**DATABASE SYSTEMS**

**PROJECT**

**UMER IMRAN (22I-1891)**

**M.SULEMAN (22I-1931)**

**DS-B**

**TITLE:**

**The Corporate Vendor - Contract Management System**

**REPORT:**

**1. Introduction**

The "Corporate Vendor Management System" is a comprehensive web-based platform designed to manage vendor registration, compliance tracking, and contract performance for businesses. This system aims to streamline the vendor lifecycle management process, improve efficiency, and ensure regulatory compliance while offering an intuitive interface for administrators and vendors.

The project combines front-end, back-end, and database management to provide a seamless experience for users and administrators alike.

**2. Purpose / Objectives:**

The main purpose of this project is to create a system to help large organizations manage vendors and contracts efficiently**.** It is like building a digital tool that helps track vendors, contracts, purchase orders, budgets, and performance evaluations. Here's what it aims to achieve:

1. **Organized Vendor Management**: Keep all vendor details in one place, including their performance and compliance with company standards.
2. **Streamlined Contract Handling**: Simplify creating, negotiating, renewing, and tracking contracts, so nothing is missed or delayed.
3. **Efficient Purchase Order Tracking**: Allow departments to create and monitor purchase orders while staying within budgets.
4. **Budget Monitoring**: Ensure spending is controlled and within limits by tracking all expenses related to vendors and contracts.
5. **Performance Evaluation**: Regularly review and rate vendors based on quality, delivery, and other metrics to make informed decisions.
6. **Automation and Notifications**: Set up alerts for contract renewals or budget issues to ensure no deadlines or limits are overlooked.

#### ****3. System Architecture****

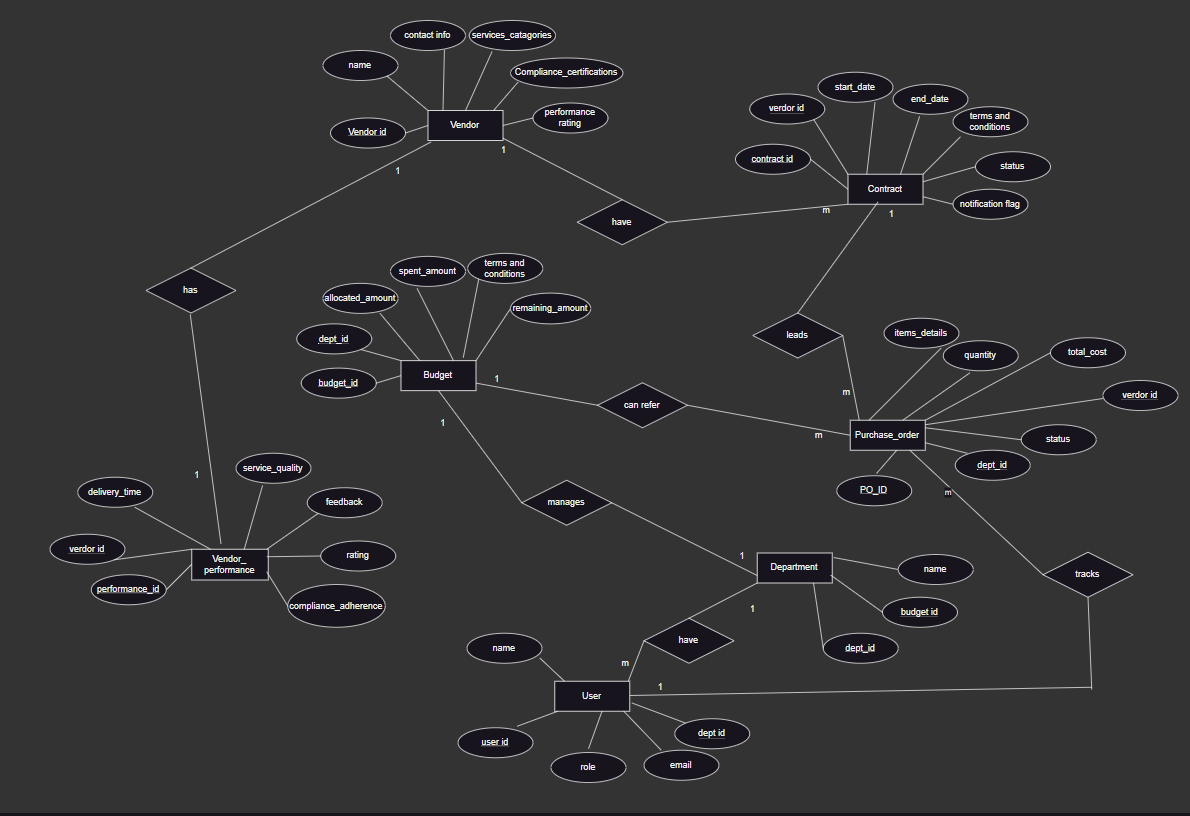
The project follows a **three-tier architecture** comprising:

1. **Presentation Layer (Front-End):**
   * Technologies Used: HTML5, CSS3, JavaScript
   * Purpose: Build user-friendly interfaces for vendors and administrators.
   * Key Components:
     + **Home Page:** Provides navigation options for vendor registration and login.
     + **Vendor Registration Form:** Captures details such as name, contact information, service categories, certifications, and performance ratings.
     + **Login Page:** Enables vendors to log in using their registered email.
     + **Contract page, Budget Page, Purchase Order page, Vendor Performance report pages.**
2. **Business Logic Layer (Back-End):**
   * Framework Used: Node.js with Express.js
   * Purpose: Handle HTTP requests, route data, and communicate with the database.
   * Key Features:
     + Vendor registration and login functionalities.
     + Validation and error handling for database queries.
     + Middleware for parsing and managing form data.
3. **Data Layer (Database):**
   * Database Used: MySQL
   * Purpose: Store vendor information, ensuring data consistency and availability.
   * Key Tables:
     + **Vendor Table:** Stores vendor details such as Name, Contact Info, Service Categories, Compliance Certifications, and Performance Ratings.

#### ****4. Technologies Used****

1. **Front-End:**
   * **HTML5:** For structuring web pages.
   * **CSS3:** For styling and responsive design.
   * **JavaScript:** For client-side interactivity.
2. **Back-End:**
   * **Node.js:** Provides the server environment.
   * **Express.js:** Manages server-side routing and middleware.
3. **Database:**
   * **MySQL:** Used for data storage and retrieval.
4. **Development Tools:**
   * Code Editor: Visual Studio Code
   * Database Client: MySQL Workbench
   * Dependency Manager: npm (Node Package Manager)

**5. ER DIAGRAM:**

****

**6. Schema Diagram (written and using reverse engineer):**

**1. Vendor**  
(**Vendor\_ID**, Name, Contact\_Info, Service\_Categories, Compliance\_Certifications, Performance\_Rating)

**2. Contract**  
(**Contract\_ID**, Vendor\_ID, Start\_Date, End\_Date, Terms\_and\_Conditions, Status, Renewal\_Notification\_Flag)

**3.Budget**  
(**Budget\_ID**, Vendor\_ID, Allocated\_Amount, Spent\_Amount, Remaining\_Amount)

**4. Purchase\_Order**  
(**PO\_ID**, Vendor\_ID, Items\_Details, Quantity, Total\_Cost, Status)

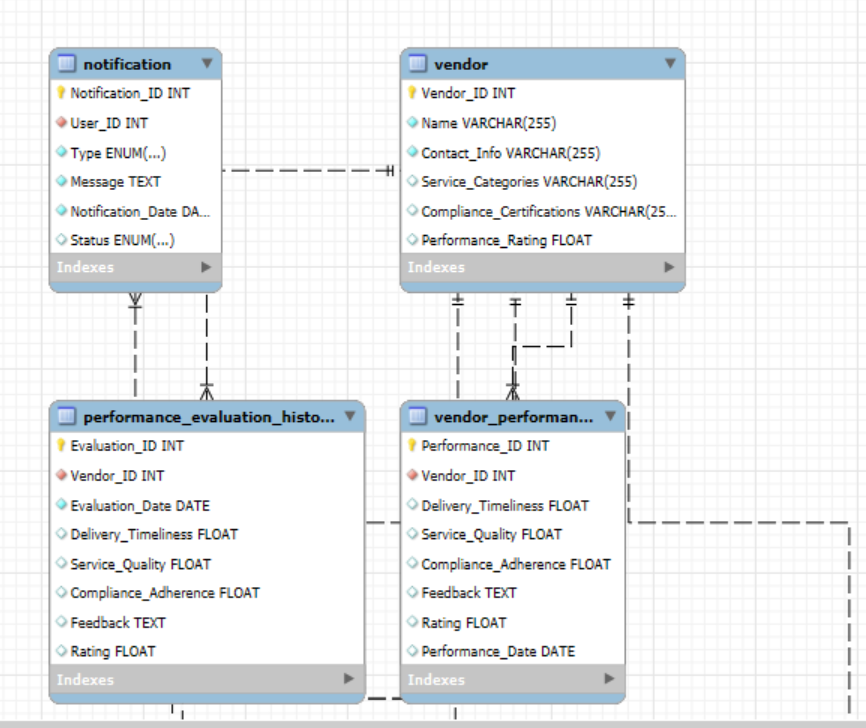
**5. Vendor\_Performance**  
(**Performance\_ID**, Vendor\_ID, Delivery\_Timeliness, Service\_Quality, Compliance\_Adherence, Feedback, Rating)

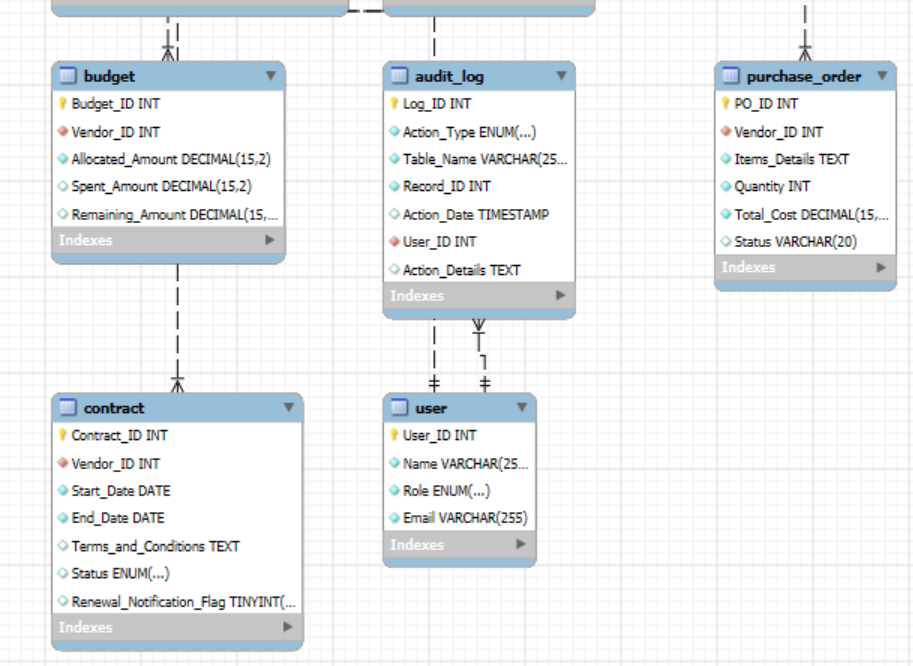
**6. User**  
(**User\_ID**, Name, Role, Email)

**7. Performance\_Evaluation\_History**  
(**Evaluation\_ID**, Vendor\_ID, Evaluation\_Date, Delivery\_Timeliness, Service\_Quality, Compliance\_Adherence, Feedback, Rating)

**8. Notification**  
(**Notification\_ID**, User\_ID, Type, Message, Notification\_Date, Status)

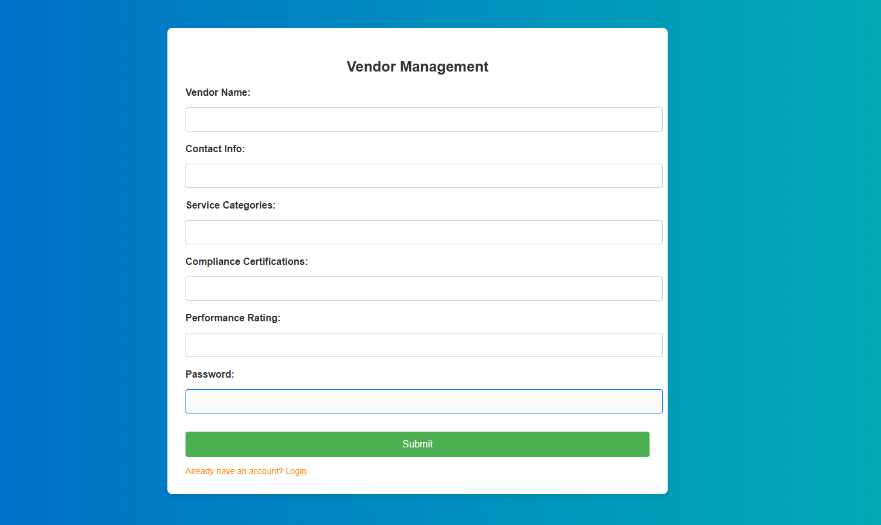
**9. Audit\_Log**  
(**Log\_ID**, Action\_Type, Table\_Name, Record\_ID, Action\_Date, User\_ID, Action\_Details)

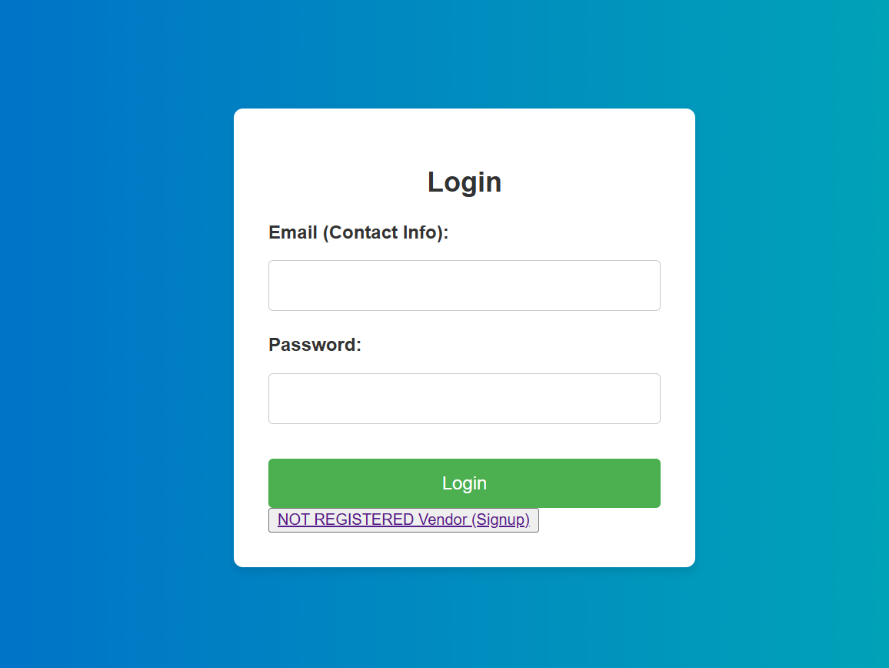


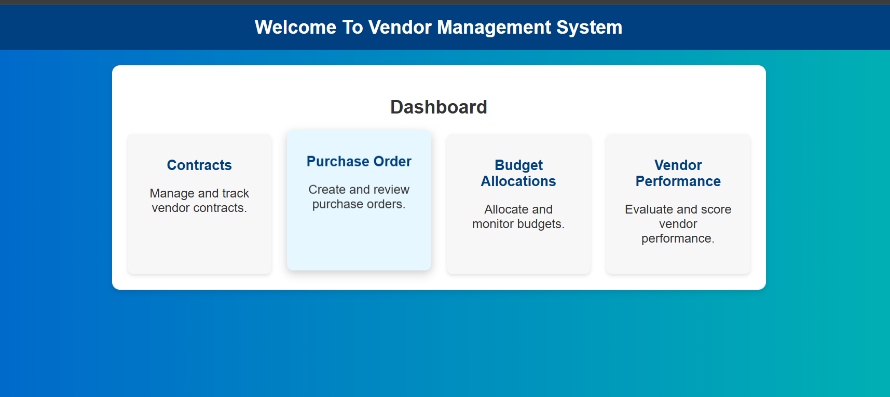


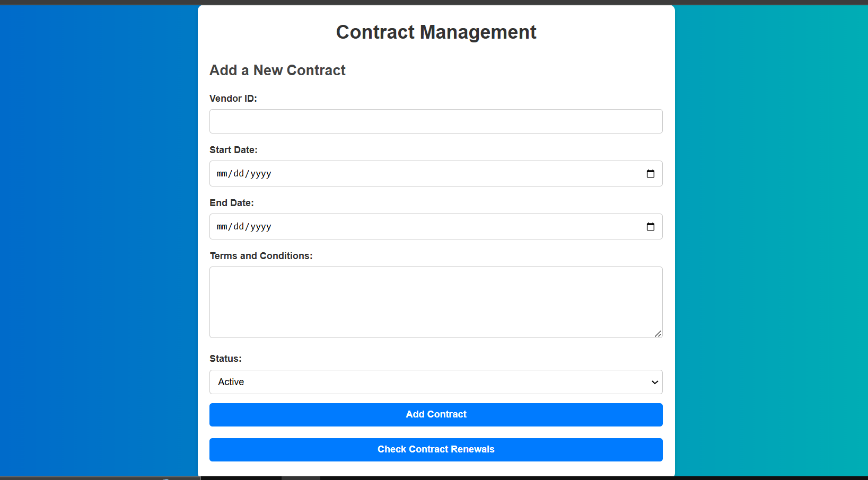
**7. Front end screens shots:**

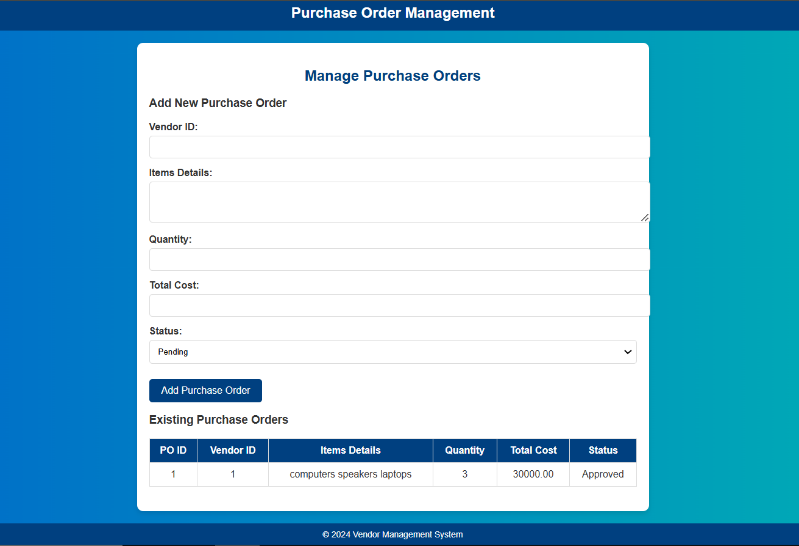


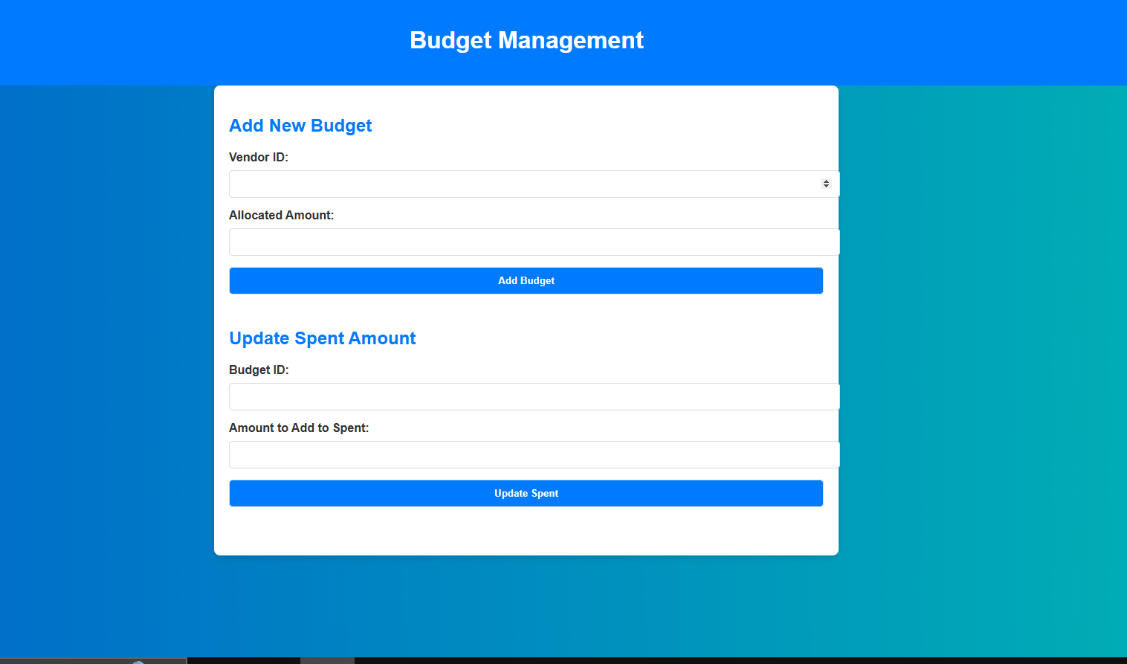
****

**­ **

****

****

****

****

****

**8. Back end codes (node js /Express) (key steps):**

* + **INDEX.JS**

**The index.js is made for making the connection with the node js and the sql workbench:**

// Import required modules

var mysql = require("mysql2");

// Establish MySQL connection

var connection = mysql.createConnection({

    host: 'localhost',

    user: 'root',

    password: '12345678',

    multipleStatements: true // Allow multiple SQL statements

});

**Further the DDL scripts for database creation and the tables creation the DDL scripts was also written in index.js.**

// DDL Script to create database, tables, and procedures

var ddlScript = `

CREATE DATABASE IF NOT EXISTS CorporateVendorManagement;

USE CorporateVendorManagement;

CREATE TABLE IF NOT EXISTS Vendor (

    Vendor\_ID INT AUTO\_INCREMENT PRIMARY KEY,

    Name VARCHAR(255) NOT NULL,

    Contact\_Info VARCHAR(255) NOT NULL,

    Service\_Categories VARCHAR(255),

    Compliance\_Certifications VARCHAR(255),

    Performance\_Rating FLOAT DEFAULT 0

);

CREATE TABLE IF NOT EXISTS Contract (

    Contract\_ID INT AUTO\_INCREMENT PRIMARY KEY,

    Vendor\_ID INT NOT NULL,

    Start\_Date DATE NOT NULL,

    End\_Date DATE NOT NULL,

    Terms\_and\_Conditions TEXT,

    Status ENUM('Active', 'Expired', 'Pending', 'Renewed') DEFAULT 'Pending',

    Renewal\_Notification\_Flag BOOLEAN DEFAULT FALSE,

    FOREIGN KEY (Vendor\_ID) REFERENCES Vendor(Vendor\_ID) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Budget (

    Budget\_ID INT AUTO\_INCREMENT PRIMARY KEY,

     Vendor\_ID INT NOT NULL,

    Allocated\_Amount DECIMAL(15, 2) NOT NULL,

    Spent\_Amount DECIMAL(15, 2) DEFAULT 0,

    Remaining\_Amount DECIMAL(15, 2) GENERATED ALWAYS AS (Allocated\_Amount - Spent\_Amount) STORED,

    FOREIGN KEY (Vendor\_ID) REFERENCES Vendor(Vendor\_ID) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Purchase\_Order (

    PO\_ID INT AUTO\_INCREMENT PRIMARY KEY,

    Vendor\_ID INT NOT NULL,

    Items\_Details TEXT NOT NULL,

    Quantity INT NOT NULL,

    Total\_Cost DECIMAL(15, 2) NOT NULL,

    Status ENUM('Created', 'Fulfilled', 'Pending') DEFAULT 'Pending',

    FOREIGN KEY (Vendor\_ID) REFERENCES Vendor(Vendor\_ID) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS Vendor\_Performance (

    Performance\_ID INT AUTO\_INCREMENT PRIMARY KEY,

    Vendor\_ID INT NOT NULL,

    Delivery\_Timeliness FLOAT DEFAULT 0,

    Service\_Quality FLOAT DEFAULT 0,

    Compliance\_Adherence FLOAT DEFAULT 0,

    Feedback TEXT,

    Rating FLOAT AS ((Delivery\_Timeliness + Service\_Quality + Compliance\_Adherence) / 3) STORED,

    FOREIGN KEY (Vendor\_ID) REFERENCES Vendor(Vendor\_ID) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS User (

    User\_ID INT AUTO\_INCREMENT PRIMARY KEY,

    Name VARCHAR(255) NOT NULL,

    Role ENUM('Procurement Manager', 'Budget Manager', 'Contract Team Member', 'Department Head') NOT NULL,

    Email VARCHAR(255) NOT NULL UNIQUE

);

CREATE TABLE IF NOT EXISTS Performance\_Evaluation\_History (

    Evaluation\_ID INT AUTO\_INCREMENT PRIMARY KEY,

    Vendor\_ID INT NOT NULL,

    Evaluation\_Date DATE NOT NULL,

    Delivery\_Timeliness FLOAT,

    Service\_Quality FLOAT,

    Compliance\_Adherence FLOAT,

    Feedback TEXT,

    Rating FLOAT AS ((Delivery\_Timeliness + Service\_Quality + Compliance\_Adherence) / 3) STORED,

    FOREIGN KEY (Vendor\_ID) REFERENCES Vendor(Vendor\_ID) ON DELETE CASCADE

);

* + **Using Express (code):**
* const express = require('express');
* const bodyParser = require('body-parser');
* const mysql = require('mysql2');
* const path = require('path');
* const app = express();
* const port = 3000;
* // MySQL connection setup
* const db = mysql.createConnection({
* host: 'localhost',
* user: 'root', // Your MySQL username
* password: '12345678', // Your MySQL password
* database: 'CorporateVendorManagement' // Your database name
* });
* // Connect to MySQL database
* db.connect((err) => {
* if (err) {
* console.error('Could not connect to MySQL:', err);
* } else {
* console.log('Connected to MySQL');
* }
* });
* // Middleware to parse incoming requests
* app.use(bodyParser.urlencoded({ extended: true }));
* app.use(bodyParser.json());
* // Serve static files (like styles.css) from the 'public' folder
* app.use(express.static(path.join(\_\_dirname, 'public')));
* // Serve the home page
* app.get('/', (req, res) => {
* res.sendFile(path.join(\_\_dirname, 'home.html')); // Home page with buttons for login and registration
* });
* // Serve the vendor management form (vendor.html)
* app.get('/vendor', (req, res) => {
* res.sendFile(path.join(\_\_dirname, 'vendor.html')); // Vendor registration form
* });
* // Serve the login page
* app.get('/login', (req, res) => {
* res.sendFile(path.join(\_\_dirname, 'login.html')); // Vendor login form
* });
  + - **So in this way the express and app is being coded for making html forms and inserting data into the sql database.**
    - **We can see the html pages are linked and many more pages are linked in similar way.**

All these would be included in Home.js

* **Vendor.js**

**app.post('/add-vendor', (req, res) => {**

**const { Name, Contact\_Info, Service\_Categories, Compliance\_Certifications, Performance\_Rating } = req.body;**

**const query = `**

**INSERT INTO Vendor (Name, Contact\_Info, Service\_Categories, Compliance\_Certifications, Performance\_Rating)**

**VALUES (?, ?, ?, ?, ?)**

**`;**

**db.query(query, [Name, Contact\_Info, Service\_Categories, Compliance\_Certifications, Performance\_Rating], (err, result) => {**

**if (err) {**

**console.error('Error inserting data:', err);**

**return res.status(500).send('Error inserting data.');**

**}**

**res.send('Vendor data has been inserted successfully!');**

**});**

**});**

* **LOGIN.JS**

**app.post('/login', (req, res) => {**

**const { email } = req.body;**

**const query = `SELECT \* FROM Vendor WHERE Contact\_Info = ?`;**

**db.query(query, [email], (err, results) => {**

**if (err) {**

**console.error('Error querying database:', err);**

**return res.status(500).send('Error querying database.');**

**}**

**if (results.length > 0) {**

**return res.send('You are eligible to open the dashboard!');**

**} else {**

**return res.send('Email not found. Please register first.');**

**}**

**});**

**});**

* **Contract.js**

app.post('/add-contract', (req, res) => {

    const { Vendor\_ID, Start\_Date, End\_Date, Terms\_and\_Conditions, Status } = req.body;

    const query = `

        INSERT INTO Contract (Vendor\_ID, Start\_Date, End\_Date, Terms\_and\_Conditions, Status)

        VALUES (?, ?, ?, ?, ?)

    `;

    const values = [Vendor\_ID, Start\_Date, End\_Date, Terms\_and\_Conditions, Status];

    db.query(query, values, (err, result) => {

        if (err) {

            console.error('Error inserting contract:', err);

            return res.status(500).send('Error inserting contract.');

        }

        res.send('Contract added successfully!');

    });

});

// Endpoint to set a renewal alert for a contract

app.post('/set-alert', (req, res) => {

    const { contractId, renewalFlag } = req.body;

    const query = `

        UPDATE Contract

        SET Renewal\_Notification\_Flag = ?

        WHERE Contract\_ID = ?

    `;

    db.query(query, [renewalFlag, contractId], (err, result) => {

        if (err) {

            console.error('Error updating contract alert:', err);

            return res.status(500).send('Error updating contract alert.');

        }

        if (result.affectedRows > 0) {

            res.send('Renewal alert updated successfully!');

        } else {

            res.status(404).send('Contract not found.');

        }

    });

});

// Endpoint to check for contracts about to expire in 30 days

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

    const today = new Date();

    const thirtyDaysFromNow = new Date(today);

    thirtyDaysFromNow.setDate(today.getDate() + 30);

    // Format the dates as strings in 'YYYY-MM-DD' format

    const todayFormatted = today.toISOString().split('T')[0];

    const thirtyDaysFormatted = thirtyDaysFromNow.toISOString().split('T')[0];

    const query = `

        SELECT Contract\_ID, Vendor\_ID, End\_Date

        FROM Contract

        WHERE End\_Date BETWEEN ? AND ?

    `;

    db.query(query, [todayFormatted, thirtyDaysFormatted], (err, results) => {

        if (err) {

            console.error('Error checking renewals:', err);

            return res.status(500).send('Error checking contract renewals.');

        }

        res.json(results);

    });

});

* **Purchase.js**

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

    const query = `

        SELECT

            PO\_ID, Vendor\_ID, Items\_Details, Quantity, Total\_Cost, Status

        FROM

            Purchase\_Order

    `;

    db.query(query, (err, results) => {

        if (err) {

            console.error('Error fetching purchase orders:', err);

            return res.status(500).send('Error fetching purchase orders.');

        }

        res.json(results);

    });

});

// Endpoint to add a new purchase order (excluding Department\_ID)

app.post('/add-purchase-order', (req, res) => {

    const { Vendor\_ID, Items\_Details, Quantity, Total\_Cost, Status } = req.body;

    const query = `

        INSERT INTO Purchase\_Order (Vendor\_ID, Items\_Details, Quantity, Total\_Cost, Status)

        VALUES (?, ?, ?, ?, ?)

    `;

    const values = [Vendor\_ID, Items\_Details, Quantity, Total\_Cost, Status || 'Pending'];

    db.query(query, values, (err, result) => {

        if (err) {

            console.error('Error inserting purchase order:', err);

            return res.status(500).send('Error inserting purchase order.');

        }

        res.send('Purchase order added successfully!');

    });

});

// Endpoint to update the status of a purchase order

app.post('/update-status', (req, res) => {

    const { PO\_ID, Status } = req.body;

    const query = `

        UPDATE Purchase\_Order

        SET Status = ?

        WHERE PO\_ID = ?

    `;

    db.query(query, [Status, PO\_ID], (err, result) => {

        if (err) {

            console.error('Error updating purchase order status:', err);

            return res.status(500).send('Error updating purchase order status.');

        }

        if (result.affectedRows > 0) {

            res.send('Purchase order status updated successfully!');

        } else {

            res.status(404).send('Purchase order not found.');

        }

    });

});

* **Budget.js:**

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

    const query = `

        SELECT

            Budget\_ID, Vendor\_ID, Allocated\_Amount, Spent\_Amount, Remaining\_Amount

        FROM

            Budget

    `;

    db.query(query, (err, results) => {

        if (err) {

            console.error('Error fetching budgets:', err);

            return res.status(500).send('Error fetching budgets.');

        }

        res.json(results);

    });

});

// Endpoint to add a new budget

app.post('/add-budget', (req, res) => {

    const { Vendor\_ID, Allocated\_Amount } = req.body;

    const query = `

        INSERT INTO Budget (Vendor\_ID, Allocated\_Amount)

        VALUES (?, ?)

    `;

    const values = [Vendor\_ID, Allocated\_Amount];

    db.query(query, values, (err, result) => {

        if (err) {

            console.error('Error inserting budget:', err);

            return res.status(500).send('Error inserting budget.');

        }

        res.send('Budget added successfully!');

    });

});

// Endpoint to update the spent amount in a budget

app.post('/update-spent', (req, res) => {

    const { Budget\_ID, Spent\_Amount } = req.body;

    const query = `

        UPDATE Budget

        SET Spent\_Amount = Spent\_Amount + ?

        WHERE Budget\_ID = ?

    `;

    db.query(query, [Spent\_Amount, Budget\_ID], (err, result) => {

        if (err) {

            console.error('Error updating spent amount:', err);

            return res.status(500).send('Error updating spent amount.');

        }

        if (result.affectedRows > 0) {

            res.send('Spent amount updated successfully!');

        } else {

            res.status(404).send('Budget not found.');

        }

    });

});

// Endpoint to delete a budget

app.delete('/delete-budget/:id', (req, res) => {

    const { id } = req.params;

    const query = `

        DELETE FROM Budget

        WHERE Budget\_ID = ?

    `;

    db.query(query, [id], (err, result) => {

        if (err) {

            console.error('Error deleting budget:', err);

            return res.status(500).send('Error deleting budget.');

        }

        if (result.affectedRows > 0) {

            res.send('Budget deleted successfully!');

        } else {

            res.status(404).send('Budget not found.');

        }

    });

});

* **Vendor performance.js**

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

    const { vendorId, startDate, endDate, sortBy } = req.query;

    let query = `

        SELECT

            Vendor\_Performance.Vendor\_ID,

            Vendor\_Performance.Performance\_ID,

            Vendor\_Performance.Delivery\_Timeliness,

            Vendor\_Performance.Service\_Quality,

            Vendor\_Performance.Compliance\_Adherence,

            Vendor\_Performance.Rating,

            Vendor\_Performance.Feedback

        FROM

            Vendor\_Performance

        JOIN

            Vendor ON Vendor\_Performance.Vendor\_ID = Vendor.Vendor\_ID

        WHERE

            Vendor\_Performance.Performance\_Date BETWEEN ? AND ?

    `;

    const values = [startDate, endDate];

    if (vendorId) {

        query += ` AND Vendor\_Performance.Vendor\_ID = ?`;

        values.push(vendorId);

    }

    query += ` ORDER BY ${sortBy || 'Rating'} DESC`; // Default sort by Rating

    db.query(query, values, (err, results) => {

        if (err) {

            console.error('Error fetching vendor performance:', err);

            return res.status(500).send('Error fetching vendor performance.');

        }

        res.json(results);

    });

});

// Endpoint to add a new vendor performance record

app.post('/add-vendor-performance', (req, res) => {

    const { Vendor\_ID, Delivery\_Timeliness, Service\_Quality, Compliance\_Adherence, Feedback } = req.body;

    const query = `

        INSERT INTO Vendor\_Performance (Vendor\_ID, Delivery\_Timeliness, Service\_Quality, Compliance\_Adherence, Feedback)

        VALUES (?, ?, ?, ?, ?)

    `;

    const values = [Vendor\_ID, Delivery\_Timeliness, Service\_Quality, Compliance\_Adherence, Feedback];

    db.query(query, values, (err, result) => {

        if (err) {

            console.error('Error inserting vendor performance:', err);

            return res.status(500).send('Error inserting vendor performance.');

        }

        res.send('Vendor performance record added successfully!');

    });

});

// Endpoint to update vendor performance record

app.post('/update-vendor-performance', (req, res) => {

    const { Performance\_ID, Delivery\_Timeliness, Service\_Quality, Compliance\_Adherence, Feedback } = req.body;

    const query = `

        UPDATE Vendor\_Performance

        SET Delivery\_Timeliness = ?, Service\_Quality = ?, Compliance\_Adherence = ?, Feedback = ?

        WHERE Performance\_ID = ?

    `;

    const values = [Delivery\_Timeliness, Service\_Quality, Compliance\_Adherence, Feedback, Performance\_ID];

    db.query(query, values, (err, result) => {

        if (err) {

            console.error('Error updating vendor performance:', err);

            return res.status(500).send('Error updating vendor performance.');

        }

        if (result.affectedRows > 0) {

            res.send('Vendor performance updated successfully!');

        } else {

            res.status(404).send('Vendor performance not found.');

        }

    });

});

// Endpoint to delete a vendor performance record

app.delete('/delete-vendor-performance/:id', (req, res) => {

    const { id } = req.params;

    const query = `

        DELETE FROM Vendor\_Performance

        WHERE Performance\_ID = ?

    `;

    db.query(query, [id], (err, result) => {

        if (err) {

            console.error('Error deleting vendor performance:', err);

            return res.status(500).send('Error deleting vendor performance.');

        }

        if (result.affectedRows > 0) {

            res.send('Vendor performance deleted successfully!');

        } else {

            res.status(404).send('Vendor performance not found.');

        }

    });

});

**9. PROCEDURES:**

### **1. Vendor Registration (Procedure: VendorRegistration)**

#### ****Purpose:****

* To simplify the process of adding new vendors to the database.
* Allows easy integration of vendor details into the Vendor table.

#### ****Significance:****

* Automates the vendor registration process by reducing manual SQL commands.
* Ensures consistent data entry for new vendors with required attributes such as name, contact information, service categories, and compliance certifications.

-- Stored Procedure for Vendor Registration

CREATE PROCEDURE VendorRegistration (

    IN p\_Name VARCHAR(255),

    IN p\_Contact\_Info VARCHAR(255),

    IN p\_Service\_Categories VARCHAR(255),

    IN p\_Compliance\_Certifications VARCHAR(255)

)

BEGIN

    INSERT INTO Vendor (Name, Contact\_Info, Service\_Categories, Compliance\_Certifications)

    VALUES (p\_Name, p\_Contact\_Info, p\_Service\_Categories, p\_Compliance\_Certifications);

END;

### **2. Contract Management (Procedure: CreateContract):**

#### ****Purpose:****

* To manage and create contracts for vendors by inserting relevant contract details into the Contract table.

#### ****Significance:****

* Simplifies contract creation by allowing essential contract attributes (vendor ID, start date, end date, terms, etc.) to be inserted in one step.
* Ensures the association between a vendor and their contracts is properly maintained through the Vendor\_ID foreign key.
* -- Stored Procedure for Contract Management
* CREATE PROCEDURE CreateContract (
* IN p\_Vendor\_ID INT,
* IN p\_Start\_Date DATE,
* IN p\_End\_Date DATE,
* IN p\_Terms\_and\_Conditions TEXT
* )
* BEGIN
* INSERT INTO Contract (Vendor\_ID, Start\_Date, End\_Date, Terms\_and\_Conditions)
* VALUES (p\_Vendor\_ID, p\_Start\_Date, p\_End\_Date, p\_Terms\_and\_Conditions);
* END;

### **3. Vendor Performance Evaluation (Procedure:** VendorPerformanceEvaluation**)**

#### ****Purpose:****

* To assess and record vendor performance metrics such as delivery timeliness, service quality, and compliance adherence.
* Stores performance feedback for future reference and evaluation.

#### ****Significance:****

* Facilitates consistent evaluation of vendor performance based on defined metrics.
* Enables the organization to maintain a detailed record of vendor reliability and quality.

-- Stored Procedure for Vendor Performance Evaluation

CREATE PROCEDURE VendorPerformanceEvaluation (

    IN p\_Vendor\_ID INT,

    IN p\_Delivery\_Timeliness FLOAT,

    IN p\_Service\_Quality FLOAT,

    IN p\_Compliance\_Adherence FLOAT,

    IN p\_Feedback TEXT

)

BEGIN

    INSERT INTO Vendor\_Performance (Vendor\_ID, Delivery\_Timeliness, Service\_Quality, Compliance\_Adherence, Feedback)

    VALUES (p\_Vendor\_ID, p\_Delivery\_Timeliness, p\_Service\_Quality, p\_Compliance\_Adherence, p\_Feedback);

END;

`;

**10. TRIGGERS:**

**Contract Renewal Notifications:**

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

    const today = new Date();

    const thirtyDaysFromNow = new Date(today);

    thirtyDaysFromNow.setDate(today.getDate() + 30);

    // Format the dates as strings in 'YYYY-MM-DD' format

    const todayFormatted = today.toISOString().split('T')[0];

    const thirtyDaysFormatted = thirtyDaysFromNow.toISOString().split('T')[0];

    const query = `

        SELECT Contract\_ID, Vendor\_ID, End\_Date

        FROM Contract

        WHERE End\_Date BETWEEN ? AND ?

    `;

    db.query(query, [todayFormatted, thirtyDaysFormatted], (err, results) => {

        if (err) {

            console.error('Error checking renewals:', err);

            return res.status(500).send('Error checking contract renewals.');

        }

        res.json(results);

    });

});

**DDL script:**

DELIMITER //

CREATE TRIGGER check\_contract\_renewals\_insert

AFTER INSERT ON contracts

FOR EACH ROW

BEGIN

DECLARE days\_left INT;

-- Calculate days left until expiration

SET days\_left = DATEDIFF(NEW.expiry\_date, CURDATE());

-- If the contract is expiring in the next 30 days, log the notification

IF days\_left <= 30 AND days\_left >= 0 THEN

-- Insert a record into the 'contract\_notifications' table for the expiring contract

INSERT INTO contract\_notifications (contract\_id, message, notification\_date)

VALUES (NEW.contract\_id, CONCAT('Contract ', NEW.contract\_id, ' is expiring soon!'), CURDATE());

END IF;

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER check\_contract\_renewals\_update

AFTER UPDATE ON contracts

FOR EACH ROW

BEGIN

DECLARE days\_left INT;

-- Calculate days left until expiration

SET days\_left = DATEDIFF(NEW.expiry\_date, CURDATE());

-- If the contract is expiring in the next 30 days, log the notification

IF days\_left <= 30 AND days\_left >= 0 THEN

-- Insert a record into the 'contract\_notifications' table for the expiring contract

INSERT INTO contract\_notifications (contract\_id, message, notification\_date)

VALUES (NEW.contract\_id, CONCAT('Contract ', NEW.contract\_id, ' is expiring soon!'), CURDATE());

END IF;

END //

DELIMITER ;

**Purchase order budget checks query:**

SELECT

po.PO\_ID,

po.Total\_Cost,

b.Allocated\_Amount,

(po.Total\_Cost - b.Allocated\_Amount) AS Over\_Budget\_Amount

FROM

Purchase\_Order po

JOIN

Budget b ON po.Vendor\_ID = b.Vendor\_ID

WHERE

po.Total\_Cost > b.Allocated\_Amount;

#### ****11. Conclusion****

The Corporate Vendor Management System is a scalable and robust solution for managing vendors. By leveraging modern technologies, the system simplifies the vendor lifecycle management process, ensuring data accuracy and compliance. This project lays the groundwork for future enhancements, making it adaptable to various business requirements.