**Detailed Project Report**

**1. Database Creation**

The first step in the process is creating the database. In this case, we use the SQL command CREATE DATABASE IF NOT EXISTS vendor\_management; to create a database named vendor\_management. The IF NOT EXISTS clause ensures that the database will only be created if it doesn't already exist. Once the database is created, the USE vendor\_management; command makes it the active database for the subsequent operations. This is crucial as it establishes the context in which all the tables, triggers, and procedures will be defined.

**2. Table Creation**

* **Vendors Table**

The Vendors table stores key information about the vendors involved in the system. This table is designed to store vendor-related details such as their name, service category, contact information, and compliance status. The **VendorID** is an integer that automatically increments with each new record, serving as the primary key and ensuring that each vendor has a unique identifier. The **VendorName** field stores the name of the vendor, which is a required field. **ServiceCategory** describes the type of services offered by the vendor, while **ContactName, ContactPhone,** and **ContactEmail** store the vendor's primary contact information. The **ComplianceStatus** is a boolean value that indicates whether the vendor complies with the organization's required standards, with a default value of FALSE. The **Certification** field stores any certifications that the vendor holds. Lastly, the **CreatedAt** and **UpdatedAt** fields are automatically updated to track when the record was created and last modified, ensuring that data is always current.

* **Departments Table**

The Departments table stores information about the various departments within the organization. Each department has a unique identifier, **DepartmentID**, which serves as the primary key for this table. The **DepartmentName** field holds the name of the department, and **Budget** stores the allocated budget for that department. The table ensures that all department-related details are tracked in a centralized location, helping the organization manage departmental budgets effectively.

* **Budgets Table**

The Budgets table is closely linked to the Departments table, as it tracks the budgets allocated to each department. Each record in this table is identified by a unique **BudgetID**, which serves as the primary key. The **DepartmentID** is a foreign key that links each budget record to a specific department, ensuring that budgets are associated with the right department. **AllocatedAmount** represents the total budget allocated to the department, and **RemainingAmount** keeps track of the budget still available for use. This table is essential for monitoring financial resources and ensuring that departments do not exceed their allocated budgets.

* **PurchaseOrders Table**

The PurchaseOrders table is responsible for storing the details of purchase orders made by different departments for vendors. Each purchase order is identified by a unique PurchaseOrderID, which is the primary key. The VendorID and DepartmentID are foreign keys that link each purchase order to the relevant vendor and department, respectively. The OrderDate records the date when the purchase order was made, and TotalAmount captures the total cost of the order. The BudgetID is a foreign key to the Budgets table, indicating which department's budget is being used for the purchase. Finally, the Status field holds the current status of the order, which can be Created, Fulfilled, or Cancelled, helping track the order's progress through the procurement process

* **Contracts Table**

The Contracts table stores information about the contracts between the organization and vendors. Each contract has a unique ContractID, which serves as the primary key. The VendorID is a foreign key that links each contract to a specific vendor. The ContractName field describes the name or title of the contract, while ContractDetails holds additional information about the terms and conditions. StartDate and EndDate track the duration of the contract, while TotalAmount represents the financial value of the contract. The Status field indicates whether the contract is Active, Expired, or Terminated, helping the organization manage its vendor agreements. Lastly, CreatedAt and UpdatedAt track the record's creation and modification times.

**3. Triggers**

* **Update Vendor Timestamp Trigger**

Triggers are special SQL procedures that automatically execute when specific events occur in the database. The **Update Vendor Timestamp Trigger** is designed to automatically update the **UpdatedAt** timestamp in the Vendors table whenever a vendor's record is modified. This ensures that the timestamp remains current and reflects the last time the vendor's details were changed. The trigger is set to run before an update operation, ensuring that the new timestamp is set before the update occurs.

* **Update Contract Timestamp Trigger**

Similarly, the Update Contract Timestamp Trigger ensures that the UpdatedAt timestamp in the Contracts table is updated whenever contract details are modified. This helps maintain accurate records of when contract information was last altered, aiding in contract management and auditing processes. Like the vendor timestamp trigger, this trigger is set to run before any update operation on the Contracts table.

**4. Stored Procedures**

* **Register Vendor Procedure**

The RegisterVendor stored procedure is used to insert new vendor records into the Vendors table. It takes several input parameters, such as the vendor's name, service category, contact information, compliance status, and certification. Before inserting the data, the procedure checks that mandatory fields (such as VendorName and ContactName) are not null. It also validates the phone number and email formats to ensure the data is correct. If any validation fails, the procedure raises an error. Once the data is validated, the vendor information is inserted into the Vendors table. This stored procedure ensures that vendor registration follows consistent rules and maintains data integrity.

**HTML CODE EXPLANATION:**

**Structure and Features:**

1. Sidebar Navigation:
   * The sidebar is fixed to the left side of the page, making navigation intuitive and accessible at all times.
   * It contains links to different sections of the system, such as *Dashboard*, *Vendors*, *Contracts*, *Purchase Orders*, and *Budgets*.
   * It highlights the active section with a distinct color for better user experience and expands slightly on hover.
2. Main Content:
   * The main content area adjusts based on the sidebar width and is styled for readability.
   * Key sections are:
     + Dashboard: This serves as a quick overview, displaying information like active contracts and their expiration dates in a table format.
     + Vendors: Includes a form for vendor registration with fields like vendor name, contact information, certifications, and ratings. Below the form, a table lists registered vendors with their contact details and ratings.
     + Contracts: Lists contract details, including titles, associated vendors, and expiration dates, in a tabular format.
     + Purchase Orders: Displays purchase order information such as PO ID, vendor, amount, and current status (e.g., pending or approved).
     + Budgets: Summarizes the budget for each fiscal year, showing the total allocated amount, spent amount, and remaining balance.
3. Styling and Design:
   * The layout is styled with CSS, focusing on a professional and modern look. The sidebar uses a dark theme with white text, while the main content employs lighter colors for better contrast.
   * Tables are styled with alternating row colors for better readability. Interactive elements like buttons and input fields have hover effects for enhanced usability.
   * A responsive design ensures the system adapts to smaller screens by collapsing the sidebar and adjusting the main content layout.
4. Interactive Elements:
   * The vendor registration form uses standard input fields for text, email, number, and dropdown selections. A submit button allows the user to register vendors.
   * Tables across sections dynamically display data, making it easy to organize and retrieve information.

The above HTML code represents a **Corporate Vendor Management** webpage. It has a sidebar navigation and is divided into several sections, each handling a specific aspect of vendor and contract management. Here's a detailed breakdown:

**Structure and Features**

1. **Sidebar Navigation**:
   * A fixed sidebar on the left side of the page contains navigation links.
   * Links lead to different sections: Dashboard, Vendors, Contracts, Purchase Orders, and Budgets.
   * The active link is highlighted with a distinct color.
2. **Main Content**:
   * Positioned next to the sidebar, this section dynamically adjusts its layout when the screen size changes, ensuring responsiveness.
3. **Dashboard Section**:
   * This section provides a summary of key details like active contracts.
   * A table displays active contracts with columns for **Contract Title**, **Vendor**, and **Expiration Date**.
4. **Vendor Registration**:
   * Includes a form where users can register vendors by entering their name, contact info, compliance certifications, and ratings.
   * Displays a list of registered vendors in a table format with details like name, contact info, and rating.
5. **Contracts Section**:
   * Provides an overview of contracts, displayed in a table format.
   * Columns include **Contract Title**, **Vendor**, and **Expiration Date**.
6. **Purchase Orders**:
   * This section lists purchase orders with details such as PO ID, Vendor, Amount, and Status.
   * The table highlights pending and approved orders.
7. **Budgets**:
   * Displays a budget overview in a tabular format with columns for Fiscal Year, Total Amount, Allocated Amount, and Remaining Balance.
   * Multiple budget entries show how funds are distributed and utilized.

**Design Elements:**

* **Color Scheme**:
  + Sidebar and headings use a dark theme with contrasting white text.
  + Tables are styled with alternating row colors and hover effects to enhance readability.
* **Form Styling**:
  + Vendor registration form uses input fields for text, email, and number types.
  + Includes placeholders and validations for user input.
  + A button styled with a hover effect submits the form.
* **Responsiveness**:
  + Uses media queries to adapt the layout for smaller screens. The sidebar becomes relative, and padding adjusts for readability on mobile devices

The provided HTML code represents a **Vendor Management System** with a simple form to add vendor details and a table to display the list of registered vendors.

At the top, the webpage includes a title and a heading, "Vendor Management," to introduce the purpose of the application. The main feature is a form, identified by the ID addVendorForm, which allows users to input vendor information. The form contains several fields:

* Vendor Name: A required text field for the vendor's name.
* Service Category: An optional text field to specify the type of services offered by the vendor.
* Contact Name: An optional text field for the primary contact person's name.
* Contact Phone: A required text field for the contact phone number.
* Contact Email: A required email field for the vendor's email address.
* Compliance Status: A checkbox to indicate whether the vendor meets compliance standards.
* Certification: An optional text field for entering any certifications held by the vendor.

The form concludes with a submit button labeled **"Add Vendor"** that allows users to save the entered details.

Below the form, there is a **Vendor List** section featuring a table. The table is structured with a header row containing five columns:

1. ID - Displays the unique identifier for each vendor.
2. Vendor Name - Shows the name of the vendor.
3. Contact Email - Lists the email address of the vendor.
4. Contact Phone - Contains the vendor's contact phone number.
5. Actions - Provides a placeholder for possible operations (e.g., editing or deleting vendor details).

This HTML code represents a **Login Page** for a Corporate Vendor Management system. The design focuses on user-friendly features with a modern and visually appealing layout.

The page begins with the inclusion of meta tags for proper character encoding (UTF-8) and responsiveness, ensuring it adapts well to different screen sizes. The title of the page, "Login - Corporate Vendor Management," reflects its purpose.

The **styling**, embedded within the <style> tag, uses the **Roboto font**, fetched from Google Fonts, for a clean and professional look. The background features a **linear gradient** transitioning from peach (#f7b7a3) to pink (#fca5e2), providing a vibrant yet subtle aesthetic. The content is centered with ample padding, ensuring a neat and uncluttered appearance.

The **header (<h1>)** displays the page title, "Login to Corporate Vendor Management," in large, white, and bold text, creating a strong focal point.

The **login form**, styled with the class login-form, is vertically aligned and centered on the page. It uses a **POST method** for secure data submission to the /login endpoint. The form contains two input fields:

1. **Username Field**: A text input field for entering the username, marked as required.
2. **Password Field**: A password input field that masks the entered text for privacy, also marked as required.

Below the input fields, there is a **login button** styled with a green background (#1abc9c). The button changes to a darker shade of green (#16a085) when hovered over, enhancing interactivity. The button spans the width of the input fields for visual consistency.

At the bottom, a **call-to-action message** encourages users without an account to sign up, providing a link to the sign-up page (/signup.html). The link text is white to match the design theme.

Overall, the layout combines simplicity, elegance, and functionality, ensuring a seamless experience for users accessing the Corporate Vendor Management system.

**Structure and Design:**

1. **HTML Setup**: The document starts with meta tags for character encoding (UTF-8) and viewport settings, ensuring proper display across devices. The title of the page is "Vendor Performance Evaluation."
2. **CSS Styling**: Inline styles define a professional and user-friendly appearance:
   * The form is centered on the page with a maximum width of 500px, white background, rounded corners, and a subtle shadow for visual distinction.
   * Form elements such as labels, inputs, and buttons are styled for readability and usability. Inputs and textareas have rounded corners, uniform padding, and border styling, while buttons use a green theme with hover effects.

**Functional Elements:**

1. **Form Fields**: The form includes the following fields:
   * **Vendor Dropdown**: A dynamically populated dropdown menu for selecting a vendor.
   * **Rating Input**: A number field allowing ratings between 0 and 5, with a step of 0.1 for precision.
   * **Review Textarea**: A multi-line text field for user reviews, with placeholder text for guidance.
2. **Submit Button**: A green button at the bottom of the form for submitting the evaluation. Hover effects change its color to a darker green for better user experience.

**JavaScript Functionality:**

1. **Dynamic Vendor Loading**:
   * An async function (loadVendors) fetches vendor data from the /get-vendors endpoint.
   * Upon receiving a response, the vendor dropdown is populated dynamically with options. Each option uses the vendor’s id as its value and name as its label.
   * Error handling ensures appropriate feedback is provided in case of server or network issues.
2. **Form Submission**:
   * The form submission process is intercepted with a submit event listener, preventing the default page reload.
   * The selected vendor, rating, and review are extracted from the form fields and compiled into a JSON object.
   * This data is sent to the /submit-performance-evaluation endpoint via a POST request using the fetch API.
   * The response is handled to confirm success or display error messages as needed.

**User Workflow:**

1. When the page loads, the **loadVendors function** is called to fetch and populate the vendor dropdown. If no vendors are available, an alert notifies the user.
2. After filling out the form, users submit the evaluation by clicking the "Submit Evaluation" button. The data is sent to the server, and the user receives feedback on the submission status.

**Key Features:**

* **Responsive and Clean Design**: A visually appealing form that adapts well to different devices.
* **Dynamic Vendor Population**: Vendors are fetched dynamically, avoiding hardcoding and ensuring up-to-date data.
* **Error Handling**: Errors during vendor loading or form submission are logged and conveyed to the user.
* **Interactivity**: Real-time feedback is provided after form submission, improving the user experience.

It features a form with three key fields: **Contract**, **Order Amount**, and **Order Date**. The form is styled for a clean and modern appearance using CSS, with features like rounded borders, shadows, and a responsive design to ensure usability on various devices. The **Contract** field is a dropdown menu dynamically populated with options fetched from the server. Users are expected to select a contract from the dropdown, input the desired order amount, and specify the order date.

The accompanying JavaScript code includes a function, loadContracts(), that fetches contract data from the server endpoint /get-contracts. If successful, it populates the dropdown menu with contract titles and their respective IDs. If no contracts are available or an error occurs, appropriate alerts are displayed.

Form submission is handled by an event listener on the submit button. When submitted, the form prevents the default page reload, collects user input, and sends it as a JSON object to the server endpoint /create-purchase-order using a POST request. If the request is successful, a success message is displayed; otherwise, an error message is shown. The JavaScript code is designed to handle errors gracefully and provide user feedback in case of failures.

This HTML document presents a **Sign-Up Page** for a **Corporate Vendor Management System**. The page features an elegant and visually appealing design, achieved using a combination of responsive styling and a gradient background transitioning from peach to pink. The header, styled with a large, bold white font, immediately captures the user's attention with a welcoming message.

The central element of the page is the **Sign-Up Form**, which is designed to be simple, user-friendly, and functional. It consists of three fields:

1. **Username**: A text input field for entering the desired username.
2. **Password**: A secure password input field for user authentication.
3. **Role**: A dropdown menu allowing users to select their role (either "User" or "Admin").

Each input field and button is styled with rounded corners, consistent spacing, and appropriate padding to ensure a clean and modern look. The form's button, styled with a warm orange hue, becomes slightly darker when hovered over, adding an interactive and polished touch.

The form submission is configured to use the **POST** method to send user data to the /signup endpoint on the server. Below the form, there's a call-to-action message for users who already have an account, including a link to the login page styled in white to maintain consistency with the theme.

**Form Fields and Functionality**

1. **Vendor Name**:
   * A text input field where users can enter the vendor's name.
   * Includes a required attribute to ensure this field is filled before submission.
2. **Contact Info**:
   * An email input field for the vendor's contact information.
   * Validation for email format is enforced, making it user-friendly and secure.
3. **Compliance Certifications**:
   * A multi-line textarea field where users can enter any certifications the vendor may have.
   * Optional, allowing flexibility in data entry.
4. **Rating (0 - 5)**:
   * A number input field for assigning a rating between 0 and 5.
   * Includes a step attribute for decimal inputs and min/max constraints for validation.
5. **Submit Button**:
   * A styled button for submitting the form, clearly labeled "Register Vendor."
   * Includes hover effects for a polished, interactive user experience.

**Styling**

The form's design uses CSS to ensure:

* **Responsive Layout**: The form is centered on the page with a maximum width of 500px for compatibility across devices.
* **Aesthetic Appeal**: Soft shadows, rounded corners, and consistent spacing provide a modern, professional look.
* **User Comfort**: Adequate padding and clear labels improve usability.

**JavaScript for Dynamic Behavior**

The JavaScript handles the form submission process:

1. **Prevention of Default Submission**:
   * The submit event is intercepted to prevent the browser's default form behavior.
2. **Data Collection and Packaging**:
   * Captures user inputs and structures them into a JSON object.
3. **Server Communication**:
   * Uses the fetch API to send the collected data via a POST request to the /register-vendor endpoint.
   * Ensures secure communication with appropriate headers and JSON payload.
4. **Error Handling**:
   * Displays user-friendly alerts for success or errors.
   * Logs errors to the console for debugging purposes.

* **Document Type Declaration and Basic Structure**

The document starts with <!DOCTYPE html>, which declares the document as an HTML5 document, ensuring modern standards and compatibility across browsers. The <html> tag marks the beginning of the HTML document, and the lang="en" attribute specifies that the document's primary language is English. The <head> section contains metadata and links to external resources, such as a stylesheet.

* **Metadata in the Head Section**

The <head> section includes:

* <meta charset="UTF-8">: This specifies that the document uses UTF-8 encoding, supporting a wide range of characters.
* <meta name="viewport" content="width=device-width, initial-scale=1.0">: This tag ensures the page is responsive by adjusting its layout based on the screen size of the device.
* <title>Vendor Management</title>: Defines the title of the web page, displayed on the browser tab.
* <link rel="stylesheet" href="styles.css">: Links an external CSS file named styles.css to define the visual style of the page. The rel="stylesheet" attribute indicates it is a style sheet, and href specifies the file path.

**Body Content**

* **Page Header**

The <h1>Vendor Management</h1> creates a heading for the page, indicating its purpose: vendor management. It gives users a clear understanding of the page's functionality.

* **Form for Adding a New Vendor**

The <form id="addVendorForm"> element defines a form for collecting vendor information. Forms are interactive elements where users can input data. Each form field is explained below:

* <input type="text" id="vendorName" placeholder="Vendor Name" required />: A text input field for entering the vendor's name. The placeholder attribute displays placeholder text inside the field, and required ensures that this field must be filled before submission.
* <input type="text" id="serviceCategory" placeholder="Service Category" />: A text input field for specifying the service category of the vendor.
* <input type="text" id="contactName" placeholder="Contact Name" />: A text input field for entering the contact person's name.
* <input type="text" id="contactPhone" placeholder="Contact Phone" required />: A text input field for the vendor's phone number. This field is mandatory.
* <input type="email" id="contactEmail" placeholder="Contact Email" required />: An email input field for the vendor's email address. The type="email" ensures that the input follows an email format, and required makes it mandatory.
* <label for="complianceStatus">Compliance Status</label>: A label for the compliance status checkbox, improving accessibility and usability.
* <input type="checkbox" id="complianceStatus" />: A checkbox input for indicating whether the vendor is compliant.
* <input type="text" id="certification" placeholder="Certification" />: A text input field for specifying certifications held by the vendor.
* <button type="submit">Add Vendor</button>: A button for submitting the form. When clicked, it triggers the form's submission logic.
* **Vendor List Section**

The <h2>Vendor List</h2> heading introduces a section for displaying a list of vendors. It is followed by an HTML table structure:

* Table Structure:
  + <table>: Defines the table element.
  + <thead>: Contains the table's header row.
    - <tr>: A table row within the header.
    - <th>: Table header cells for ID, Vendor Name, Contact Email, Contact Phone, and Actions.
  + <tbody id="vendorList">: An empty <tbody> element with an id of vendorList, where vendor records will be dynamically added via JavaScript.

**JavaScript Inclusion**

At the end of the <body> section, the <script src="script.js"></script> tag links to an external JavaScript file named script.js. This file will handle the dynamic functionalities of the page, such as adding vendors, displaying the vendor list, and managing actions. Including the script at the end ensures the HTML content is loaded before the JavaScript executes.

The CSS code provided sets up a visually appealing and user-friendly layout for the vendor management interface. Below is an explanation of the styling, categorized into different functional areas:

**CSS STYLING CODE EXPLANATON:**

**Basic Reset**

The universal selector (\*) is used to remove the default margin and padding for all HTML elements, ensuring consistent spacing across browsers. Additionally, the box-sizing: border-box property adjusts the box model so that padding and borders are included in an element’s total width and height. This makes layout management simpler and more predictable.

**Body Styling**

The body element is styled to use a clean, modern font, specifically Arial or a similar sans-serif font. The background is set to a light grayish color (#f4f4f9) to provide a neutral and professional backdrop. The text color is a dark gray (#333) for good contrast and readability. A padding of 20px ensures that the content doesn’t stick to the edges of the viewport.

**Main Title (h1)**

The main heading (h1) is centrally aligned using text-align: center and is given a large font size of 36px for prominence. The color (#2d3a3a) is a dark greenish shade, complementing the light background. The margin below the title creates separation from the rest of the content.

**Form Styling**

The form is designed to be compact and user-friendly. It has a white background and slightly rounded corners (border-radius: 8px) for a modern look. Padding inside the form ensures that its elements are not cramped, while a box shadow gives it a subtle 3D effect, making it stand out against the page background.

Input fields within the form are styled to be wide (width: 100%) and easy to interact with, having adequate padding and border-radius. Focused input fields change border color to a darker greenish shade (#2d3a3a), providing a clear visual cue for user interaction. The submit button has a dark green background, white text, and a hover effect that changes the background to an even darker shade for an interactive feel.

**Vendor List Title (h2)**

The subheading (h2) follows a similar style to the main title but with a smaller font size (24px). This creates a clear visual hierarchy between the primary and secondary headings. It is also center-aligned and uses the same color scheme.

**Table Styling**

The table is styled to display vendor information cleanly and professionally. It spans the full width of its container (width: 100%) and features a white background with a subtle box shadow for separation from the page. Borders between rows are light gray, enhancing readability without being obtrusive.

Table headers (th) have a dark greenish background (#2d3a3a) with white text for contrast, making them stand out from the data rows. The table cells (td) are padded for better spacing and alignment.

**Button Styling in Table Rows**

Action buttons in table rows are styled with a red background (#e74c3c) and white text to signify their importance, often used for deletion or critical actions. Rounded corners (border-radius: 4px) and a hover effect darkening the background provide a polished and interactive user experience.

**NODE JS CODE EXPLANATIONS:**

1. **Index.js**

**Importing Required Modules**

The script begins by importing necessary modules:

1. **Express**: Used to create a server and handle HTTP requests.
2. **MySQL2**: A library for connecting and interacting with a MySQL database.
3. **Path**: A core Node.js module to work with file and directory paths.

**Creating an Express App**

An instance of an Express application is created using express(). The server listens on port 3000, which is defined in the port variable.

**Middleware Configuration**

The middleware is set up to handle:

1. **JSON Parsing**: express.json() allows the server to parse incoming requests with JSON payloads.
2. **URL-encoded Data Parsing**: express.urlencoded({ extended: true }) is used to parse form data.
3. **Static File Serving**: The express.static() middleware serves front-end files from the public directory, enabling integration of HTML, CSS, and JavaScript files for the client-side interface.

**MySQL Database Configuration and Connection**

A MySQL database connection is established using the mysql.createConnection method. It specifies the host, username, password, and database name required for authentication. If the connection fails, an error message is logged, and the process terminates to ensure no further operations occur without a database connection. Upon successful connection, a confirmation message is displayed.

**API Routes for Vendor Management**

The following RESTful API endpoints are defined to interact with the database:

1. **Retrieve All Vendors (GET /vendors)**  
   This endpoint fetches all records from the Vendors table in the database. The SQL query SELECT \* FROM Vendors retrieves the data. If an error occurs during execution, a 500 status code is returned with an error message. Otherwise, the results are sent back as a JSON response.
2. **Add a New Vendor (POST /vendors)**  
   This endpoint allows adding a new vendor to the database. The server expects a JSON payload with vendor details such as name, contact information, compliance status, and certification. The INSERT INTO SQL query inserts these details into the Vendors table. If successful, a 201 status code is returned to indicate resource creation. Errors are logged and a 500 status code is returned for failures.
3. **Delete a Vendor by ID (DELETE /vendors/:id)**  
   This endpoint removes a specific vendor based on their unique VendorID. The ID is extracted from the request's URL parameters and passed to the SQL query DELETE FROM Vendors WHERE VendorID = ?. A success message is returned if the deletion is successful, or a 500 status code if an error occurs.

**Starting the Server**

The server is configured to listen on the specified port (3000). When the server starts, a log message confirms that it is running and provides the URL to access it locally.

**Purpose of the Code**

This code serves as the backend logic for a vendor management system. It handles key CRUD (Create, Read, Update, Delete) operations for managing vendor data efficiently. The structured API allows integration with a front-end client, enabling users to view, add, and delete vendors through a user-friendly interface.

**Error Handling and User Experience**

The code includes robust error handling mechanisms to log errors and send appropriate HTTP status codes. This ensures that users receive meaningful feedback about operations while providing developers with clear diagnostic information.

1. **Script.js**

**Fetching and Displaying All Vendors**

The fetchVendors function retrieves the list of vendors from the backend by making a GET request to the /vendors endpoint. Upon receiving the data in JSON format:

1. It clears any existing content in the HTML table element with the ID vendorList.
2. Iterates through the fetched data, where each vendor object contains details like VendorID, VendorName, ContactEmail, and ContactPhone.
3. Dynamically creates a new table row (<tr>), populating it with vendor details and a "Delete" button. The button calls the deleteVendor function with the vendor's ID when clicked.
4. Appends the new rows to the vendorList table, effectively displaying the data on the web page.

If an error occurs during the fetch operation, it logs an error message to the console.

**Adding a New Vendor**

When the "Add Vendor" form is submitted, the submit event listener on the form intercepts the submission:

1. Prevents the default form submission behavior using e.preventDefault().
2. Extracts values from the input fields (e.g., vendor name, service category, contact information, compliance status).
3. Converts the form data into a JSON object and sends it to the /vendors endpoint via a POST request using the Fetch API.
4. After a successful addition, it logs a confirmation message to the console and calls fetchVendors to refresh the displayed vendor list.

Errors during this operation are caught and logged in the console.

**Deleting a Vendor**

The deleteVendor function deletes a vendor by:

1. Making a DELETE request to the /vendors/:id endpoint, where :id is replaced with the vendor's unique VendorID.
2. Upon success, it logs a confirmation message and refreshes the vendor list by calling fetchVendors.
3. Logs errors to the console in case of failure.

Each row in the vendor list contains a "Delete" button, which calls this function with the corresponding vendor's ID when clicked.

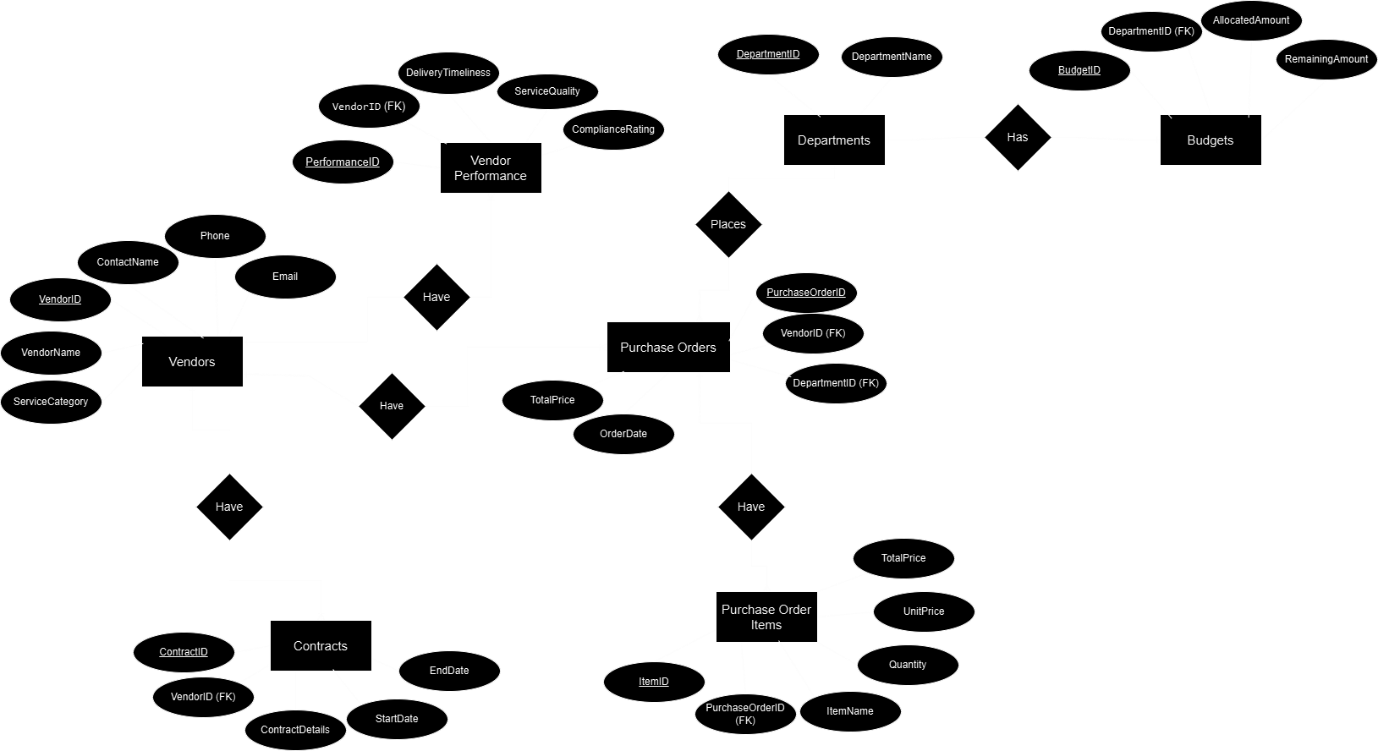
**Initializing the Vendor List**

The fetchVendors function is called when the page loads by attaching it to the window.onload event. This ensures the vendor list is displayed as soon as the page is ready.

**Purpose of the Code**

This code acts as the front-end logic for a vendor management system. It:

1. Dynamically fetches and displays vendor data from the backend.
2. Allows users to add new vendors through a form.
3. Provides a mechanism to delete vendors directly from the list.



**1. Entities and Attributes:**

* **Vendors**:
  + VendorID (Primary Key): Unique identifier for each vendor.
  + VendorName: Name of the vendor.
  + ContactName: Name of the contact person.
  + Phone: Vendor's contact phone number.
  + Email: Vendor's contact email.
  + ServiceCategory: Type of service provided by the vendor.
* **Vendor Performance**:
  + PerformanceID (Primary Key): Unique identifier for each performance record.
  + VendorID (Foreign Key): Links to Vendors.
  + DeliveryTimeliness: Rating for delivery punctuality.
  + ServiceQuality: Rating for service quality.
  + ComplianceRating: Rating for compliance with terms.
* **Contracts**:
  + ContractID (Primary Key): Unique identifier for each contract.
  + VendorID (Foreign Key): Links to Vendors.
  + ContractDetails: Description of the contract.
  + StartDate: Contract start date.
  + EndDate: Contract end date.
* **Departments**:
  + DepartmentID (Primary Key): Unique identifier for each department.
  + DepartmentName: Name of the department.
* **Budgets**:
  + BudgetID (Primary Key): Unique identifier for each budget.
  + DepartmentID (Foreign Key): Links to Departments.
  + AllocatedAmount: Total allocated budget for the department.
  + RemainingAmount: Remaining budget amount.
* **Purchase Orders**:
  + PurchaseOrderID (Primary Key): Unique identifier for each purchase order.
  + VendorID (Foreign Key): Links to Vendors.
  + DepartmentID (Foreign Key): Links to Departments.
  + OrderDate: Date when the purchase order was created.
  + TotalPrice: Total cost of the purchase order.
* **Purchase Order Items**:
  + ItemID (Primary Key): Unique identifier for each item in the purchase order.
  + PurchaseOrderID (Foreign Key): Links to Purchase Orders.
  + ItemName: Name of the item.
  + Quantity: Number of items purchased.
  + UnitPrice: Price per unit of the item.
  + TotalPrice: Total price for the item.

**2. Relationships:**

* **Vendors and Contracts**:
  + One vendor can have multiple contracts (Have relationship).
* **Vendors and Vendor Performance**:
  + One vendor can have multiple performance records (Have relationship).
* **Departments and Budgets**:
  + Each department can have one budget (Has relationship).
* **Departments and Purchase Orders**:
  + Each department can place multiple purchase orders (Places relationship).
* **Vendors and Purchase Orders**:
  + One vendor can be associated with multiple purchase orders.
* **Purchase Orders and Purchase Order Items**:
  + Each purchase order can include multiple items (Have relationship).

**3. Keys:**

* Primary Keys (PK): Unique identifiers for each entity (e.g., VendorID, ContractID).
* Foreign Keys (FK): Attributes that establish relationships between entities (e.g., VendorID in Vendor Performance links to Vendors).

**4. Purpose:**

This ERD is designed to manage:

* Vendor details and performance.
* Contract tracking.
* Departmental budgets and purchase orders.
* Detailed purchase order items and their costs.