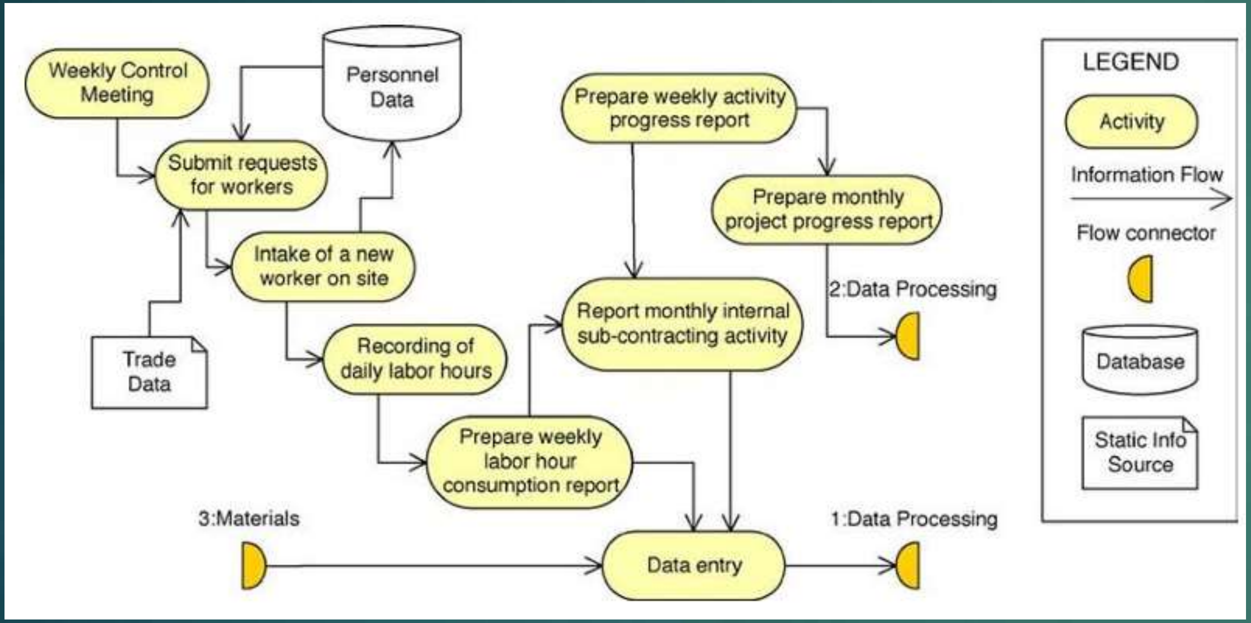
•Ceballos, C., Kolozhvari, A. A., Dolbilov, A. G., Semenov, R. N., Tsapulina, E. A., Rodriguez, A., ... & Murin, Y. A. (2023). Construction management information system at JINR. *Physics of Particles and Nuclei Letters*, *20*(5), 981-987.

•Lee, S. K., & Yu, J. H. (2012). Success model of project management information system in construction. *Automation in construction*, *25*, 82-93.

•Scherer, R. J., & Schapke, S. E. (2011). A distributed multi-model-based management information system for simulation and decision-making on construction projects. *Advanced Engineering Informatics*, *25*(4), 582-599.

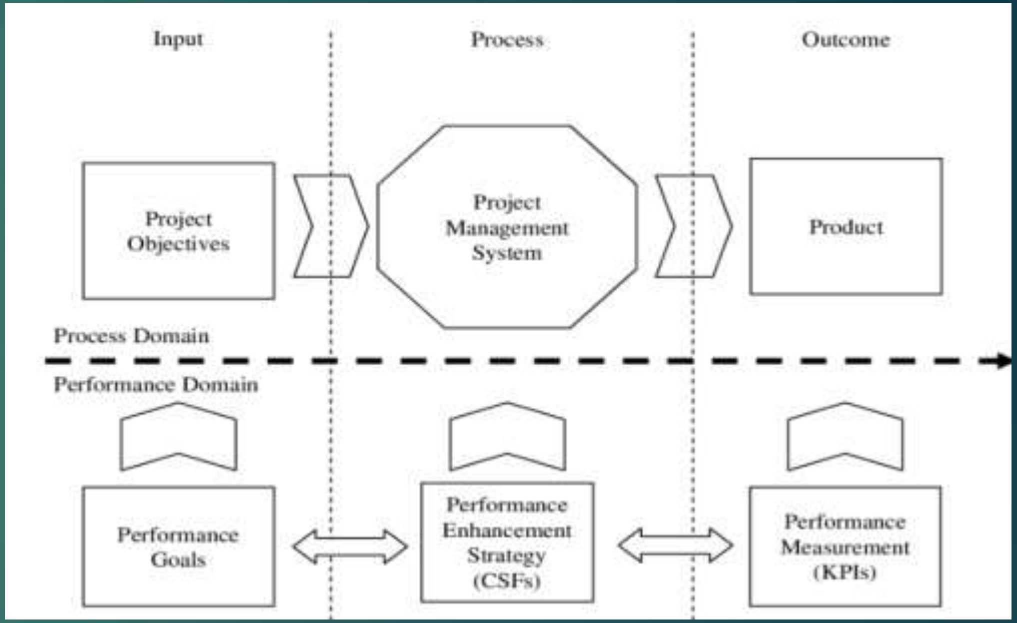
Appendices

System architecture diagram



Weekly Control Meeting
Personnel Data
Submit requests for workers
Intake of a new worker on site
Trade Data
Recording of daily labor hours
Prepare weekly labor hour consumption report
Prepare weekly activity progress report
Prepare monthly project progress report
Report monthly internal sub-contracting activity
2:Data Processing
3:Materials
Data entry
1:Data Processing
LEGEND
Activity
Information Flow
Flow connector
Database
Static Info Source

Data Flow Diagram



Input
Process
Outcome
Project Objectives
Project Management System
Product
Process Domain
Performance Domain
Performance Goals
Performance Enhancement Strategy (CSFs)
Performance Measurement (KPIs)