Invoice Software — Wireframe & Workflow

> Full step-by-step product wireframe and technical workflow for building an invoicing system like Zoho, using **React (frontend)** and **Node.js + MySQL (backend)**. This document covers screens, user flows, database design, API contract ideas, implementation tips, deployment, security, and an iterative roadmap from MVP to advanced features.

---

1) Project Overview & Goals

Goal: Build a professional invoice/accounting-lite product that supports quotations/estimates, invoices, recurring billing, payments, expenses, multi-company profiles, reporting, and integrations (email, payment gateways, accounting exports).

Target users: Freelancers, small businesses, accountants.

Primary constraints/assumptions: - Single-tenant or multi-company within one installation (configurable). - REST API (JSON) for frontend; consider GraphQL later. - File storage: local / cloud (S3) for attachments & PDFs. - PDF generation via Puppeteer (server-side) or headless Chrome.

---

2) User Roles & Permissions

1. **Super Admin** — full system control, billing, global settings.
2. **Company Admin** — manage company data, users, templates, finance settings.
3. **Accountant / Manager** — create/edit invoices, reports, reconcile payments.
4. **Sales / Staff** — create quotes, customers, basic invoices.
5. **Read-only Auditor** — view access to records and reports.

Permission model: role-based + granular permission toggles (create/edit/delete/export).

---

3) Core Modules / Screens (Wireframe list)

Start with MVP screens, then add advanced ones.

MVP Screens (must-have)

1. **Sign In / Sign Up / Forgot Password**
2. **Company Onboarding / Setup wizard** (company name, tax IDs, invoice numbering)
3. **Dashboard** — KPIs (Unpaid invoices, Overdue, MRR, Recent activity)
4. **Customers (Contacts)** — list, search, import/export
5. **Products / Services (Catalog)** — CRUD, tax rates, unit price
6. **Create Invoice** — line items, taxes, discounts, preview, save/draft/send
7. **Invoice List & Filters** — status: Draft, Sent, Viewed, Partially Paid, Paid, Overdue
8. **Invoice Details** — PDF preview, payment links, history / notes
9. **Payments** — record manual payments, reconcile, integrate gateway

10. **Estimates / Quotes** — convert to invoice
11. **Reports** — Sales by period, Top customers, Tax summary
12. **Settings** — Company details, Invoice template, Payment accounts
13. **PDF Export / Email Send** — templates and logs

Advanced Screens (phase 2+)

- **Recurring Invoices / Auto-billing management**
- **Expenses** (vendor bills, attachments)
- **Bank Reconciliation** (CSV import, match transactions)
- **Credit Notes / Refunds**
- **User & Role Management**
- **APIs & Webhooks** (for integrations)
- **Integrations screen** (Stripe, PayPal, Zoho Books export, QuickBooks export)
- **Audit Trail / Activity log**
- **Multi-company switcher**
- **Mobile-friendly app shell**

---

4) Screen-by-Screen Wireframe Details (Step-by-step flows)

4.1 Sign In / Onboarding

- Inputs: email, password, 2FA option.
- On first sign-up: onboarding wizard: company name, currency, default tax rate, invoice numbering format (prefix, next number), logo upload.

4.2 Dashboard

- Components: KPI cards (Total Due, Overdue, This Month Sales), charts (invoices by status, cashflow timeline), recent activity feed, quick actions (Create Invoice, Create Customer, Record Payment).

4.3 Customers Page

- Table with columns: Name, Email, Balance, Last Invoice Date, Actions (view/edit/invoice).
- Bulk import (CSV), export, tags, custom fields.

4.4 Product Catalog

- Columns: SKU, Name, Rate, Tax, Income Account, Actions.
- Option to toggle inventory tracking (if later required).

4.5 Create Invoice (detailed)

- Header: Customer select (search/autocomplete), invoice date, due date, invoice number (auto or manual), currency.
- Line items: product select (fills description/rate), quantity, unit price, discount (per-line or global), tax per line or item-level tax.
- Totals sidebar: subtotal, tax breakdown, discounts, rounding, total.
- Footer: Terms & conditions (template), Notes to customer, Attachments.

- Actions: Save Draft, Save & Send (email with template), Preview PDF, Mark as Sent, Convert from Estimate.

4.6 Invoice Details

- Left: PDF viewer / preview, right: timeline (sent/viewed/paid), quick actions (Email, Download PDF, Record Payment, Duplicate, Add Credit Note).

4.7 Payments

- Screen to record manual payments with method (Bank Transfer, Cash, Card), amount, date, receipt file upload.
- View list of payments, filter by invoice/customer/date.

4.8 Estimates

- Same UI as invoices but with statuses (Draft, Sent, Accepted, Rejected). Option to convert to invoice with mapping of fields.

4.9 Reports

- Date picker (custom range), charts and table exports (CSV/PDF).

4.10 Settings

- Company profile (address, GST/VAT number), email settings (SMTP, default from), payment gateway credentials, tax codes, invoice templates, numbering format, localization (currency, date format, number format), email templates.

---

5) Data Model (ER Diagram textual)

Key tables (columns summary):

- **users**: id, name, email, password_hash, role, company_id, is_active, created_at
- **companies**: id, name, currency, address, tax_id, logo_url, invoice_prefix, next_invoice_number, settings(json)
- **customers**: id, company_id, name, email, billing_address, shipping_address, phone, tax_number, custom_fields(json)
- **products**: id, company_id, sku, name, description, unit_price, tax_id, income_account, created_at
- **tax_rates**: id, company_id, name, percentage, is_compound
- **invoices**: id, company_id, customer_id, invoice_number, status, issue_date, due_date, currency, subtotal, tax_total, discount_total, total, balance_due, notes, terms, pdf_url, metadata(json), created_by
- **invoice_items**: id, invoice_id, product_id, description, quantity, unit_price, discount, tax_amount, line_total
- **payments**: id, invoice_id, company_id, amount, method, reference, received_at, note, reconciled
- **estimates**: (similar to invoices)
- **recurring_invoices**: id, invoice_template_id, interval, next_run, active
- **expenses**: id, company_id, vendor, amount, date, category, attachment_url
- **activity_logs**: id, company_id, user_id, action, target_type, target_id, timestamp
- **attachments**: id, company_id, attached_to_type, attached_to_id, url, filename

Indexes: foreign keys on company_id, customer_id, invoice_id. Use transactions for writes that touch invoice + items + payments.

---

6) Minimal SQL schema snippets (starter)

```
CREATE TABLE companies (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  currency VARCHAR(8) DEFAULT 'INR',
  logo_url VARCHAR(1024),
  invoice_prefix VARCHAR(50),
  next_invoice_number BIGINT DEFAULT 1,
  settings JSON,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE customers (
  id INT AUTO_INCREMENT PRIMARY KEY,
  company_id INT NOT NULL,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255),
  billing_address TEXT,
  phone VARCHAR(50),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (company_id) REFERENCES companies(id)
);

CREATE TABLE invoices (
  id BIGINT AUTO_INCREMENT PRIMARY KEY,
  company_id INT NOT NULL,
  customer_id INT,
  invoice_number VARCHAR(100) NOT NULL,
  status VARCHAR(50) DEFAULT 'draft',
  issue_date DATE,
  due_date DATE,
  subtotal DECIMAL(18,2) DEFAULT 0,
  tax_total DECIMAL(18,2) DEFAULT 0,
  total DECIMAL(18,2) DEFAULT 0,
  pdf_url VARCHAR(1024),
  metadata JSON,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

---

7) API Design (REST) — essential endpoints

Authentication: `POST /api/auth/login`, `POST /api/auth/register`, `POST /api/auth/refresh`

Company & Users - `GET /api/companies/:id` - `PUT /api/companies/:id` - `GET /api/users` (admin)

Customers - `GET /api/customers?search=&page=` - `POST /api/customers` - `GET /api/customers/:id` - `PUT /api/customers/:id` - `DELETE /api/customers/:id`

Products - `GET /api/products` - `POST /api/products`

Invoices - `GET /api/invoices?status=&page=&from=&to=` - `POST /api/invoices` (creates invoice + items + optional auto-generate pdf) - `GET /api/invoices/:id` - `PUT /api/invoices/:id` - `POST /api/invoices/:id/send` (email + log) - `POST /api/invoices/:id/payments` (record payment) - `GET /api/invoices/:id/pdf` (download)

Payments - `GET /api/payments` - `POST /api/payments`

Reports - `GET /api/reports/sales?from=&to=`

Webhooks & Integrations - `POST /api/webhooks/payment` (incoming from gateway)

All endpoints must use company_id from JWT or request context to enforce data separation.

---

8) Frontend Architecture (React)

- Use React with functional components and hooks.
- Routing: React Router / Next.js (if using SSR)
- State management: Redux Toolkit or React Query for server state + local UI state in component hooks.
- Forms: react-hook-form + yup for validation.
- UI Kit: Tailwind CSS + component library (Radix/shadcn or Material UI) for fast UI.
- PDF preview: iframe to `/api/invoices/:id/pdf` or embed PDF viewer.
- File uploads: direct-to-S3 or backend upload endpoint storing file and returning URL.

Component breakdown: - App shell (Navbar, Sidebar, Permissions-aware menu) - Pages: Dashboard, Customers, Products, InvoicesList, InvoiceForm, InvoiceView, Payments, Reports, Settings - Shared components: Table, Modal, DatePicker, Select (async), Autocomplete customer/product, CurrencyInput, Toast notifications

UX details: - Inline editing for invoice items. - Autosave drafts to prevent data loss. - Bulk actions (send, export, mark paid). - Powerful filtering & column customization.

---

9) PDF Generation & Emailing

- Generate PDF server-side using headless Chrome (Puppeteer) with an HTML template matching company invoice template.
- Save PDF to storage and store `pdf_url` on `invoices` table.
- Email sending: queue emails via a job queue (Bull/Redis) to avoid blocking requests.
- Email templates: HTML with placeholders (merge tags) and a plain-text fallback.

---

10) Payments & Gateways

- Integrate Stripe/PayPal/ Razorpay for payment links & webhooks.
- Create `payment_intents` or `checkout_sessions` server-side and return client tokens.
- On webhook receive, validate signature, then record payment and reconcile invoice balance.

---

11) Background Jobs / Cron

- Use BullMQ (Redis) or node-cron for:
- recurring invoices generation
- send payment reminders (email/SMS)
- PDF generation (if heavy)
- daily reporting / backup tasks

---

12) Security & Compliance

- Authentication: JWT with refresh tokens + optional 2FA.
- Authorization: RBAC + per-company scoping.
- Data validation & sanitization on server.
- Rate limiting, CORS, helmet for security headers.
- PCI: Do not store raw card data; use tokenized gateway.
- Backup & retention: regular DB backups; attachments on S3 with lifecycle rules.

---

13) Testing Strategy

- Unit tests: Jest for backend and frontend logic.
- Integration tests: Supertest for APIs.
- E2E tests: Playwright or Cypress for critical flows (create invoice, pay invoice).
- CI: GitHub Actions pipeline for lint, tests, build, and deployment.

---

14) Deployment & Scalability

- Containerize (Docker). Use managed DB (RDS / Cloud SQL) for production. Use S3 for file storage and Redis for job queue.

- Horizontal scale stateless Node.js servers behind a load balancer. Stick to connection pools for MySQL.

---

15) Roadmap / Milestones (step-by-step iteration)

Phase 1 — MVP (4–8 weeks) - Auth, onboarding, customers, products, basic invoices, PDF generation, send email, basic payments (manual), dashboard, basic reports.

Phase 2 — Core features (8–12 weeks) - Recurring invoices, payment gateway integration, estimates, credit notes, attachments, user roles.

Phase 3 — Advanced (12+ weeks) - Bank reconciliation, accountant features, audit trail, mobile responsive polish, multi-currency multi-company, API & webhooks, marketplace integrations.

---

16) Example: Step-by-step flow to implement "Create & Send Invoice" (developer tasks)

1. Design DB schema for invoices + items + customers + companies.
2. Build endpoint `POST /api/invoices` that validates payload and performs a transactional insert of invoice + items.
3. Implement invoice number generation service using `companies.next_invoice_number` with row-level lock to avoid duplicates.
4. On create, queue a job to generate PDF and save `pdf_url`.
5. Return invoice resource to client immediately with `status: draft`.
6. Implement `POST /api/invoices/:id/send` which:
7. changes status to sent
8. generates email body using template and merge fields
9. attaches invoice PDF (generate synchronously if missing)
10. queues email send job
11. On webhook from payment gateway, verify signature and record payment using `POST /api/webhooks/payment`.
12. Update invoice balance and set status to `paid` if fully paid; log activity.

---

17) UX / Wireframe Tips

- Keep create flows minimal — customer selection, 1-3 line items visible by default.
- Use progressive disclosure (advanced tax options hidden under an "advanced" toggle).
- Autosave drafts every X seconds.
- Friendly invoice numbering preview and example PDF preview while editing.
- Provide keyboard shortcuts for power users (create new invoice: `g i`).

---

18) Extras & Modern Features (consider later)

- Smart suggestions (frequently used items, suggested due dates)

- Multi-language templates
- OCR to scan bills/receipts
- AI-powered category suggestions for expenses
- Marketplace for extensions (connectors)

---

19) Deliverables checklist (what to build & test)

- [ ] Auth & onboarding
- [ ] Company settings & invoice numbering
- [ ] Customers CRUD + import/export
- [ ] Products CRUD
- [ ] Create invoice UI + API
- [ ] PDF generation & storage
- [ ] Send email & templates
- [ ] Record payments + reconcile
- [ ] Dashboard & basic reports
- [ ] Unit + integration tests
- [ ] CI/CD + deployment

---

20) Next steps I can help with

- Generate React page-level wireframes (component-level JSX + Tailwind) for any screen.
- Create the exact SQL schema and migration scripts for MySQL.
- Provide a starter repo structure (file tree) for frontend & backend.
- Draft the Puppeteer HTML template for invoice PDF.

If you want, tell me which screen to start building first and I will produce component-level React code and the matching backend endpoints and SQL migrations.