PCBancard Proposal Generator - Dual Renderer Architecture

Overview

This app supports **two rendering engines** for generating proposals:

1. **Replit Native** - Generates DOCX/PDF directly using docx-js and pdfkit
2. **Gamma API** - Creates polished presentations via Gamma's AI design platform

The agent selects which renderer to use based on the merchant's needs.

When to Use Each Renderer

Use Case	Recommended Renderer
Quick, editable document for internal use	Replit Native (DOCX)
Professional presentation for in-person pitch	Gamma (Presentation)
Email attachment to merchant	Replit Native (PDF)
Visually impressive leave-behind	Gamma (PDF export)
Offline/no API dependency	Replit Native
High-design, branded materials	Gamma

UI: Renderer Selection

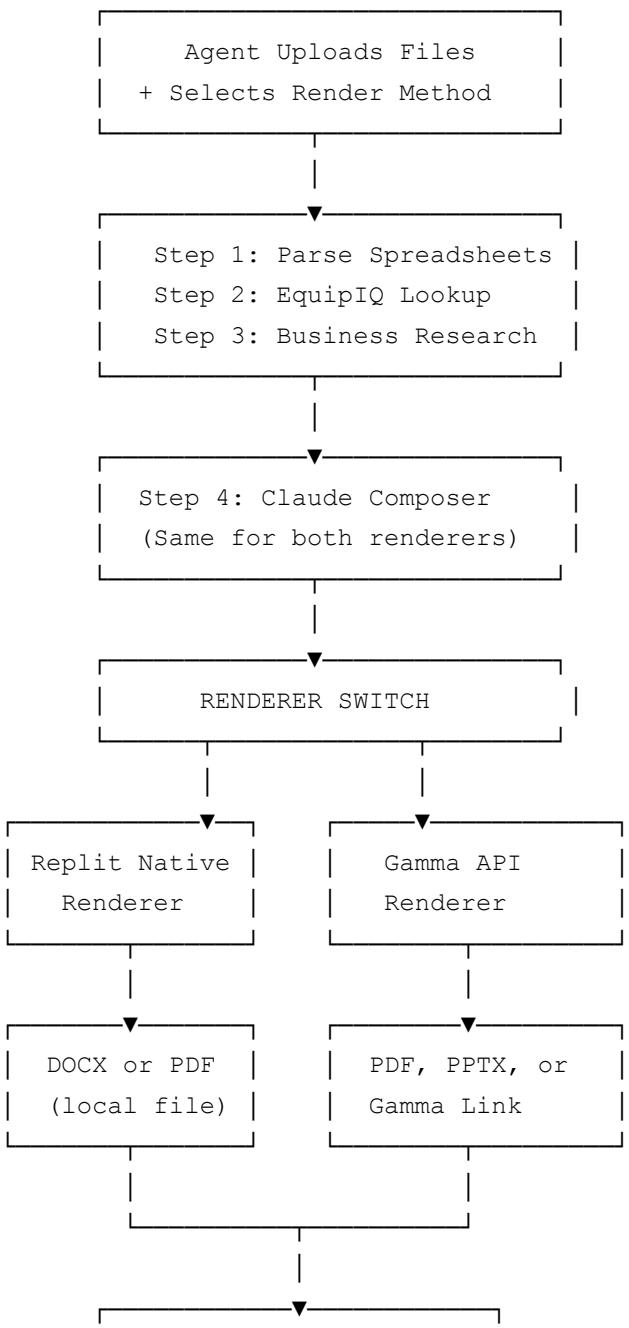
Add this to the agent form:

```
// React component for renderer selection
function RendererSelector({ value, onChange }) {
  return (
    <div className="renderer-selector">
```

```
<h3>Output Method</h3>

<div className="option-cards">
  <div
    className={`option-card ${value === 'replit' ? 'selected' : ''}`}
    onClick={() => onChange('replit')}
  >
    <div className="icon">📄</div>
    <h4>Replit Native</h4>
    <p>Fast DOCX or PDF generation</p>
    <ul>
      <li>Editable Word document</li>
      <li>No external API needed</li>
      <li>Instant generation</li>
    </ul>
  </div>

  <div
    className={`option-card ${value === 'gamma' ? 'selected' : ''}`}
    onClick={() => onChange('gamma')}
  >
    <div className="icon">✨</div>
    <h4>Gamma Presentation</h4>
    <p>AI-designed presentation</p>
    <ul>
      <li>Professional design</li>
      <li>Animated slides</li>
      <li>PDF or PPTX export</li>
    </ul>
  </div>
</div>

{value === 'replit' && (
  <div className="format-selector">
    <label>Format:</label>
    <select>
      <option value="pdf">PDF</option>
      <option value="docx">Word Document (DOCX)</option>
    </select>
  </div>
)}

{value === 'gamma' && (
  <div className="format-selector">
    <label>Export as:</label>
    <select>
      <option value="pdf">PDF</option>
```

```
            <option value="pptx">PowerPoint (PPTX)</option>
            <option value="link">Gamma Link Only</option>
          </select>
        </div>
      )}
    </div>
  );
}
```

Architecture: Dual Renderer Flow



```
                    │   Return to Agent with    │
                    │   Download Link(s)        │
                    └──────────────────────────┘
```

Gamma API Integration

Environment Setup

```
# .env file
GAMMA_API_KEY=sk-gamma-xxxxxxxx
GAMMA_API_BASE=https://public-api.gamma.app/v1.0
```

Gamma Renderer Module

```
// renderers/gamma.js

const GAMMA_API_KEY = process.env.GAMMA_API_KEY;
const GAMMA_BASE_URL = 'https://public-api.gamma.app/v1.0';

class GammaRenderer {

  constructor() {
    if (!GAMMA_API_KEY) {
      throw new Error('GAMMA_API_KEY not configured');
    }
  }

  /**
   * Generate a proposal presentation via Gamma API
   * @param {Object} blueprint - Claude-generated proposal blueprint
   * @param {Object} options - Generation options
   * @returns {Object} - URLs and metadata
   */
  async generateProposal(blueprint, options = {}) {
    const {
      exportAs = 'pdf',        // 'pdf', 'pptx', or null for link only
      themeId = 'Professional', // Gamma theme name
      numCards = 8,            // Number of slides
    } = options;

    // Convert blueprint to Gamma-friendly input text
    const inputText = this.formatForGamma(blueprint);
```

```
  // Step 1: Initiate generation
  const generationId = await this.startGeneration({
    inputText,
    format: 'presentation',
    numCards,
    themeId,
    exportAs,
    cardOptions: {
      dimensions: '16x9'
    },
    contentOptions: {
      textDensity: 'medium',
      tone: ['professional', 'confident']
    }
  });

  // Step 2: Poll for completion
  const result = await this.pollUntilComplete(generationId);

  // Step 3: Download the file if exportAs was specified
  let localFilePath = null;
  if (exportAs && result.exportUrl) {
    localFilePath = await this.downloadFile(
      result.exportUrl,
      `proposal-${Date.now()}.${exportAs}`
    );
  }

  return {
    success: true,
    gammaUrl: result.gammaUrl,
    exportUrl: result.exportUrl,
    localFilePath,
    creditsUsed: result.credits?.deducted,
    creditsRemaining: result.credits?.remaining
  };
}

/**
 * Convert Claude blueprint to Gamma input format
 */
formatForGamma(blueprint) {
  // Gamma works best with structured text using \n---\n as card breaks
  const sections = [];

  // Cover slide
```

```
    sections.push(`
# ${blueprint.cover.headline}
${blueprint.cover.subheadline}

**Prepared for:** ${blueprint.cover.prepared_for}
**Prepared by:** ${blueprint.cover.prepared_by}
**Date:** ${blueprint.cover.date}
    `.trim());

    // Executive Summary
    sections.push(`
# Executive Summary

${blueprint.executive_summary.opening_paragraph}

**Key Findings:**
${blueprint.executive_summary.key_findings.map(f => `• ${f}`).join('\n')}

**Recommendation:** ${blueprint.executive_summary.recommendation}
    `.trim());

    // Current Situation
    sections.push(`
# Your Current Processing Costs

${blueprint.current_situation.narrative}

| Card Brand | Volume | Rate | Monthly Cost |
|------------|--------|------|--------------|
${blueprint.current_situation.table.rows.map(row =>
  `| ${row.join(' | ')} |`
).join('\n')}

**Total Monthly Cost:** ${blueprint.current_situation.total_monthly}
**Effective Rate:** ${blueprint.current_situation.effective_rate}
    `.trim());

    // Dual Pricing Option
    sections.push(`
# ${blueprint.option_dual_pricing.title}

${blueprint.option_dual_pricing.tagline}

**How It Works:**
${blueprint.option_dual_pricing.how_it_works}

**Benefits:**
```

```
${blueprint.option_dual_pricing.benefits.map(b => `• ${b}`).join('\n')}

**Your Cost:**
• Monthly Program Fee: ${blueprint.option_dual_pricing.costs.monthly_program_fee}
• Processing Cost: ${blueprint.option_dual_pricing.costs.processing_cost}
• **Total Monthly: ${blueprint.option_dual_pricing.costs.total_monthly}**

**Savings:**
• Monthly: ${blueprint.option_dual_pricing.savings.monthly}
• Annual: ${blueprint.option_dual_pricing.savings.annual}
    `.trim());

    // Interchange Plus Option
    sections.push(`
# ${blueprint.option_interchange_plus.title}

${blueprint.option_interchange_plus.tagline}

**How It Works:**
${blueprint.option_interchange_plus.how_it_works}

**Benefits:**
${blueprint.option_interchange_plus.benefits.map(b => `• ${b}`).join('\n')}

**Your Cost:**
• Rate: ${blueprint.option_interchange_plus.costs.rate}
• Per Transaction: ${blueprint.option_interchange_plus.costs.per_transaction}
• **Total Monthly: ${blueprint.option_interchange_plus.costs.total_monthly}**

**Savings:**
• Monthly: ${blueprint.option_interchange_plus.savings.monthly}
• Annual: ${blueprint.option_interchange_plus.savings.annual}
    `.trim());

    // Comparison slide
    sections.push(`
# Side-by-Side Comparison

| | Current | Dual Pricing | Interchange Plus |
|---|---|---|---|
${blueprint.comparison_table.rows.map(row =>
  `| ${row.join(' | ')} |`
).join('\n')}
    `.trim());

    // Equipment slide
    if (blueprint.equipment) {
```

```
      sections.push(`
# ${blueprint.equipment.title}

**${blueprint.equipment.terminal_name}**

${blueprint.equipment.why_recommended}

**Features:**
${blueprint.equipment.features.map(f => `• ${f}`).join('\n')}
      `.trim());
    }

    // Next Steps
    sections.push(`
# Getting Started

${blueprint.next_steps.steps.map((step, i) => `${i + 1}. ${step}`).join('\n')}

**${blueprint.next_steps.cta_primary}**

${blueprint.next_steps.cta_secondary}
    `.trim());

    // Disclosures (smaller slide)
    sections.push(`
# Important Disclosures

${blueprint.assumptions_disclosures.map(d => `• ${d}`).join('\n')}
    `.trim());

    // Join with Gamma's card break syntax
    return sections.join('\n\n---\n\n');
  }

  /**
   * Start a generation request
   */
  async startGeneration(params) {
    const response = await fetch(`${GAMMA_BASE_URL}/generations`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'X-API-KEY': GAMMA_API_KEY
      },
      body: JSON.stringify(params)
    });
```

```
  if (!response.ok) {
    const error = await response.json();
    throw new Error(`Gamma API error: ${error.message || response.statusText}`)
  }

  const data = await response.json();
  return data.generationId;
}

/**
 * Poll until generation completes
 */
async pollUntilComplete(generationId, maxAttempts = 60, intervalMs = 2000) {
  for (let i = 0; i < maxAttempts; i++) {
    const response = await fetch(`${GAMMA_BASE_URL}/generations/${generationId}
      headers: {
        'X-API-KEY': GAMMA_API_KEY
      }
    });

    const data = await response.json();

    if (data.status === 'completed') {
      return {
        gammaUrl: data.gammaUrl,
        exportUrl: data.pdfUrl || data.pptxUrl,
        credits: data.credits
      };
    }

    if (data.status === 'failed') {
      throw new Error(`Gamma generation failed: ${data.error || 'Unknown error'
    }

    // Still pending, wait and retry
    await new Promise(resolve => setTimeout(resolve, intervalMs));
  }

  throw new Error('Gamma generation timed out');
}

/**
 * Download file from Gamma's temporary URL
 */
async downloadFile(url, filename) {
  const response = await fetch(url);
  const buffer = await response.arrayBuffer();
```

```
    const fs = require('fs');
    const path = require('path');

    const outputDir = path.join(process.cwd(), 'outputs', 'proposals');
    if (!fs.existsSync(outputDir)) {
      fs.mkdirSync(outputDir, { recursive: true });
    }

    const filePath = path.join(outputDir, filename);
    fs.writeFileSync(filePath, Buffer.from(buffer));

    return filePath;
  }

  /**
   * Check API health and credits
   */
  async checkStatus() {
    try {
      // Make a minimal request to verify API key works
      const response = await fetch(`${GAMMA_BASE_URL}/themes`, {
        headers: { 'X-API-KEY': GAMMA_API_KEY }
      });

      return {
        available: response.ok,
        error: response.ok ? null : await response.text()
      };
    } catch (error) {
      return {
        available: false,
        error: error.message
      };
    }
  }
}

module.exports = { GammaRenderer };
```

Replit Native Renderer (Unchanged)

```
// renderers/native.js
```

```
const { Document, Packer, Paragraph, TextRun, Table, TableRow, TableCell,
        ImageRun, Header, Footer, AlignmentType, WidthType, ShadingType,
        BorderStyle, HeadingLevel, PageBreak } = require('docx');
const fs = require('fs');
const path = require('path');

class NativeRenderer {

  constructor() {
    this.logoPath = path.join(process.cwd(), 'assets', 'pcbancard-logo.png');
  }

  async generateProposal(blueprint, options = {}) {
    const { format = 'pdf' } = options;

    if (format === 'docx') {
      return await this.generateDocx(blueprint);
    } else {
      // Generate DOCX first, then convert to PDF
      const docxPath = await this.generateDocx(blueprint);
      return await this.convertToPdf(docxPath);
    }
  }

  async generateDocx(blueprint) {
    const logo = fs.existsSync(this.logoPath)
      ? fs.readFileSync(this.logoPath)
      : null;

    const doc = new Document({
      styles: {
        default: {
          document: { run: { font: "Arial", size: 24 } }
        },
        paragraphStyles: [
          {
            id: "Heading1",
            name: "Heading 1",
            basedOn: "Normal",
            next: "Normal",
            quickFormat: true,
            run: { size: 36, bold: true, font: "Arial", color: "1E3A5F" },
            paragraph: { spacing: { before: 240, after: 120 } }
          },
          {
            id: "Heading2",
            name: "Heading 2",
```

```
        basedOn: "Normal",
        next: "Normal",
        quickFormat: true,
        run: { size: 28, bold: true, font: "Arial", color: "1E3A5F" },
        paragraph: { spacing: { before: 200, after: 100 } }
      }
    ]
  },
  sections: [{
    properties: {
      page: {
        size: { width: 12240, height: 15840 },
        margin: { top: 1440, right: 1440, bottom: 1440, left: 1440 }
      }
    },
    headers: logo ? {
      default: new Header({
        children: [
          new Paragraph({
            alignment: AlignmentType.RIGHT,
            children: [
              new ImageRun({
                type: "png",
                data: logo,
                transformation: { width: 150, height: 50 },
                altText: { title: "PCBancard", description: "Logo", name: "lo
              })
            ]
          })
        ]
      })
    } : {},
    children: this.buildSections(blueprint)
  }]
});

const buffer = await Packer.toBuffer(doc);

const outputDir = path.join(process.cwd(), 'outputs', 'proposals');
if (!fs.existsSync(outputDir)) {
  fs.mkdirSync(outputDir, { recursive: true });
}

const filename = `proposal-${Date.now()}.docx`;
const filePath = path.join(outputDir, filename);
fs.writeFileSync(filePath, buffer);
```

```
    return {
      success: true,
      localFilePath: filePath,
      format: 'docx'
    };
  }

  buildSections(blueprint) {
    // ... (same implementation as previous architecture doc)
    // See pcbancard-proposal-generator-architecture.md for full code
    return [];
  }

  async convertToPdf(docxPath) {
    // Use LibreOffice for conversion
    const { execSync } = require('child_process');
    const outputDir = path.dirname(docxPath);

    try {
      execSync(`soffice --headless --convert-to pdf --outdir "${outputDir}" "${do

      const pdfPath = docxPath.replace('.docx', '.pdf');

      return {
        success: true,
        localFilePath: pdfPath,
        format: 'pdf'
      };
    } catch (error) {
      // Fallback: return DOCX if PDF conversion fails
      console.error('PDF conversion failed:', error);
      return {
        success: true,
        localFilePath: docxPath,
        format: 'docx',
        warning: 'PDF conversion failed, returning DOCX instead'
      };
    }
  }

  async checkStatus() {
    return {
      available: true,
      error: null
    };
  }
}
```

```
module.exports = { NativeRenderer };
```

Unified Renderer Interface

```
// renderers/index.js

const { GammaRenderer } = require('./gamma');
const { NativeRenderer } = require('./native');

class ProposalRenderer {

  constructor() {
    this.gamma = new GammaRenderer();
    this.native = new NativeRenderer();
  }

  /**
   * Generate proposal using specified renderer
   * @param {string} renderer - 'gamma' or 'replit'
   * @param {Object} blueprint - Claude-generated proposal blueprint
   * @param {Object} options - Renderer-specific options
   */
  async generate(renderer, blueprint, options = {}) {
    // Check renderer availability first
    const status = await this.checkRenderer(renderer);

    if (!status.available) {
      // Fallback to other renderer
      console.warn(`${renderer} unavailable: ${status.error}. Falling back.`);
      renderer = renderer === 'gamma' ? 'replit' : 'gamma';

      const fallbackStatus = await this.checkRenderer(renderer);
      if (!fallbackStatus.available) {
        throw new Error('No renderers available');
      }
    }

    if (renderer === 'gamma') {
      return await this.gamma.generateProposal(blueprint, options);
    } else {
      return await this.native.generateProposal(blueprint, options);
    }
  }
```

```
  /**
   * Check if a renderer is available
   */
  async checkRenderer(renderer) {
    if (renderer === 'gamma') {
      return await this.gamma.checkStatus();
    } else {
      return await this.native.checkStatus();
    }
  }

  /**
   * Get status of all renderers
   */
  async getAllStatus() {
    const [gammaStatus, nativeStatus] = await Promise.all([
      this.gamma.checkStatus(),
      this.native.checkStatus()
    ]);

    return {
      gamma: gammaStatus,
      native: nativeStatus
    };
  }
}

module.exports = { ProposalRenderer };
```

Updated API Endpoint

```
// routes/proposals.js

const express = require('express');
const router = express.Router();
const { ProposalRenderer } = require('../renderers');
const { parseSpreadsheets } = require('../parsers');
const { composeProposal } = require('../composer');

const renderer = new ProposalRenderer();

// Check renderer availability
router.get('/renderers/status', async (req, res) => {
```

```
  const status = await renderer.getAllStatus();
  res.json(status);
});

// Generate proposal
router.post('/generate', async (req, res) => {
  try {
    const {
      dpFile,
      icFile,
      merchantInfo,
      agentInfo,
      rendererType,  // 'gamma' or 'replit'
      outputFormat,  // 'pdf', 'docx', 'pptx', or 'link'
      equipmentId
    } = req.body;

    // Step 1-3: Parse, equipment, research (same as before)
    const pricingData = await parseSpreadsheets(dpFile, icFile);
    const equipment = await selectEquipment(pricingData, equipmentId);
    const research = merchantInfo.website_url
      ? await researchBusiness(merchantInfo)
      : null;

    // Step 4: Claude composition
    const blueprint = await composeProposal({
      pricing: pricingData,
      equipment,
      research,
      merchant: merchantInfo,
      agent: agentInfo
    });

    // Step 5: Render using selected renderer
    const result = await renderer.generate(rendererType, blueprint, {
      exportAs: outputFormat,
      format: outputFormat
    });

    // Return result to agent
    res.json({
      success: true,
      renderer: rendererType,
      ...result,
      summary: {
        merchant: merchantInfo.business_name,
        dual_pricing_annual_savings: pricingData.option_dual_pricing.annual_savin
```

```
        interchange_plus_annual_savings: pricingData.option_interchange_plus.annu
        equipment: equipment?.terminal_name
      }
    });

  } catch (error) {
    console.error('Proposal generation error:', error);
    res.status(500).json({
      success: false,
      error: error.message
    });
  }
});

module.exports = router;
```

Agent UI: Results Display

```
// components/ProposalResult.jsx

function ProposalResult({ result }) {
  if (!result.success) {
    return (
      <div className="result error">
        <h3>❌ Generation Failed</h3>
        <p>{result.error}</p>
        <button onClick={() => window.location.reload()}>Try Again</button>
      </div>
    );
  }

  return (
    <div className="result success">
      <h3>✅ Proposal Generated!</h3>

      <div className="summary-card">
        <h4>{result.summary.merchant}</h4>
        <div className="savings-grid">
          <div className="savings-item dual-pricing">
            <span className="label">Dual Pricing Savings</span>
            <span className="monthly">${result.summary.dual_pricing_annual_saving
          </div>
          <div className="savings-item ic-plus">
            <span className="label">IC+ Savings</span>
```

```
            <span className="monthly">${result.summary.interchange_plus_annual_sa
          </div>
        </div>
        {result.summary.equipment && (
          <p className="equipment">Equipment: {result.summary.equipment}</p>
        )}
      </div>

      <div className="download-actions">
        {result.localFilePath && (
          <a
            href={`/api/download?path=${encodeURIComponent(result.localFilePath)}
            className="btn primary"
            download
          >
            📥 Download {result.format?.toUpperCase() || 'File'}
          </a>
        )}

        {result.gammaUrl && (
          <a
            href={result.gammaUrl}
            target="_blank"
            rel="noopener noreferrer"
            className="btn secondary"
          >
            🔗 Open in Gamma
          </a>
        )}
      </div>

      {result.creditsRemaining !== undefined && (
        <p className="credits-info">
          Gamma credits remaining: {result.creditsRemaining}
        </p>
      )}

      <div className="next-actions">
        <button onClick={() => emailToMerchant(result)}>
          📧 Email to Merchant
        </button>
        <button onClick={() => startNewProposal()}>
          ➕ Create Another
        </button>
      </div>
    </div>
  );
```

```
}
```

Environment Variables Required

```
# .env

# Gamma API (required for Gamma renderer)
GAMMA_API_KEY=sk-gamma-xxxxxxxx

# Claude API (required for proposal composition)
ANTHROPIC_API_KEY=sk-ant-xxxxxxxx

# Optional: Custom Gamma theme ID
GAMMA_THEME_ID=Professional

# Optional: Gemini for image generation fallback
GEMINI_API_KEY=xxxxxxxx
```

Fallback Logic

The system automatically falls back if a renderer is unavailable:

```
Agent selects Gamma → Gamma API down → Falls back to Replit Native
Agent selects Replit → LibreOffice missing → Returns DOCX without PDF conversion
Both unavailable → Error with clear message
```

Testing Both Renderers

Add this to your QA prompt:

```
### Renderer Testing

GAMMA RENDERER:
□ Gamma API key is valid
□ Gamma generation starts successfully
□ Gamma polling completes (doesn't timeout)
□ PDF download URL works
□ PPTX download URL works
```

```
□ Downloaded file is not corrupted
□ Gamma link opens correctly
□ Credits deduction is shown
□ Handles Gamma API errors gracefully
□ Falls back to Replit when Gamma is down

REPLIT NATIVE RENDERER:
□ DOCX generates successfully
□ PDF conversion works (LibreOffice)
□ Falls back to DOCX if PDF fails
□ PCBancard logo appears
□ All data renders correctly
□ Tables format properly
□ File downloads work

RENDERER SWITCHING:
□ UI toggle works
□ Selection persists during generation
□ Status indicator shows renderer availability
□ Fallback happens automatically when needed
□ User is notified of fallback
```

Summary

Feature	Replit Native	Gamma API
Output Formats	PDF, DOCX	PDF, PPTX, Link
Design Quality	Good (template-based)	Excellent (AI-designed)
Generation Speed	Fast (~5 sec)	Slower (~30 sec)
Offline Capable	Yes	No
Cost	Free	Uses Gamma credits
Editability	Full (DOCX)	Via Gamma editor
Dependencies	docx-js, LibreOffice	Gamma API key

Both renderers use the **same Claude-generated blueprint**, so the content is identical—only the visual presentation differs.