Goal: Export to PDF must download a file (no print dialog), exactly match the on-screen quote preview template, and keep text selectable/copyable. Implement a single, robust server-side HTML-to-PDF pipeline using headless Chromium (Puppeteer) and retire ad-hoc or rasterizing paths.

Do the following precisely and only touch relevant files:

1) Add server-side HTML-to-PDF download endpoint

- Install puppeteer and minimal deps.

- In server/routes.ts, replace the current /api/generate-pdf implementation (PDFKit) with Puppeteer:

- Accept POST { quoteData } (include filename optional).

- Render the same HTML used by the preview using a dedicated server-renderable template.

- Use page.setContent(renderedHTML, { waitUntil: 'networkidle0' }) or page.goto(print URL) and ensure fonts are fully loaded before page.pdf.

- Call page.pdf({ format: 'A4', printBackground: true, margin: { top: '12mm', right: '12mm', bottom: '12mm', left: '12mm' } }).

- Return the PDF buffer with headers:

- Content-Type: application/pdf

- Content-Disposition: attachment; filename="quote.pdf"

- Remove PDFKit usage and types to avoid confusion.

2) Create a shared server-renderable template that exactly matches QuotePreview

- Extract the JSX from client/src/components/QuotePreview.tsx into a shared isomorphic template function that returns an HTML string given quoteData. Create at server/html/quoteTemplate.ts.

- Do not rasterize. Preserve all text as regular HTML text. No canvases.

- Include identical structure and CSS so the server-rendered PDF is pixel-consistent with client preview.

- Create a minimal CSS bundle for print (server/html/quote.css). Use inline <style> or embed CSS in the HTML string for simplicity. Enable color fidelity for print using:

- \* { -webkit-print-color-adjust: exact; print-color-adjust: exact; }

- Replace any @assets imports with absolute paths served from /public or data URLs to ensure Puppeteer can load them on the server.

- Ensure page numbers and all 5 sections (Cover, Intro + Diagram, BOM + Costs, Payment Terms, IP + Contact) render identically.

3) Unify data model fields across client/server

- Standardize item fields to match shared/schema.ts:

- BOM: { no, partNumber, productDescription, quantity, unitPrice?, totalPrice? }

- Costs: { productDescription, quantity, unitPrice, totalPrice, isDiscount }

- Remove any use of “description” for BOM (must be productDescription) and ensure both preview and server template use the same keys.

4) Frontend integration: one-click download button

- In client/src/components/QuotePreview.tsx, update handleExport('pdf') to:

- Build quoteData from current form state (already done).

- POST to /api/generate-pdf and stream the Blob to a downloaded file programmatically (no window.print):

const res = await fetch('/api/generate-pdf', { method: 'POST', headers: { 'Content-Type': 'application/json' }, body: JSON.stringify({ quoteData, filename: suggestedName }) });

const blob = await res.blob();

const url = URL.createObjectURL(blob);

const a = document.createElement('a');

a.href = url;

a.download = suggestedName || 'quote.pdf';

document.body.appendChild(a);

a.click();

a.remove();

URL.revokeObjectURL(url);

- Keep the “Word” export stub unchanged for now or hide it behind a feature flag.

5) Remove conflicting/legacy paths

- Remove or disable print-quote.tsx auto window.print flow.

- Remove any html2canvas/html2pdf.js/jspdf export paths (these rasterize and break text selection). Keep libraries in package.json only if still used elsewhere; otherwise, remove.

- Remove server PDFKit route and any references.

6) Fonts, images, and assets in Puppeteer

- Serve images (logos, frames, diagrams) from /public/assets and reference them with absolute URLs relative to server (e.g., /assets/emet-logo.png). Ensure the Express static middleware is configured if needed.

- Ensure web fonts load before generating PDF:

- After page.setContent, wait for fonts:

await page.evaluate(async () => { if (document.fonts?.ready) { await document.fonts.ready; } });

- Alternatively, waitUntil: 'networkidle0' after injecting CSS.

- Set printBackground: true in page.pdf to preserve branding colors.

- Verify text remains selectable in the generated PDF.

7) Consistency & cleanup

- Refactor any remaining mismatched field names.

- Remove noisy console.log in production paths; guard logs by NODE\_ENV.

- Add try/catch with sensible error messages on both client and server.

- Update package.json scripts if needed, and ensure dev/start still work on Replit.

8) Acceptance criteria (run and verify)

- Clicking “Export to PDF” downloads a file directly (no print dialog).

- The PDF visually matches the on-screen preview across all 5 sections.

- Text in the PDF is selectable and copyable (verify in Acrobat/Preview).

- BOM and Costs totals, currency formatting, and visibility toggles match.

- Images and brand colors appear correctly in the PDF.

- Endpoint handles large BOM tables with automatic page breaks.

9) Provide a concise diff summary and file list of changes.

- List all created/modified/deleted files.

- Highlight how the shared template mirrors the client preview.

- Note remaining TODOs (e.g., Word export).

When done, run the app, generate a sample PDF with dummy data, and confirm selectable text and exact visual fidelity. Provide the sample PDF in the repo root for review."