Detailed Technical Implementation Plan

Frontend Development

Tasks:

1. **Setup ReactJS Project**:
    - Initialize a new ReactJS project using Create React App.
    - Integrate TailwindCSS for styling.
2. **Develop UI Components**:
    - **Dashboard**: Overview of system metrics and notifications.
    - **Forms**: Reusable form components for creating and editing records.
    - **Tables**: Data display tables with sorting, filtering, and pagination.
    - **Modals**: For actions requiring confirmation or additional details.
    - **Notifications**: Alerts for success, error, and info messages.
3. **Implement State Management**:
    - Use Redux for state management.
    - Define actions, reducers, and selectors for each module.
4. **API Integration**:
    - Create Axios instances for API calls.
    - Develop service modules for making API requests.
    - Handle API responses and errors gracefully.
5. **Responsive Design**:
    - Ensure all components are mobile-friendly.
    - Use TailwindCSS utilities for responsive layouts.
6. **Testing**:
    - Write unit tests for individual components using Jest.
    - Write integration tests for complex interactions using React Testing Library.

Backend Development

Tasks:

1. **Setup Node.js/Laravel Project**:
    - Initialize a new Node.js or Laravel project.
    - Install required dependencies (Express/Koa for Node.js, Laravel packages).
2. **Database Schema Design**:
    - Design database schema in PostgreSQL.
    - Create ER diagrams for visual representation.
3. **Develop RESTful APIs**:
    - **Invoicing and Sales APIs**: Endpoints for creating, updating, retrieving, and deleting invoices and sales orders.

    - **Purchase APIs**: Endpoints for managing purchase orders, suppliers, and procurement processes.
    - **Inventory APIs**: Endpoints for managing stock levels, serial numbers, locations, and warehouses.
    - **Accounting APIs**: Endpoints for financial transactions, chart of accounts, and generating reports.
    - **User and Access Control APIs**: Endpoints for managing user roles, permissions, and authentication.
4. **Authentication and Authorization**:
    - Implement JWT-based authentication.
    - Role-based access control for securing endpoints.
5. **Integration with eBay APIs**:
    - Use eBay SDK to integrate Account, Analytics, Finance, Feed, Fulfillment, Inventory, Logistics, Post Order, and Return Management APIs.
    - Implement service modules to handle API interactions.
6. **Business Logic Implementation**:
    - Implement business logic for each module:
        - **Invoicing and Sales**: Validation, calculation of totals, discounts, and taxes.
        - **Purchases**: Supplier management, purchase order lifecycle.
        - **Inventory**: Stock level alerts, serial number tracking, warehouse operations.
        - **Accounting**: Transaction recording, multi-currency handling, financial reporting.
        - **User Management**: Role assignment, permission checks.
7. **Testing**:
    - Write unit tests for individual functions and methods using Mocha/Chai (Node.js) or PHPUnit (Laravel).
    - Write integration tests for API endpoints using Supertest (Node.js) or Laravel's built-in testing tools.

Module-Specific Implementation Details

1. Invoicing and Sales Modules

Backend:

- **Endpoints**:
    - `POST /invoices`: Create a new invoice.
    - `GET /invoices`: Retrieve a list of invoices.
    - `GET /invoices/:id`: Retrieve a specific invoice.
    - `PUT /invoices/:id`: Update an invoice.
    - `DELETE /invoices/:id`: Delete an invoice.
- **Business Logic**:

    - Calculate invoice totals, apply discounts, and taxes.
    - Validate product availability and pricing.
- **Database Tables**:
    - `invoices`
    - `invoice_items`
    - `customers`
    - `products`

Frontend:

- **Components**:
    - Invoice Form: For creating and editing invoices.
    - Invoice List: Displaying a list of invoices.
    - Invoice Details: Viewing invoice details.
- **State Management**:
    - Actions for fetching, creating, updating, and deleting invoices.
    - Selectors for accessing invoice data.

2. Purchase Modules

Backend:

- **Endpoints**:
    - `POST /purchases`: Create a new purchase order.
    - `GET /purchases`: Retrieve a list of purchase orders.
    - `GET /purchases/:id`: Retrieve a specific purchase order.
    - `PUT /purchases/:id`: Update a purchase order.
    - `DELETE /purchases/:id`: Delete a purchase order.
- **Business Logic**:
    - Manage supplier relationships and procurement processes.
    - Track purchase order status.
- **Database Tables**:
    - `purchases`
    - `purchase_items`
    - `suppliers`
    - `products`

Frontend:

- **Components**:
    - Purchase Order Form: For creating and editing purchase orders.
    - Purchase Order List: Displaying a list of purchase orders.
    - Purchase Order Details: Viewing purchase order details.

- **State Management**:
    - Actions for fetching, creating, updating, and deleting purchase orders.
    - Selectors for accessing purchase order data.

3. Inventory Modules

Backend:

- **Endpoints**:
    - `POST /inventory`: Add new inventory items.
    - `GET /inventory`: Retrieve a list of inventory items.
    - `GET /inventory/:id`: Retrieve specific inventory items.
    - `PUT /inventory/:id`: Update inventory items.
    - `DELETE /inventory/:id`: Delete inventory items.
- **Business Logic**:
    - Manage stock levels, serial numbers, and warehouse locations.
    - Implement safety stock alerts and advanced routing.
- **Database Tables**:
    - `inventory`
    - `inventory_locations`
    - `products`
    - `warehouses`

Frontend:

- **Components**:
    - Inventory Form: For adding and editing inventory items.
    - Inventory List: Displaying a list of inventory items.
    - Inventory Details: Viewing inventory details.
- **State Management**:
    - Actions for fetching, creating, updating, and deleting inventory items.
    - Selectors for accessing inventory data.

4. Accounting Modules

Backend:

- **Endpoints**:
    - `POST /transactions`: Create a new financial transaction.
    - `GET /transactions`: Retrieve a list of transactions.
    - `GET /transactions/:id`: Retrieve a specific transaction.
    - `PUT /transactions/:id`: Update a transaction.
    - `DELETE /transactions/:id`: Delete a transaction.
- **Business Logic**:

    - Record financial transactions, handle multi-currency, and generate reports.
- **Database Tables**:
    - `transactions`
    - `accounts`
    - `currencies`
    - `financial_reports`

Frontend:

- **Components**:
    - Transaction Form: For creating and editing financial transactions.
    - Transaction List: Displaying a list of transactions.
    - Financial Reports: Viewing balance sheets, general ledgers, etc.
- **State Management**:
    - Actions for fetching, creating, updating, and deleting transactions.
    - Selectors for accessing financial data.

5. Access Rights and Customization

Backend:

- **Endpoints**:
    - `POST /users`: Create a new user.
    - `GET /users`: Retrieve a list of users.
    - `GET /users/:id`: Retrieve a specific user.
    - `PUT /users/:id`: Update a user.
    - `DELETE /users/:id`: Delete a user.
    - `POST /roles`: Create a new role.
    - `GET /roles`: Retrieve a list of roles.
    - `GET /roles/:id`: Retrieve a specific role.
    - `PUT /roles/:id`: Update a role.
    - `DELETE /roles/:id`: Delete a role.
- **Business Logic**:
    - Manage user roles and permissions.
    - Secure endpoints based on user roles.
- **Database Tables**:
    - `users`
    - `roles`
    - `permissions`
    - `user_roles`

Frontend:

- **Components**:
  - User Management: Creating and managing users and roles.
  - Role Management: Assigning and managing permissions for roles.
- **State Management**:
  - Actions for fetching, creating, updating, and deleting users and roles.
  - Selectors for accessing user and role data.

6. Cloud Server Setup

Tasks:

- **Provision Cloud Infrastructure**:
  - Setup servers on AWS or Alicloud.
- **CI/CD Pipeline**:
  - Setup CI/CD pipelines using GitHub Actions or Jenkins.
- **Containerization**:
  - Dockerize applications for consistent deployment.
- **Environment Configuration**:
  - Configure development, staging, and production environments.
- **Deployment**:
  - Deploy applications to the cloud server.
- **Monitoring and Logging**:
  - Implement monitoring using Prometheus and Grafana.
  - Set up logging with the ELK stack.

7. eBay REST API Integration

Backend:

- **Endpoints**:
  - Create service modules to interact with eBay Account, Analytics, Finance, Feed, Fulfillment, Inventory, Logistics, Post Order, and Return Management APIs.
  - Handle authentication, error handling, and data parsing.
- **Business Logic**:
  - Implement business rules for synchronizing data with eBay.
- **Database Tables**:
  - `ebay_accounts`
  - `ebay_listings`
  - `ebay_orders`
  - `ebay_returns`

Frontend:

- **Components**:
  - eBay Account Management: Manage eBay account connections.

    - eBay Listings: View and manage eBay product listings.
    - eBay Orders: View and manage eBay orders.
    - eBay Returns: Handle eBay returns.
- **State Management**:
    - Actions for synchronizing data with eBay APIs.
    - Selectors for accessing eBay-related data.

Next Steps for Engineers

1. **Initial Setup**:
    - Set up the development environment for frontend and backend.
    - Provision cloud infrastructure on AWS/Alicloud.
2. **Frontend Development**:
    - Create the initial ReactJS project and integrate TailwindCSS.
    - Develop UI components and implement state management.
    - Integrate with backend APIs.
3. **Backend Development**:
    - Set up the Node.js/Laravel project and design the database schema.
    - Develop RESTful APIs and implement business logic for each module.
    - Integrate eBay APIs and handle data synchronization.
4. **Cloud Setup and Deployment**:
    - Set up CI/CD pipelines and configure containerization.
    - Deploy applications to the cloud and set up monitoring and logging.
5. **Module-Specific Development**:
    - Follow the detailed implementation plan for each module, ensuring all endpoints, business logic, and frontend components are developed and integrated.
6. **Testing and QA**:
    - Conduct unit and integration testing for frontend and backend.
    - Perform SIT and UAT for each module.
7. **Training and Documentation**:
    - Provide training sessions for the client on using the ERP system.
    - Prepare comprehensive documentation for system usage and administration.
