Vendor & Purchase Order Manager — Complete Project Guide

Table of Contents

---

1. Project Overview

Vendor & Purchase Order Manager is a full-stack MERN application for managing procurement operations. It tracks the entire lifecycle:

```
Vendor → Purchase Order → Delivery → Invoice → Payment → Reconciliation
```

Who uses it?

- **Admin** — Full control over everything
- **Manager** — Manages vendors & purchase orders
- **Accountant** — Handles invoices, payments, journals
- **Viewer** — Read-only access to key data

---

2. Tech Stack & Dependencies

Backend (Node.js + Express)

Package	What It Does
express	Web server framework — handles HTTP requests, routing, middleware
mongoose	MongoDB ORM — defines schemas, validates data, queries database
jsonwebtoken (JWT)	Creates and verifies login tokens for authentication
bcryptjs	Hashes passwords so they're never stored in plain text

cors	Allows the React frontend (port 5173) to talk to the backend (port 5000)
dotenv	Loads environment variables from `.env` file (DB URL, JWT secret)
express-rate-limit	Prevents abuse by limiting API calls (e.g., 100 requests/15 min)
multer	Handles file uploads (invoice PDFs, CSV files)
pdfkit	Generates PDF documents for purchase orders
exceljs	Creates Excel spreadsheets for data export
csv-parser	Reads CSV files for bulk vendor/PO import
node-cron	Schedules recurring tasks (check late deliveries every hour)
socket.io	Real-time communication (instant notifications)
nodemailer	Sends emails (payment confirmations, alerts)
swagger-jsdoc + swagger-ui-express	Auto-generates API documentation at `/api/docs`
google-auth-library	Verifies Google OAuth tokens for social login
speakeasy + qrcode	Two-factor authentication (2FA) support
node-cache	In-memory caching for frequently accessed data

Frontend (React + Vite)

Package	What It Does
react	UI library — builds the interface using components
react-dom	Renders React components into the browser DOM
react-router-dom	Client-side routing — navigates between pages without refresh
axios	HTTP client — makes API calls to the backend
recharts	Chart library — renders area charts, bar charts, line charts, pie charts
lucide-react	Icon library — provides 1000+ SVG icons
react-hot-toast	Toast notification popups (success/error messages)
socket.io-client	Connects to backend Socket.io for real-time updates
@react-oauth/google	Google login button component
tailwindcss	Utility-first CSS framework — styles everything with classes
vite	Build tool — fast development server with hot module replacement

---

3. Project Folder Structure

```
vendor system/
├── client/                          # React Frontend
│   ├── public/                      # Static files
│   ├── src/
│   │   ├── components/              # Reusable UI components
│   │   │   ├── Layout.jsx           # Main layout (sidebar + header)
│   │   │   ├── ProtectedRoute.jsx   # Auth guard wrapper
│   │   │   ├── StatsCard.jsx        # Dashboard stat cards
│   │   │   └── NotificationBell.jsx # Header notification dropdown
│   │   ├── context/                 # React Context providers
│   │   │   ├── AuthContext.jsx      # Authentication + RBAC permissions
│   │   │   └── ThemeContext.jsx     # Dark/light mode + color themes
│   │   ├── lib/
│   │   │   └── api.js               # Axios instance with auth headers
│   │   ├── pages/                   # One file per page/route
│   │   │   ├── Login.jsx
│   │   │   ├── Dashboard.jsx
│   │   │   ├── Vendors.jsx
│   │   │   ├── PurchaseOrders.jsx
│   │   │   ├── Invoices.jsx
│   │   │   ├── Payments.jsx
│   │   │   ├── Analytics.jsx
│   │   │   ├── KanbanBoard.jsx
│   │   │   ├── Contracts.jsx
│   │   │   ├── InventoryPage.jsx
│   │   │   ├── Budgets.jsx
│   │   │   ├── Forecast.jsx
│   │   │   ├── VendorCompare.jsx
│   │   │   ├── JournalEntries.jsx
│   │   │   ├── Reconciliation.jsx
│   │   │   ├── Users.jsx
│   │   │   ├── AuditLogs.jsx
│   │   │   └── AccessDenied.jsx
│   │   ├── App.jsx                  # Route definitions
│   │   ├── main.jsx                 # Entry point (providers wrap)
│   │   └── index.css                # Global styles + theme variables
│   ├── tailwind.config.js           # Tailwind CSS configuration
│   └── package.json
│
├── server/                          # Node.js Backend
│   ├── config/
│   │   ├── db.js                    # MongoDB connection
│   │   └── swagger.js               # API docs configuration
│   ├── middleware/
│   │   ├── auth.js                  # JWT token verification
│   │   ├── rbac.js                  # Role-Based Access Control
│   │   ├── errorHandler.js          # Global error handling
│   │   └── rateLimiter.js           # Request rate limiting
│   ├── models/                      # MongoDB schemas (20 models)
```

```
│   │   ├── User.js
│   │   ├── Vendor.js
│   │   ├── PurchaseOrder.js
│   │   ├── Invoice.js
│   │   ├── Payment.js
│   │   ├── Contract.js
│   │   ├── Budget.js
│   │   ├── Inventory.js
│   │   ├── JournalEntry.js
│   │   ├── AuditLog.js
│   │   ├── Notification.js
│   │   └── ... (9 more)
│   ├── controllers/                # Business logic (26 controllers)
│   ├── routes/                     # API endpoint definitions (21 route files)
│   ├── utils/
│   │   └── auditLogger.js          # Audit trail helper
│   ├── server.js                   # Entry point
│   ├── seed.js                     # Database seeder
│   ├── .env                        # Environment variables
│   └── package.json
```

---

4. How To Run

Prerequisites

- **Node.js** v18+ installed
- **MongoDB** running (local or MongoDB Atlas cloud)

Step 1: Backend

```
cd "vendor system/server"
npm install
# Create .env file with:
#   MONGO_URI=mongodb://localhost:27017/vendor-manager
#   JWT_SECRET=your-secret-key
#   PORT=5000
node seed.js          # Populate database with sample data
node server.js        # Start backend on port 5000
```

Step 2: Frontend

```
cd "vendor system/client"
npm install
npm run dev           # Start frontend on port 5173
```

Step 3: Open

- App: http://localhost:5173
- API Docs: http://localhost:5000/api/docs
- Login: admin@vendor.com / admin123

---

5. Database Design — MongoDB Models

5.1 User Model

File: `server/models/User.js`

Stores accounts with hashed passwords and role assignments.

Field	Type	Purpose
`name`	String	User's display name
`email`	String (unique)	Login identifier
`password`	String	Bcrypt-hashed password
`role`	Enum	`admin`, `manager`, `accountant`, `viewer`, `vendor`
`googleId`	String	For Google OAuth users
`authProvider`	Enum	`local` or `google`
`enable2FA`	Boolean	Two-factor auth toggle

How password hashing works:

```
// Before saving, the "pre-save" hook runs:
userSchema.pre('save', async function(next) {
    if (!this.isModified('password')) return next(); // Skip if password unchanged
    const salt = await bcrypt.genSalt(10);           // Generate random salt
    this.password = await bcrypt.hash(this.password, salt); // Hash password
    next();
});

// When logging in, we compare:
userSchema.methods.matchPassword = async function(entered) {
    return await bcrypt.compare(entered, this.password);
};
```

5.2 Vendor Model

File: `server/models/Vendor.js`

Field	Type	Purpose
`name`	String	Company name
`contactPerson`	String	Primary contact
`phone`, `email`	String	Contact details
`address`	String	Physical address
`gstTaxId`	String	Tax identification number

`rating`	Number (0-5)	Star rating
`performanceScore`	Number (0-100)	Auto-calculated score based on delivery/payment history
`itemPrices`	Array	Catalog of items with prices this vendor offers
`riskIndex`	Enum	`Low`, `Medium`, `High` — vendor reliability risk
`earlyPaymentDiscountPercentage`	Number	Discount for paying early

5.3 PurchaseOrder Model

File: `server/models/PurchaseOrder.js`

Field	Type	Purpose
`poNumber`	String (auto)	Auto-generated: `PO-0001`, `PO-0002`, etc.
`vendor`	ObjectId → Vendor	Link to vendor
`items`	Array	`[{name, qty, unitPrice, total}]`
`totalAmount`	Number (auto)	Sum of all item totals (calculated in pre-save)
`orderDate`	Date	When the order was placed
`expectedDeliveryDate`	Date	When delivery is expected
`actualDeliveryDate`	Date	When delivery actually happened
`status`	Enum	`Pending` → `Delivered` → `Cancelled`
`isLateDelivery`	Boolean (auto)	`true` if actual > expected delivery date
`isRecurring`	Boolean	Whether this PO auto-repeats
`recurringInterval`	Enum	`Weekly`, `Monthly`, `Quarterly`, `None`
`approvalStatus`	Enum	`Draft` → `Submitted` → `Approved` / `Rejected`
`approvalHistory`	Array	Audit trail of who approved/rejected and when

Auto-calculations in pre-save hook:

```
purchaseOrderSchema.pre('save', async function(next) {
    // 1. Calculate item totals
    this.items = this.items.map(item => {
        item.total = item.qty * item.unitPrice;
        return item;
    });
    this.totalAmount = this.items.reduce((sum, item) => sum + item.total, 0);

    // 2. Detect late delivery
    if (this.actualDeliveryDate && this.expectedDeliveryDate &&
        this.actualDeliveryDate > this.expectedDeliveryDate) {
```

```
        this.isLateDelivery = true;
    }

    // 3. Auto-generate PO number on creation
    if (this.isNew) {
        const count = await mongoose.model('PurchaseOrder').countDocuments();
        this.poNumber = `PO-${String(count + 1).padStart(4, '0')}`;
    }
    next();
});
```

5.4 Invoice Model

File: `server/models/Invoice.js`

Field	Type	Purpose
`purchaseOrder`	ObjectId → PO	Links invoice to its PO
`vendor`	ObjectId → Vendor	Which vendor issued it
`invoiceNumber`	String (unique)	e.g., `INV-2024-001`
`amount`	Number	Invoice amount
`paidAmount`	Number	How much has been paid
`outstandingAmount`	Number (auto)	`netPayable - paidAmount`
`paymentStatus`	Enum (auto)	`Unpaid` / `Partial` / `Paid`
`dueDate`	Date	Payment deadline
`taxableAmount`, `taxRate`, `taxAmount`	Number	Tax engine
`withholdingPercentage`, `withholdingAmount`	Number	Tax withholding
`netPayable`	Number (auto)	`amount + tax - withholding`
`currency`, `exchangeRate`	String/Number	Multi-currency support

Auto-calculated fields:

```
invoiceSchema.pre('save', function(next) {
    this.taxAmount = this.taxableAmount * (this.taxRate / 100);
    this.withholdingAmount = this.amount * (this.withholdingPercentage / 100);
    this.netPayable = this.amount + this.taxAmount - this.withholdingAmount;
    this.outstandingAmount = this.netPayable - this.paidAmount;

    // Auto-determine payment status
    if (this.paidAmount <= 0) this.paymentStatus = 'Unpaid';
    else if (this.paidAmount >= this.netPayable) this.paymentStatus = 'Paid';
    else this.paymentStatus = 'Partial';
```

```
    next();
});
```

5.5 Payment Model

File: `server/models/Payment.js`

Field	Type	Purpose
`invoice`	ObjectId → Invoice	Which invoice is being paid
`vendor`	ObjectId → Vendor	Payee
`amount`	Number	Payment amount
`paymentMethod`	Enum	`Card`, `UPI`, `Bank Transfer`, `Manual`
`transactionId`	String (auto)	Auto-generated: `TXN-...`
`paymentStatus`	Enum	`Pending`, `Success`, `Failed`, `Refunded`
`scheduledDate`	Date	For scheduled future payments
`discountApplied`	Number	Early payment discount amount
`idempotencyKey`	String	Prevents duplicate payments

5.6 Other Models

Model	File	Purpose
Contract	`Contract.js`	Vendor contracts with start/end dates, value, status
Budget	`Budget.js`	Department budgets with allocated/utilized amounts
Inventory	`Inventory.js`	Stock items with quantity, reorder point, unit cost
JournalEntry	`JournalEntry.js`	Double-entry accounting: debits + credits
AuditLog	`AuditLog.js`	Every system action logged (who, what, when)
Notification	`Notification.js`	User notifications (late delivery, overspend, etc.)
BatchPayment	`BatchPayment.js`	Group multiple payments together
Webhook	`Webhook.js`	External system integration endpoints
VendorLedger	`VendorLedger.js`	Running balance per vendor

---

6. Authentication & Authorization

6.1 How Login Works

```
┌──────────┐                    POST /api/auth/login                    ┌──────────┐
│  React   │ ──────────────────────────────────────────> │  Express  │
```

```
| Frontend |  { email, password }            | Backend   |
|          |                                 |           |
|          |  <------------------------------|           |
|          |  { token, user, role }          |           |
└──────────┘                                 └───────────┘

     |                                            |
     |  Stores token in localStorage              |  1. Find user by email
     |  Sets axios header:                        |  2. Compare hashed password
     |  Authorization: Bearer <token>             |  3. Generate JWT token
     |                                            |  4. Return user + token
```

JWT Token is a base64-encoded string containing:

```
{
    "id": "user_id_here",      // Identifies the user
    "iat": 1708876800,          // Issued at (timestamp)
    "exp": 1711468800           // Expires in 30 days
}
```

6.2 How Protected Routes Work

Backend middleware (`middleware/auth.js`):

```
const protect = async (req, res, next) => {
    // 1. Extract token from "Authorization: Bearer <token>" header
    const token = req.headers.authorization?.split(' ')[1];

    // 2. Verify token using JWT_SECRET
    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    // 3. Find user in database, attach to request
    req.user = await User.findById(decoded.id).select('-password');

    next(); // Allow the request to continue
};
```

6.3 Role-Based Access Control (RBAC)

Backend middleware (`middleware/rbac.js`):

```
// Only allow specified roles
const authorize = (...roles) => (req, res, next) => {
    if (!roles.includes(req.user.role)) {
        return res.status(403).json({ message: 'Access denied' });
    }
    next();
};

// Usage in routes:
```

```
router.get('/vendors', protect, authorize('admin', 'manager', 'viewer'), getVendors);
router.post('/vendors', protect, authorize('admin', 'manager'), createVendor);
```

Frontend permissions (`context/AuthContext.jsx`):

```
const ROLE_PERMISSIONS = {
    admin: {
        canViewDashboard: true, canViewVendors: true, canWriteVendors: true,
        canViewPurchaseOrders: true, canWritePurchaseOrders: true,
        canViewInvoices: true, canViewAnalytics: true, canViewUsers: true,
        // ... all true
    },
    manager: {
        canViewDashboard: true, canViewVendors: true, canWriteVendors: true,
        canViewInvoices: false,  // Can't see invoices
        canViewUsers: false,     // Can't manage users
        // ...
    },
    accountant: {
        canViewVendors: false,   // Can't see vendors
        canViewInvoices: true,   canWriteInvoices: true,
        canViewPayments: true,   canViewJournal: true,
        // Focused on financial pages
    },
    viewer: {
        // Read-only access to most pages, no write permissions
    },
};
```

How pages are protected (`App.jsx`):

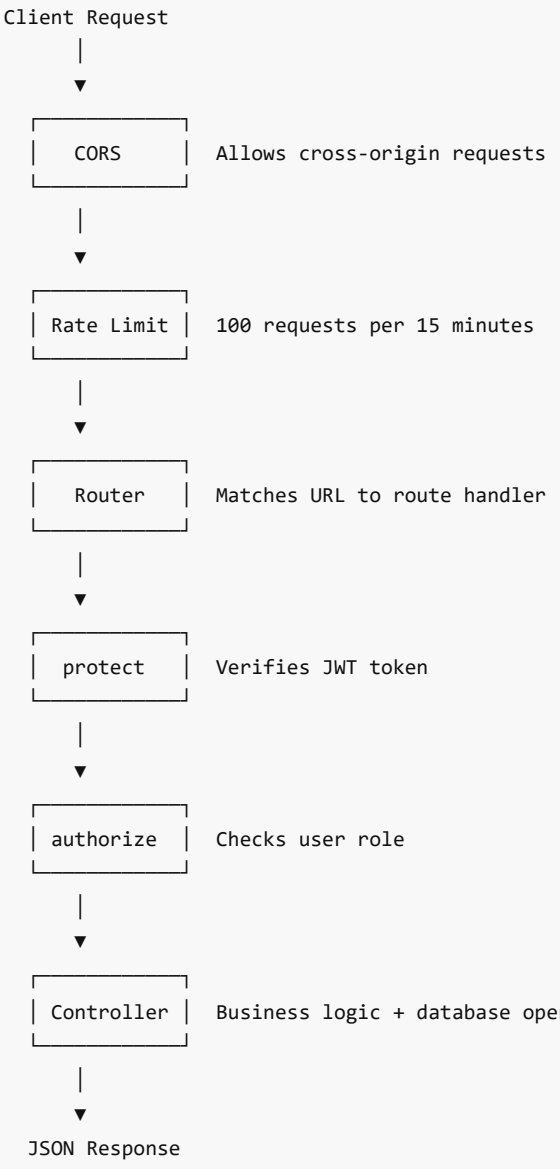
```
// Each route checks a permission key before rendering
const route = (path, perm, Component) => (
    <Route path={path} element={
        <ProtectedRoute>      {/* Must be logged in */}
            <RoleRoute permissionKey={perm}>  {/* Must have permission */}
                <Layout><Component /></Layout>
            </RoleRoute>
        </ProtectedRoute>
    } />
);

// Routes:
{route('/', 'canViewDashboard', Dashboard)}
{route('/vendors', 'canViewVendors', Vendors)}
{route('/invoices', 'canViewInvoices', Invoices)}
```

---

7. Backend — Server Architecture

7.1 Request Flow



7.2 How a Controller Works (Example: Create Vendor)

```
// controllers/vendorController.js
const createVendor = async (req, res) => {
    try {
        // 1. Extract data from request body
        const { name, contactPerson, phone, email, address } = req.body;

        // 2. Create vendor in MongoDB
        const vendor = await Vendor.create({
            name, contactPerson, phone, email, address
        });
```

```
        // 3. Log audit trail
        await logAudit(req.user._id, 'CREATE', 'Vendor', vendor._id);

        // 4. Send response
        res.status(201).json(vendor);
    } catch (error) {
        res.status(400).json({ message: error.message });
    }
};
```

7.3 Cron Jobs (Scheduled Tasks)

File: `server/server.js`

Schedule	Task	What It Does
Every hour	`generateNotifications()`	Checks for late deliveries, overdue invoices, creates notifications
Every midnight	Recurring PO check	Finds delivered recurring POs, auto-generates new ones
Every 6 hours	`checkReorderPoints()`	Checks inventory levels, creates POs for low-stock items
Every 30 min	`processScheduledPayments()`	Processes payments scheduled for today

---

8. Frontend — React Architecture

8.1 Component Hierarchy

```
main.jsx
├── GoogleOAuthProvider     (Google login support)
├── BrowserRouter           (URL routing)
├── ThemeProvider            (Dark mode + color themes)
├── AuthProvider             (Login state + permissions)
├── App                      (Route definitions)
│   ├── Login               (Public page)
│   └── ProtectedRoute      (Auth guard)
│       └── RoleRoute        (Permission check)
│           └── Layout       (Sidebar + Header)
│               ├── Sidebar navigation
│               ├── Theme picker
│               ├── NotificationBell
│               └── {Page Component}  (Dashboard, Vendors, etc.)
└── Toaster                  (Toast notifications)
```

8.2 How API Calls Work

File: `client/src/lib/api.js`

```
import axios from 'axios';

const api = axios.create({
    baseURL: '/api',  // Proxied to localhost:5000 by Vite
});

// Automatically attach JWT token to every request
api.interceptors.request.use((config) => {
    const token = localStorage.getItem('token');
    if (token) config.headers.Authorization = `Bearer ${token}`;
    return config;
});

export default api;
```

Usage in a page:

```
// Dashboard.jsx
const { data } = await api.get('/dashboard');  // GET /api/dashboard
setStats(data);  // Update component state with response data
```

8.3 How React Context Works

Context is React's way to share data between all components without passing props down every level.

```
┌──────────────────────────────────────┐
│ AuthProvider (wraps entire app)      │
│                                      │
│ Provides: user, token, role,         │
│           permissions, login(), logout() │
│                                      │
│  ┌──────────────────────────────┐    │
│  │ Any child component can access:  │    │
│  │   const { user, role } = useAuth() │ │
│  │                              │    │
│  │   if (permissions.canViewVendors)│    │
│  │       show Vendors page      │    │
│  └──────────────────────────────┘    │
└──────────────────────────────────────┘
```

8.4 How Tailwind CSS Works

Instead of writing CSS files, you apply utility classes directly:

```
<!-- Traditional CSS: -->
<div class="card">...</div>
/* card { background: white; border-radius: 16px; padding: 24px; } */

<!-- Tailwind CSS: -->
<div class="bg-white rounded-2xl p-6 shadow-card border border-slate-100">...</div>
```

Custom theme colors (defined via CSS variables):

```
/* index.css — changes when theme switches */
:root[data-accent="indigo"] {
    --primary-500: 99 102 241;  /* Purple-blue */
}
:root[data-accent="emerald"] {
    --primary-500: 16 185 129;  /* Green */
}
```

```
/* tailwind.config.js — references the variables */
colors: {
    primary: {
        500: withOpacity('--primary-500'),  // Uses CSS variable
    }
}
```

Now `bg-primary-500` automatically changes color when the theme switches!

---

9. Every Feature Explained

9.1 Dashboard

Files: `Dashboard.jsx` + `dashboardController.js`

Shows overview cards:

- Total Vendors, Purchase Orders, Pending Payments, Overdue Payments, Late Deliveries, Outstanding Amount
- **Monthly Purchase Trend** — Area chart showing spending over last 12 months
- **Top Vendor** — Highest performing vendor by score
- **Vendor-wise Expense** — Horizontal bar chart of spending per vendor

9.2 Vendors

Files: `Vendors.jsx` + `vendorController.js` + `Vendor.js`

Full CRUD for suppliers:

- Add/Edit/Delete vendors with contact details, tax ID
- View vendor performance score (auto-calculated from delivery history)
- Risk index classification (Low/Medium/High)
- Item price catalog per vendor

9.3 Purchase Orders

Files: `PurchaseOrders.jsx` + `poController.js` + `PurchaseOrder.js`

- Create multi-item orders linked to vendors
- Auto-generated PO numbers (PO-0001, PO-0002...)
- Status flow: Pending → Delivered → Cancelled
- Approval workflow: Draft → Submitted → Approved/Rejected
- Recurring order settings (Weekly/Monthly/Quarterly)
- Expected vs actual delivery date tracking

- PDF export of PO documents

9.4 Invoices

Files: `Invoices.jsx` + `invoiceController.js` + `Invoice.js`

- Link invoices to purchase orders
- Track payment status: Unpaid → Partial → Paid
- Auto-calculate outstanding = amount + tax - withholding - paid
- File upload for invoice documents
- Multi-currency support with exchange rates
- Tax engine (taxable amount, tax rate, withholding)

9.5 Payments

Files: `Payments.jsx` + `paymentController.js` + `Payment.js`

- Record payments against invoices
- Multiple methods: Card, UPI, Bank Transfer, Manual
- Auto-generated transaction IDs
- Scheduled future payments (cron processes them)
- Early payment discount calculation
- Payment approval workflow
- Idempotency keys prevent duplicate payments

9.6 Kanban Board

File: `KanbanBoard.jsx`

Visual drag-and-drop board for PO status tracking:

```
┌──────────┐  ┌──────────┐  ┌──────────┐
│  Pending  │  │ Delivered │  │ Cancelled │
│           │  │           │  │           │
│  PO-0001  │  │  PO-0003  │  │  PO-0005  │
│  PO-0002  │  │  PO-0004  │  │           │
└──────────┘  └──────────┘  └──────────┘
```

9.7 Analytics

Files: `Analytics.jsx` + `analyticsController.js`

4 analytics views:

1. **Payment Aging** — Pie chart (0-30, 31-60, 61-90, 90+ days)
2. **Vendor Spend** — Bar chart of top spenders
3. **Monthly PO Growth** — Line chart with growth %
4. **Vendor Reliability** — Table with scores, on-time %, risk index

9.8 Contracts

Files: `Contracts.jsx` + `contractController.js` + `Contract.js`

- Vendor contracts with start/end dates
- Contract value and status tracking
- Renewal reminders

9.9 Inventory

Files: `InventoryPage.jsx` + `inventoryController.js` + `Inventory.js`

- Track stock quantities per item
- Reorder point alerts (cron checks every 6 hours)
- Auto-create PO when stock falls below reorder level
- Unit cost tracking

9.10 Budgets

Files: `Budgets.jsx` + `budgetController.js` + `Budget.js`

- Department-level budget allocation
- Track utilized vs remaining budget
- Budget exceeded notifications

9.11 Forecasting

File: `Forecast.jsx` + `forecastController.js`

- Spending prediction based on historical data
- Trend analysis and projections

9.12 Vendor Compare

File: `VendorCompare.jsx`

- Side-by-side comparison of vendor metrics
- Performance scores, total spend, delivery reliability

9.13 Journal Entries

Files: `JournalEntries.jsx` + `journalController.js` + `JournalEntry.js`

- Double-entry accounting (every transaction has a debit and credit)
- Links to invoices and payments
- Account-level tracking

9.14 Reconciliation

Files: `Reconciliation.jsx` + `reconciliationController.js`

- Match payments to invoices
- Identify discrepancies
- Reconciliation status tracking

9.15 User Management

Files: `Users.jsx` + `userController.js`

- Admin can create/edit/delete users
- Assign roles (admin, manager, accountant, viewer)
- View user activity

9.16 Audit Logs

Files: `AuditLogs.jsx` + `auditController.js` + `AuditLog.js`

Every system action is logged:

```
{
    "user": "Admin",
    "action": "CREATE",
    "entity": "PurchaseOrder",
    "entityId": "PO-0015",
    "timestamp": "2026-02-25T10:30:00Z"
}
```

9.17 Notifications

File: `NotificationBell.jsx` + `notificationController.js`

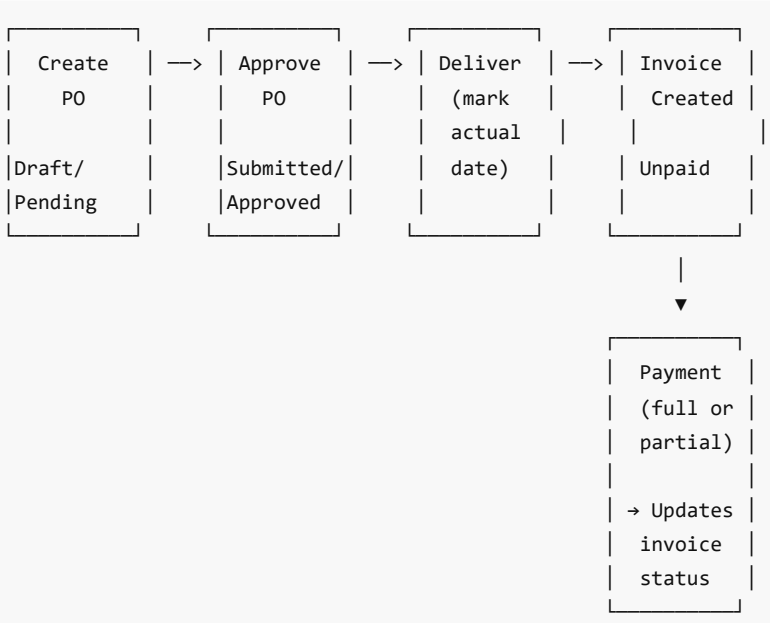
Real-time bell notifications for:

- Late deliveries
- Overdue invoices
- Overspending alerts
- Budget exceeded
- PO approved/rejected
- Payment received

---

10. Business Logic & Automated Systems

10.1 Complete Order Lifecycle



10.2 Vendor Performance Score Calculation

```
// vendorController.js — when fetching vendor detail
const totalOrders = await PurchaseOrder.countDocuments({ vendor: id });
```

```
const lateOrders = await PurchaseOrder.countDocuments({
    vendor: id, isLateDelivery: true
});
const onTimeRate = totalOrders > 0 ? ((totalOrders - lateOrders) / totalOrders) : 0;

const performanceScore = Math.round(
    (onTimeRate * 0.6 + (vendor.rating / 5) * 0.4) * 100
);
// 60% weight on delivery reliability, 40% on star rating
```

10.3 Recurring Orders (Cron)

```
// server.js — runs at midnight daily
cron.schedule('0 0 * * *', async () => {
    // Find recurring POs that have been delivered
    const recurringPOs = await PurchaseOrder.find({
        isRecurring: true,
        recurringInterval: { $ne: 'None' },
        status: 'Delivered'
    });

    for (const po of recurringPOs) {
        const intervalDays = { Weekly: 7, Monthly: 30, Quarterly: 90 };
        const daysSinceOrder = (Date.now() - po.orderDate) / (1000 * 60 * 60 * 24);

        if (daysSinceOrder >= intervalDays[po.recurringInterval]) {
            // Create new PO with same items/vendor
            const newPO = new PurchaseOrder({
                vendor: po.vendor,
                items: po.items,
                isRecurring: true,
                recurringInterval: po.recurringInterval,
            });
            await newPO.save();
        }
    }
});
```

---

11. AI Features

11.1 Best Vendor Suggestion

Endpoint: `GET /api/vendors/suggest?item=Paper`

How it works:

1. Searches all vendors whose `itemPrices` catalog contains the requested item
2. For each matching vendor, examines delivery history and pricing
3. Ranks by composite score: **price (lower = better) + reliability (higher = better)**
4. Returns the top recommendation with reasoning

11.2 Overspending Detection

Endpoint: `POST /api/purchase-orders/overspending-check`

How it works:

1. Takes an item name and current price
2. Queries all historical POs containing that same item
3. Calculates the **average historical price**
4. If current price is significantly above average, flags it as overspending
5. Returns percentage difference and suggested fair price

---

12. API Reference

Authentication

Method	Endpoint	Description
POST	`/api/auth/login`	Login with email/password → returns JWT token
POST	`/api/auth/register`	Create new user account
GET	`/api/auth/me`	Get current logged-in user

Vendors

Method	Endpoint	Description
GET	`/api/vendors`	List all vendors
POST	`/api/vendors`	Create vendor
GET	`/api/vendors/:id`	Get vendor detail + performance
PUT	`/api/vendors/:id`	Update vendor
DELETE	`/api/vendors/:id`	Delete vendor
GET	`/api/vendors/suggest?item=X`	AI: suggest best vendor for item

Purchase Orders

Method	Endpoint	Description
GET	`/api/purchase-orders`	List all POs (filterable)
POST	`/api/purchase-orders`	Create new PO
PUT	`/api/purchase-orders/:id`	Update PO
DELETE	`/api/purchase-orders/:id`	Delete PO
POST	`/api/purchase-orders/overspending-check`	AI: check overspending

Invoices

Method	Endpoint	Description
GET	`/api/invoices`	List all invoices
POST	`/api/invoices`	Create invoice linked to PO
PUT	`/api/invoices/:id`	Update invoice
DELETE	`/api/invoices/:id`	Delete/archive invoice

Payments

Method	Endpoint	Description
GET	`/api/payments`	List all payments
POST	`/api/payments`	Record a payment
PUT	`/api/payments/:id`	Update payment

Dashboard & Analytics

Method	Endpoint	Description
GET	`/api/dashboard`	Dashboard stats + charts data
GET	`/api/analytics/payment-aging`	Payment aging breakdown
GET	`/api/analytics/vendor-spend`	Spend per vendor
GET	`/api/analytics/monthly-growth`	Monthly PO growth trend
GET	`/api/analytics/vendor-reliability`	Vendor reliability rankings

Other Endpoints

Method	Endpoint	Description
GET/POST	`/api/budgets`	Budget management
GET/POST	`/api/contracts`	Contract management
GET/POST	`/api/inventory`	Inventory management
GET/POST	`/api/journal-entries`	Journal entries
GET	`/api/reconciliation`	Reconciliation data
GET	`/api/notifications`	User notifications
GET	`/api/audit-logs`	Audit trail
GET/POST	`/api/users`	User management
GET/POST	`/api/export/*`	Excel export
POST	`/api/import/csv`	CSV import

GET	`/api/health`	Server health check

---

13. Key Design Patterns

13.1 Pre-Save Hooks (Mongoose Middleware)

Code that runs automatically before saving a document:

```
schema.pre('save', function(next) {
    // Auto-calculate fields, validate data, generate IDs
    next();
});
```

Used in: PurchaseOrder (total, PO number, late detection), Invoice (tax, outstanding), Payment (transaction ID), User (password hashing)

13.2 React Context Pattern

Share state across all components without prop drilling:

```
// Create context
const AuthContext = createContext();

// Provider wraps the app
<AuthContext.Provider value={{ user, login, logout }}>
    {children}
</AuthContext.Provider>

// Any component can consume
const { user } = useContext(AuthContext);
```

13.3 Protected Route Pattern

Components that redirect to login if not authenticated:

```
const ProtectedRoute = ({ children }) => {
    const { token, loading } = useAuth();
    if (loading) return <Spinner />;
    if (!token) return <Navigate to="/login" />;
    return children;
};
```

13.4 Audit Trail Pattern

Every important action is logged:

```
await logAudit(userId, 'CREATE', 'PurchaseOrder', poId, { details });
```

---

14. Glossary of Technologies

Term	What It Is
MERN	MongoDB + Express + React + Node.js — full-stack JavaScript
REST API	Architectural style — uses HTTP methods (GET, POST, PUT, DELETE) for CRUD
JWT	JSON Web Token — encrypted token for stateless authentication
Mongoose	ODM (Object Data Modeling) library that maps MongoDB documents to JavaScript objects
Middleware	Functions that run between receiving a request and sending a response
RBAC	Role-Based Access Control — different permissions for different user types
CRUD	Create, Read, Update, Delete — the four basic data operations
Context API	React's built-in way to pass data through component trees
CSS Variables	Custom properties like `--primary-500` that can change dynamically
Cron Job	Scheduled task that runs at specific times (like a timer)
Socket.io	Library for real-time, bidirectional communication between client and server
Axios	Promise-based HTTP client for making API calls
Vite	Modern build tool that provides fast dev server with hot module replacement
Pre-save Hook	Mongoose lifecycle method that runs before a document is saved to MongoDB
Idempotency	Ensuring the same operation produces the same result if repeated

---

End of Document

Generated: February 25, 2026 Project: Vendor & Purchase Order Manager v5.0.0