

AI-Powered Flyer Generation System - Replit Implementation Guide

CRITICAL: READ THIS FIRST

Your current system is failing because it's trying to use an image generation model to create an entire flyer as a single image. This will NEVER produce reliable, text-accurate results.

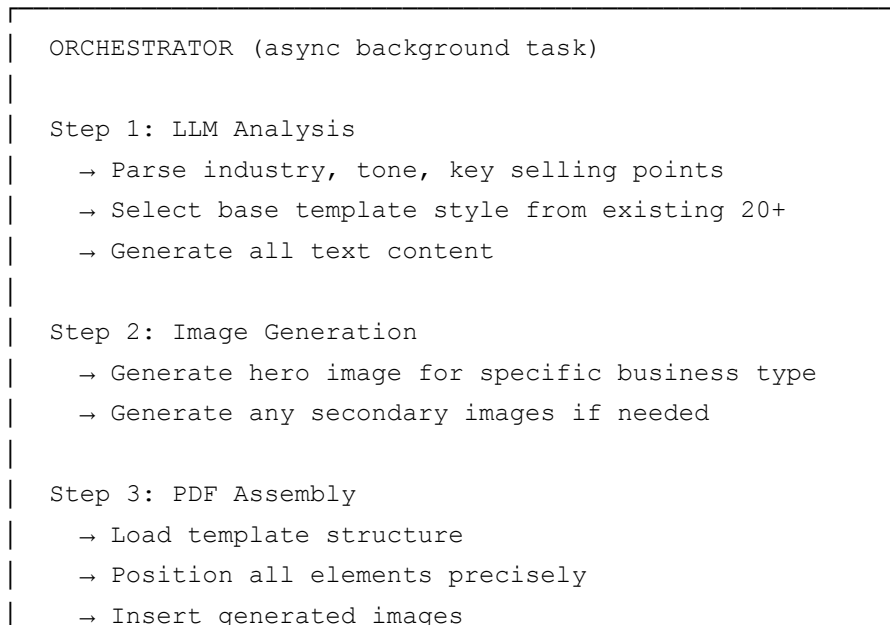
The Solution: A multi-agent pipeline that:

1. **AI Agent 1 (LLM):** Analyzes request → selects template style → generates copy
 2. **AI Agent 2 (Image Gen):** Creates ONLY the hero image for the specific business type
 3. **Assembly Engine:** Programmatically builds the PDF using ReportLab with precise positioning
 4. **Background Worker:** Handles async processing and notifications
-

ARCHITECTURE OVERVIEW

User Request

↓



```
|   → Insert generated text
|   → Apply branding (logos, colors)
|
| Step 4: Finalize & Notify
|   → Save PDF to storage
|   → Update database status
|   → Trigger notification
|
```

↓

Final PDF Ready for Download

COMPLETE PYTHON IMPLEMENTATION

File 1: `flyer_generator/orchestrator.py`

```
"""
Main orchestrator for the multi-agent flyer generation pipeline.
This runs as a background task and coordinates all AI agents.
"""

import asyncio
import os
import json
import logging
from datetime import datetime
from typing import Optional, Dict, Any
import uuid

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

class FlyerGenerationOrchestrator:
    """
    Coordinates the multi-step flyer generation process.
    Each step is handled by a specialized agent.
    """

    def __init__(self, db_connection, storage_client):
        self.db = db_connection
        self.storage = storage_client
        self.llm_agent = LLMCopyAgent()
        self.image_agent = ImageGenerationAgent()
```

```

self.pdf_assembler = PDFAssembler()

async def generate_flyer(
    self,
    request_id: str,
    industry: str,
    custom_prompt: str,
    contact_info: Dict[str, str],
    callback_url: Optional[str] = None
) -> str:
    """
    Main entry point for flyer generation.
    Returns the request_id for tracking.
    """
    try:
        # Update status: Starting
        await self._update_status(request_id, "processing", "Starting generat

        # =====
        # STEP 1: LLM Analysis and Copy Generation
        # =====
        logger.info(f"[{request_id}] Step 1: LLM Analysis")
        await self._update_status(request_id, "processing", "Analyzing reques

        llm_result = await self.llm_agent.analyze_and_generate(
            industry=industry,
            custom_prompt=custom_prompt,
            contact_info=contact_info
        )

        if not llm_result.get("success"):
            raise Exception(f"LLM generation failed: {llm_result.get('error')}

        # =====
        # STEP 2: Image Generation
        # =====
        logger.info(f"[{request_id}] Step 2: Image Generation")
        await self._update_status(request_id, "processing", "Generating custo

        image_result = await self.image_agent.generate_hero_image(
            industry=industry,
            style_hints=llm_result.get("style_hints", {}),
            business_description=llm_result.get("business_description", "")
        )

        if not image_result.get("success"):
            # Fall back to stock image if generation fails

```

```

        logger.warning(f"[{request_id}] Image generation failed, using fa
        image_result = await self.image_agent.get_fallback_image(industry

# =====
# STEP 3: PDF Assembly
# =====
logger.info(f"[{request_id}] Step 3: PDF Assembly")
await self._update_status(request_id, "processing", "Assembling final

pdf_result = await self.pdf_assembler.create_flyer(
    template_style=llm_result.get("template_style", "dual_pricing"),
    hero_image_path=image_result.get("image_path"),
    copy_content=llm_result.get("copy_content"),
    contact_info=contact_info,
    branding=self._get_branding_config()
)

if not pdf_result.get("success"):
    raise Exception(f"PDF assembly failed: {pdf_result.get('error')}")

# =====
# STEP 4: Finalize and Notify
# =====
logger.info(f"[{request_id}] Step 4: Finalizing")

# Upload to storage
pdf_url = await self.storage.upload_file(
    file_path=pdf_result.get("pdf_path"),
    destination=f"flyers/{request_id}.pdf"
)

# Update database with completion
await self._update_status(
    request_id,
    "completed",
    "Flyer generated successfully!",
    pdf_url=pdf_url
)

# Send notification if callback provided
if callback_url:
    await self._send_notification(callback_url, request_id, pdf_url)

logger.info(f"[{request_id}] Generation complete: {pdf_url}")
return pdf_url

except Exception as e:

```

```

        logger.error(f"[{request_id}] Generation failed: {str(e)}")
        await self._update_status(request_id, "failed", str(e))
        raise

    async def _update_status(
        self,
        request_id: str,
        status: str,
        message: str,
        pdf_url: Optional[str] = None
    ):
        """Update the generation status in the database."""
        update_data = {
            "status": status,
            "message": message,
            "updated_at": datetime.utcnow().isoformat()
        }
        if pdf_url:
            update_data["pdf_url"] = pdf_url

        # Update your database here
        # await self.db.flyer_requests.update_one(
        #     {"request_id": request_id},
        #     {"$set": update_data}
        # )
        pass

    async def _send_notification(self, callback_url: str, request_id: str, pdf_url: str):
        """Send webhook notification when complete."""
        import aiohttp
        async with aiohttp.ClientSession() as session:
            await session.post(callback_url, json={
                "request_id": request_id,
                "status": "completed",
                "pdf_url": pdf_url
            })

    def _get_branding_config(self) -> Dict[str, Any]:
        """Return PCBancard branding configuration."""
        return {
            "primary_color": "#6366F1", # Purple from your screenshots
            "secondary_color": "#F59E0B", # Yellow/gold
            "logo_path": "assets/pcbancard_logo.png",
            "company_name": "PCBancard",
            "tagline": "PAY $0 TO PROCESS",
            "fonts": {
                "heading": "Helvetica-Bold",
            }
        }

```

```

        "body": "Helvetica",
        "accent": "Helvetica-BoldOblique"
    }
}

```

File 2: flyer_generator/llm_agent.py

```

"""
LLM Agent for analyzing requests and generating flyer copy.
Uses Gemini or Claude API for intelligent content generation.
"""

import os
import json
from typing import Dict, Any, Optional
import google.generativeai as genai

class LLMCopyAgent:
    """
    Handles all LLM-based tasks:
    - Analyzing the user's request
    - Selecting appropriate template style
    - Generating all text content for the flyer
    """

    def __init__(self):
        # Initialize Gemini
        genai.configure(api_key=os.environ.get("GEMINI_API_KEY"))
        self.model = genai.GenerativeModel('gemini-1.5-flash')

        # Load your existing flyer templates as reference
        self.template_catalog = self._load_template_catalog()

    def _load_template_catalog(self) -> Dict[str, Any]:
        """
        Load information about your existing 20+ flyer templates.
        This helps the LLM understand what styles are available.
        """
        return {
            "dual_pricing": {
                "description": "Standard dual pricing flyer with fee comparison t
                "industries": ["retail", "restaurants", "general"],
                "layout": "two_column",
                "key_sections": ["benefits_list", "fee_comparison", "qr_code", "c
            },
            "hotsauce_pos": {

```

```

        "description": "Restaurant/bar focused with POS system emphasis",
        "industries": ["restaurants", "bars", "food_trucks"],
        "layout": "feature_focused",
        "key_sections": ["pos_features", "savings_highlight", "contact"]
    },
    "cash_advance": {
        "description": "Merchant cash advance focused",
        "industries": ["any"],
        "layout": "benefit_driven",
        "key_sections": ["funding_steps", "key_features", "contact"]
    },
    "b2b_level23": {
        "description": "B2B/Level 2&3 processing for commercial clients",
        "industries": ["b2b", "wholesale", "manufacturing"],
        "layout": "professional",
        "key_sections": ["level23_explanation", "savings_example", "conta
    },
    "industry_specific": {
        "description": "Industry-customized with relevant imagery",
        "industries": ["veterinarians", "automotive", "garden_centers",
            "liquor_stores", "attorneys", "tattoo_parlors",
            "salons", "convenience_stores", "rock_gravel",
            "pizzerias", "dentists", "gun_shops"],
        "layout": "two_column_with_hero",
        "key_sections": ["industry_benefits", "fee_comparison", "technolo
    }
}

```

```

async def analyze_and_generate(
    self,
    industry: str,
    custom_prompt: str,
    contact_info: Dict[str, str]
) -> Dict[str, Any]:
    """
    Main method: Analyze the request and generate all flyer content.
    """

    # Build the comprehensive prompt
    system_prompt = self._build_system_prompt()
    user_prompt = self._build_user_prompt(industry, custom_prompt, contact_in

    try:
        response = self.model.generate_content([
            {"role": "user", "parts": [system_prompt + "\n\n" + user_prompt]}
        ])
    
```

```

        # Parse the structured response
        result = self._parse_llm_response(response.text)
        result["success"] = True
        return result

    except Exception as e:
        return {
            "success": False,
            "error": str(e)
        }

    def _build_system_prompt(self) -> str:
        return ""You are an expert marketing copywriter specializing in merchant

```

CRITICAL CONTEXT - PCBancard's Value Proposition:

- Dual Pricing allows merchants to PAY \$0 TO PROCESS credit cards
- The processing fee is passed to the customer (clearly displayed)
- Cash customers get a discount
- Month-to-month contracts with NO cancellation fees
- Free equipment provided
- Includes HotSauce POS technology
- Supports giving back to local charities

AVAILABLE TEMPLATE STYLES:

1. dual_pricing - Standard comparison layout showing traditional vs dual pricing
2. hotsauce_pos - Restaurant/bar focused with POS features
3. cash_advance - Merchant funding focused
4. b2b_level23 - B2B commercial processing
5. industry_specific - Customized for specific verticals

YOUR OUTPUT MUST BE VALID JSON with this exact structure:

```

{
  "template_style": "one of the template names above",
  "style_hints": {
    "mood": "professional/friendly/urgent/etc",
    "color_emphasis": "which brand color to emphasize",
    "imagery_style": "description of ideal hero image"
  },
  "business_description": "one sentence describing ideal hero image scene",
  "copy_content": {
    "headline": "main attention-grabbing headline",
    "subheadline": "supporting statement",
    "benefit_bullets": ["bullet 1", "bullet 2", "bullet 3", "bullet 4", "bullet 5"],
    "cta_text": "call to action text",
    "industry_hook": "industry-specific opening line",
    "savings_statement": "specific savings claim",
    "technology_highlight": "tech feature to emphasize"
  }
}

```

```

    }
}"""

def _build_user_prompt(
    self,
    industry: str,
    custom_prompt: str,
    contact_info: Dict[str, str]
) -> str:
    return f"""Generate flyer content for the following:

INDUSTRY: {industry}
CUSTOM INSTRUCTIONS: {custom_prompt if custom_prompt else "Standard dual pricing"}
CONTACT NAME: {contact_info.get('name', 'Your Local Representative')}

Consider what would resonate most with a {industry} business owner. What are thei

Remember: Output ONLY valid JSON, no markdown formatting, no code blocks."""

def _parse_llm_response(self, response_text: str) -> Dict[str, Any]:
    """Parse the LLM's JSON response."""
    # Clean up common issues
    cleaned = response_text.strip()
    if cleaned.startswith("`json`"):
        cleaned = cleaned[7:]
    if cleaned.startswith("`"):
        cleaned = cleaned[3:]
    if cleaned.endswith("`"):
        cleaned = cleaned[:-3]

    return json.loads(cleaned)

```

File 3: flyer_generator/image_agent.py

```

"""
Image Generation Agent for creating custom hero images.
Uses Gemini Imagen or falls back to curated stock images.
"""

import os
import uuid
import aiohttp
import base64
from typing import Dict, Any, Optional
from pathlib import Path

```

```

class ImageGenerationAgent:
    """
    Handles AI image generation for flyer hero images.
    Generates business-appropriate imagery for specific industries.
    """

    def __init__(self):
        self.output_dir = Path("generated_images")
        self.output_dir.mkdir(exist_ok=True)
        self.fallback_images = self._load_fallback_catalog()

    def _load_fallback_catalog(self) -> Dict[str, str]:
        """
        Map of industry to fallback stock image paths.
        Use these when AI generation fails or for consistency.
        """
        return {
            "dentist": "assets/stock/dentist_office.jpg",
            "dental": "assets/stock/dentist_office.jpg",
            "veterinarians": "assets/stock/vet_clinic.jpg",
            "automotive": "assets/stock/auto_shop.jpg",
            "restaurants": "assets/stock/restaurant.jpg",
            "bars": "assets/stock/bar.jpg",
            "liquor_stores": "assets/stock/liquor_store.jpg",
            "gun_shops": "assets/stock/gun_shop.jpg",
            "firearms": "assets/stock/gun_shop.jpg",
            "garden_centers": "assets/stock/garden_center.jpg",
            "retail": "assets/stock/retail_store.jpg",
            "salons": "assets/stock/salon.jpg",
            "tattoo_parlors": "assets/stock/tattoo_shop.jpg",
            "pizzerias": "assets/stock/pizzeria.jpg",
            "food_trucks": "assets/stock/food_truck.jpg",
            "attorneys": "assets/stock/law_office.jpg",
            "convenience_stores": "assets/stock/convenience_store.jpg",
            "rock_gravel": "assets/stock/construction.jpg",
            "general": "assets/stock/business_generic.jpg"
        }

    async def generate_hero_image(
        self,
        industry: str,
        style_hints: Dict[str, Any],
        business_description: str
    ) -> Dict[str, Any]:
        """
        Generate a custom hero image for the flyer.
        """

```

```

# Build the image generation prompt
prompt = self._build_image_prompt(industry, style_hints, business_descrip

try:
    # Try Gemini Imagen first
    image_path = await self._generate_with_gemini(prompt)

    return {
        "success": True,
        "image_path": image_path,
        "source": "ai_generated"
    }

except Exception as e:
    print(f"AI image generation failed: {e}")
    return {
        "success": False,
        "error": str(e)
    }

def _build_image_prompt(
    self,
    industry: str,
    style_hints: Dict[str, Any],
    business_description: str
) -> str:
    """Build an effective prompt for image generation."""

    # Industry-specific scene descriptions
    industry_scenes = {
        "dentist": "A modern, clean dental office with a friendly dentist and
        "dental": "A modern, clean dental office with a friendly dentist and
        "veterinarians": "A caring veterinarian examining a happy dog in a cl
        "automotive": "Professional auto mechanic in clean shop working on ca
        "restaurants": "Busy restaurant scene with happy diners and server, w
        "bars": "Upscale bar with bartender serving customers, ambient lighti
        "liquor_stores": "Well-organized liquor store interior with friendly
        "gun_shops": "Professional firearm retailer display case with knowled
        "firearms": "Professional firearm retailer display case with knowledg
        "garden_centers": "Vibrant garden center with colorful plants and hel
        "retail": "Modern retail store with engaged customers and professiona
        "salons": "Stylish hair salon with stylist and happy client, modern d
        "tattoo_parlors": "Clean professional tattoo studio with artist at wo
        "pizzerias": "Authentic pizzeria kitchen with chef preparing pizza",
        "food_trucks": "Colorful food truck serving happy customers",
        "attorneys": "Professional law office with attorney at desk, bookshel

```

```

        "convenience_stores": "Well-stocked convenience store with friendly c
        "rock_gravel": "Construction/landscaping supply yard with equipment a
    }

    base_scene = industry_scenes.get(
        industry.lower(),
        "Professional business environment with engaged staff and customers"
    )

    mood = style_hints.get("mood", "professional and welcoming")

    prompt = f"""Professional marketing photograph for a business flyer.
Scene: {base_scene}
{business_description}
Style: High-quality commercial photography, {mood} mood
Lighting: Professional studio-style lighting, bright and clear
Composition: Suitable for a marketing flyer hero image, landscape orientation
Quality: 4K, sharp focus, commercial advertising quality
IMPORTANT: No text or logos in the image. Clean background suitable for text over

    return prompt

async def _generate_with_gemini(self, prompt: str) -> str:
    """Generate image using Gemini Imagen API."""
    import google.generativeai as genai

    # Note: Gemini Imagen has specific API requirements
    # This is a placeholder - actual implementation depends on API availabili

    genai.configure(api_key=os.environ.get("GEMINI_API_KEY"))

    # For Imagen 3, you'd use something like:
    # model = genai.ImageGenerationModel("imagen-3.0-generate-001")
    # response = model.generate_images(prompt=prompt, number_of_images=1)

    # Since Imagen availability varies, here's the fallback approach:
    # Use the existing stock image system
    raise NotImplementedError("Falling back to stock images")

async def get_fallback_image(self, industry: str) -> Dict[str, Any]:
    """Get a curated stock image for the industry."""

    industry_key = industry.lower().replace(" ", "_")
    image_path = self.fallback_images.get(
        industry_key,
        self.fallback_images["general"]
    )

```

```

    return {
        "success": True,
        "image_path": image_path,
        "source": "stock_fallback"
    }

```

File 4: flyer_generator/pdf_assembler.py

```

"""
PDF Assembly Engine using ReportLab.
Creates professional flyers by combining AI-generated content with templates.
"""

import os
from pathlib import Path
from typing import Dict, Any, List, Optional
from datetime import datetime
import uuid

from reportlab.lib import colors
from reportlab.lib.pagesizes import letter
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib.units import inch
from reportlab.lib.enums import TA_LEFT, TA_CENTER, TA_RIGHT
from reportlab.platypus import (
    SimpleDocTemplate, Paragraph, Spacer, Image, Table, TableStyle,
    Frame, PageTemplate, BaseDocTemplate, FrameBreak
)
from reportlab.pdfgen import canvas
from reportlab.lib.utils import ImageReader

class PDFAssembler:
    """
    Assembles the final PDF flyer from components.
    Uses ReportLab for precise layout control.
    """

    def __init__(self):
        self.output_dir = Path("generated_flyers")
        self.output_dir.mkdir(exist_ok=True)
        self.styles = self._create_styles()

    def _create_styles(self) -> Dict[str, ParagraphStyle]:
        """Create custom paragraph styles for the flyer."""
        base_styles = getSampleStyleSheet()

```

```
return {
  "main_headline": ParagraphStyle(
    "MainHeadline",
    parent=base_styles["Heading1"],
    fontSize=28,
    textColor=colors.HexColor("#1a1a2e"),
    spaceAfter=6,
    fontName="Helvetica-Bold",
    leading=32
  ),
  "subheadline": ParagraphStyle(
    "Subheadline",
    parent=base_styles["Heading2"],
    fontSize=14,
    textColor=colors.HexColor("#6366F1"),
    spaceAfter=12,
    fontName="Helvetica-Bold"
  ),
  "section_header": ParagraphStyle(
    "SectionHeader",
    parent=base_styles["Heading3"],
    fontSize=12,
    textColor=colors.HexColor("#F59E0B"),
    spaceBefore=10,
    spaceAfter=6,
    fontName="Helvetica-Bold"
  ),
  "body_text": ParagraphStyle(
    "BodyText",
    parent=base_styles["Normal"],
    fontSize=9,
    textColor=colors.HexColor("#333333"),
    spaceAfter=4,
    fontName="Helvetica",
    leading=12
  ),
  "bullet_point": ParagraphStyle(
    "BulletPoint",
    parent=base_styles["Normal"],
    fontSize=9,
    textColor=colors.HexColor("#333333"),
    leftIndent=15,
    spaceAfter=3,
    fontName="Helvetica",
    bulletIndent=5
  ),
}
```

```

        "cta_text": ParagraphStyle(
            "CTAText",
            parent=base_styles["Normal"],
            fontSize=11,
            textColor=colors.white,
            alignment=TA_CENTER,
            fontName="Helvetica-Bold"
        ),
        "contact_info": ParagraphStyle(
            "ContactInfo",
            parent=base_styles["Normal"],
            fontSize=10,
            textColor=colors.HexColor("#333333"),
            fontName="Helvetica"
        ),
        "footer_text": ParagraphStyle(
            "FooterText",
            parent=base_styles["Normal"],
            fontSize=8,
            textColor=colors.HexColor("#666666"),
            alignment=TA_CENTER,
            fontName="Helvetica"
        )
    }

async def create_flyer(
    self,
    template_style: str,
    hero_image_path: str,
    copy_content: Dict[str, Any],
    contact_info: Dict[str, str],
    branding: Dict[str, Any]
) -> Dict[str, Any]:
    """
    Create the final PDF flyer.
    """
    try:
        # Generate unique filename
        output_filename = f"flyer_{uuid.uuid4().hex[:8]}.pdf"
        output_path = self.output_dir / output_filename

        # Create the PDF based on template style
        if template_style == "dual_pricing":
            await self._create_dual_pricing_flyer(
                output_path, hero_image_path, copy_content, contact_info, bra
            )
        elif template_style == "industry_specific":

```

```

        await self._create_industry_flyer(
            output_path, hero_image_path, copy_content, contact_info, bra
        )
    else:
        # Default to dual pricing layout
        await self._create_dual_pricing_flyer(
            output_path, hero_image_path, copy_content, contact_info, bra
        )

    return {
        "success": True,
        "pdf_path": str(output_path)
    }

except Exception as e:
    return {
        "success": False,
        "error": str(e)
    }

async def _create_dual_pricing_flyer(
    self,
    output_path: Path,
    hero_image_path: str,
    copy_content: Dict[str, Any],
    contact_info: Dict[str, str],
    branding: Dict[str, Any]
):
    """Create the standard dual pricing comparison flyer."""

    c = canvas.Canvas(str(output_path), pagesize=letter)
    width, height = letter

    # ===== HEADER SECTION =====
    # Purple header bar
    c.setFillColor(colors.HexColor("#6366F1"))
    c.rect(0, height - 100, width, 100, fill=True, stroke=False)

    # "PAY $0 TO PROCESS" headline
    c.setFillColor(colors.white)
    c.setFont("Helvetica-Bold", 32)
    c.drawString(30, height - 50, "PAY $0 TO")
    c.drawString(30, height - 85, "PROCESS")

    # Subheadline
    c.setFont("Helvetica", 11)
    c.drawString(200, height - 50, "Stop losing money every time")

```

```

c.drawString(200, height - 65, "your customer swipes their card.")

# Logo placeholder (right side of header)
c.setFont("Helvetica-Bold", 14)
c.drawRightString(width - 30, height - 45, "PCBancard")

# Industry tag (rotated on right edge)
c.saveState()
c.setFillColor(colors.HexColor("#F59E0B"))
c.rect(width - 25, height - 400, 25, 300, fill=True, stroke=False)
c.setFillColor(colors.white)
c.translate(width - 8, height - 250)
c.rotate(90)
c.setFont("Helvetica-Bold", 10)
industry_text = copy_content.get("industry_hook", "Excellent for Your Bus
c.drawCentredString(0, 0, industry_text)
c.restoreState()

# ===== HERO IMAGE SECTION =====
try:
    if hero_image_path and os.path.exists(hero_image_path):
        img = ImageReader(hero_image_path)
        c.drawImage(img, 30, height - 280, width=250, height=170, preserv
except Exception as e:
    # Draw placeholder if image fails
    c.setFillColor(colors.HexColor("#f0f0f0"))
    c.rect(30, height - 280, 250, 170, fill=True, stroke=False)
    c.setFillColor(colors.HexColor("#999999"))
    c.setFont("Helvetica", 12)
    c.drawCentredString(155, height - 200, "[Business Image]")

# ===== LEFT COLUMN: BENEFITS =====
y_pos = height - 310

c.setFillColor(colors.HexColor("#6366F1"))
c.setFont("Helvetica-Bold", 12)
c.drawString(30, y_pos, "BENEFITS OF THE DUAL")
c.drawString(30, y_pos - 15, "PRICING PROGRAM")

y_pos -= 40
c.setFillColor(colors.HexColor("#333333"))
c.setFont("Helvetica", 8)

benefits = copy_content.get("benefit_bullets", [
    "PCBancard's Dual Pricing Program allows merchants to offer a 'card p
    "Both prices are clearly displayed on the terminal or the ticket",
    "Eliminates the processing costs associated with credit card acceptan

```

```

        "No need to increase the costs of goods and services",
        "Month-to-month contracts with NO cancellation fees",
        "Ability to give back to a local charity through your business proces
    ])

for i, benefit in enumerate(benefits[:6]):
    # Yellow bullet point
    c.setFillColor(colors.HexColor("#F59E0B"))
    c.circle(35, y_pos - 3, 3, fill=True, stroke=False)

    c.setFillColor(colors.HexColor("#333333"))
    # Wrap text if needed
    text = benefit[:80] + "..." if len(benefit) > 80 else benefit
    c.drawString(45, y_pos, text)
    y_pos -= 18

# ===== RIGHT COLUMN: VALUE PROP =====
right_x = 300
y_pos = height - 130

c.setFillColor(colors.HexColor("#6366F1"))
c.setFont("Helvetica-Bold", 11)
c.drawString(right_x, y_pos, "PAY $0 TO PROCESS CREDIT CARDS")

y_pos -= 20
c.setFillColor(colors.HexColor("#333333"))
c.setFont("Helvetica", 8)

value_text = [
    "With processing fees going up like never before,",
    "business owners like you are forced to pay",
    "outrageous fees to accept credit cards.",
    "",
    "With PCBancard's Dual Pricing Program,",
    "you'll pay $0 to process credit cards—and put",
    "that money back into your bottom line, instead."
]

for line in value_text:
    c.drawString(right_x, y_pos, line)
    y_pos -= 12

# Technology highlight
y_pos -= 10
c.setFillColor(colors.HexColor("#F59E0B"))
c.setFont("Helvetica-Bold", 10)
c.drawString(right_x, y_pos, "NEW TECHNOLOGY AVAILABLE!")

```

```

y_pos -= 15
c.setFill_color(colors.HexColor("#333333"))
c.setFont("Helvetica", 8)
tech_text = copy_content.get("technology_highlight",
    "PCBancard offers the latest equipment pre-programmed with the Dual P

# Word wrap the technology text
words = tech_text.split()
lines = []
current_line = []
for word in words:
    current_line.append(word)
    if len(" ".join(current_line)) > 45:
        lines.append(" ".join(current_line[:-1]))
        current_line = [word]
if current_line:
    lines.append(" ".join(current_line))

for line in lines[:4]:
    c.drawString(right_x, y_pos, line)
    y_pos -= 11

# ===== FEE COMPARISON TABLE =====
table_y = 200

# Traditional Processing
c.setFill_color(colors.HexColor("#1a1a2e"))
c.setFont("Helvetica-Bold", 9)
c.drawString(30, table_y, "Traditional Processing Statement")

table_y -= 15
c.setFont("Helvetica", 7)
fees = [
    ("Total Fees", ""),
    ("Interchange Fees", "$1,317.77"),
    ("Card Processing Fees", "$199.93"),
    ("Card Brand Assessment", "$103.76"),
    ("Other Fees", "$137.43")
]

for label, amount in fees:
    c.drawString(35, table_y, label)
    c.drawRightString(180, table_y, amount)
    table_y -= 10

c.setFont("Helvetica-Bold", 8)

```

```

c.drawString(35, table_y, "Total Fees charged to Merchant: $1759.48")

# Dual Pricing Processing
table_y -= 25
c.setFillColor(colors.HexColor("#6366F1"))
c.setFont("Helvetica-Bold", 9)
c.drawString(30, table_y, "Dual Pricing Processing Statement")

table_y -= 15
c.setFillColor(colors.HexColor("#333333"))
c.setFont("Helvetica", 7)
dp_fees = [
    ("Total Program Fees", "$64.95"),
    ("Interchange Fees", "$0.00"),
    ("Card Processing Fees", "$0.00"),
    ("Card Brand Assessment", "$0.00"),
    ("Other Fees", "$0.00")
]

for label, amount in dp_fees:
    c.drawString(35, table_y, label)
    c.drawRightString(180, table_y, amount)
    table_y -= 10

c.setFillColor(colors.HexColor("#6366F1"))
c.setFont("Helvetica-Bold", 8)
c.drawString(35, table_y, "Total Fees Charged to Merchant: $64.95")

# ===== QR CODE & CONTACT SECTION =====
# Right side - QR code placeholder and contact
c.setFillColor(colors.HexColor("#333333"))
c.setFont("Helvetica-Bold", 9)
c.drawString(right_x, 200, "Ready to fight back against processing fees?"

c.setFont("Helvetica", 8)
c.drawString(right_x, 185, "Scan the code and see")
c.drawString(right_x, 173, "how one business owner")
c.drawString(right_x, 161, "put $2,000 back into")
c.drawString(right_x, 149, "his business every")
c.drawString(right_x, 137, "month.")

# QR code placeholder
c.setFillColor(colors.HexColor("#f0f0f0"))
c.rect(right_x + 150, 130, 70, 70, fill=True, stroke=True)
c.setFillColor(colors.HexColor("#666666"))
c.setFont("Helvetica", 8)
c.drawCentredString(right_x + 185, 160, "[QR Code]")

```

```

# Contact info box
c.setFillColor(colors.HexColor("#f8f8f8"))
c.rect(right_x + 130, 40, 120, 80, fill=True, stroke=False)

c.setFillColor(colors.HexColor("#333333"))
c.setFont("Helvetica-Bold", 9)
c.drawString(right_x + 140, 105, "Name:")
c.drawString(right_x + 140, 85, "Phone:")
c.drawString(right_x + 140, 65, "Email:")

c.setFont("Helvetica", 9)
c.drawString(right_x + 180, 105, contact_info.get("name", "")[:20])
c.drawString(right_x + 180, 85, contact_info.get("phone", "")[:15])
c.drawString(right_x + 140, 50, contact_info.get("email", "")[:30])

# ===== FOOTER =====
# Purple footer bar
c.setFillColor(colors.HexColor("#6366F1"))
c.rect(0, 0, width, 35, fill=True, stroke=False)

# Footer content
c.setFillColor(colors.white)
c.setFont("Helvetica-Bold", 10)
c.drawString(30, 15, "PCBancard")

# Payment logos placeholder
c.setFont("Helvetica", 8)
c.drawRightString(width - 30, 15, "VISA | MC | AMEX | DISCOVER")

# Equipment images placeholder
c.setFillColor(colors.HexColor("#e0e0e0"))
c.rect(200, 50, 80, 60, fill=True, stroke=False)
c.setFillColor(colors.HexColor("#666666"))
c.setFont("Helvetica", 7)
c.drawCentredString(240, 75, "[Equipment]")

c.save()

async def _create_industry_flyer(
    self,
    output_path: Path,
    hero_image_path: str,
    copy_content: Dict[str, Any],
    contact_info: Dict[str, str],
    branding: Dict[str, Any]
):

```

```

        """Create an industry-specific customized flyer."""
        # This follows the same pattern as dual_pricing but with
        # industry-specific customizations
        # For now, delegate to the dual pricing template
        await self._create_dual_pricing_flyer(
            output_path, hero_image_path, copy_content, contact_info, branding
        )

```

File 5: `flyer_generator/background_worker.py`

```

"""
Background worker for async flyer generation.
Handles job queuing, processing, and notifications.
"""

import asyncio
import os
from datetime import datetime
from typing import Optional, Dict, Any
import uuid
import json

# For Replit, you might use their built-in database or Redis
# This example uses a simple in-memory queue for demonstration

class FlyerGenerationQueue:
    """
    Manages the background job queue for flyer generation.
    """

    def __init__(self):
        self.jobs = {} # In production, use Redis or a proper queue
        self.orchestrator = None

    def initialize(self, db_connection, storage_client):
        """Initialize the orchestrator with dependencies."""
        from .orchestrator import FlyerGenerationOrchestrator
        self.orchestrator = FlyerGenerationOrchestrator(db_connection, storage_client)

    async def enqueue_job(
        self,
        industry: str,
        custom_prompt: str,
        contact_info: Dict[str, str],
        user_id: str,
        callback_url: Optional[str] = None

```

```

) -> str:
    """
    Add a new flyer generation job to the queue.
    Returns the job ID for tracking.
    """
    job_id = str(uuid.uuid4())

    job_data = {
        "id": job_id,
        "user_id": user_id,
        "industry": industry,
        "custom_prompt": custom_prompt,
        "contact_info": contact_info,
        "callback_url": callback_url,
        "status": "queued",
        "created_at": datetime.utcnow().isoformat(),
        "updated_at": datetime.utcnow().isoformat()
    }

    self.jobs[job_id] = job_data

    # Start processing in background
    asyncio.create_task(self._process_job(job_id))

    return job_id

async def _process_job(self, job_id: str):
    """Process a single job from the queue."""
    job = self.jobs.get(job_id)
    if not job:
        return

    try:
        job["status"] = "processing"
        job["updated_at"] = datetime.utcnow().isoformat()

        # Run the orchestrator
        pdf_url = await self.orchestrator.generate_flyer(
            request_id=job_id,
            industry=job["industry"],
            custom_prompt=job["custom_prompt"],
            contact_info=job["contact_info"],
            callback_url=job.get("callback_url")
        )

        job["status"] = "completed"
        job["pdf_url"] = pdf_url

```

```

        job["completed_at"] = datetime.utcnow().isoformat()

    except Exception as e:
        job["status"] = "failed"
        job["error"] = str(e)

    job["updated_at"] = datetime.utcnow().isoformat()

def get_job_status(self, job_id: str) -> Optional[Dict[str, Any]]:
    """Get the current status of a job."""
    return self.jobs.get(job_id)

# Global queue instance
flyer_queue = FlyerGenerationQueue()

# ===== API ENDPOINTS =====
# Add these to your Flask/FastAPI app

"""
Example Flask integration:

from flask import Flask, request, jsonify
from flyer_generator.background_worker import flyer_queue

app = Flask(__name__)

@app.route('/api/flyers/generate', methods=['POST'])
async def generate_flyer():
    data = request.json

    job_id = await flyer_queue.enqueue_job(
        industry=data.get('industry'),
        custom_prompt=data.get('custom_prompt', ''),
        contact_info=data.get('contact_info', {}),
        user_id=data.get('user_id'),
        callback_url=data.get('callback_url')
    )

    return jsonify({
        "success": True,
        "job_id": job_id,
        "message": "Flyer generation started. Check status at /api/flyers/status/"
    })

@app.route('/api/flyers/status/<job_id>', methods=['GET'])

```

```
def get_flyer_status(job_id):
    status = flyer_queue.get_job_status(job_id)

    if not status:
        return jsonify({"error": "Job not found"}), 404

    return jsonify(status)
"""
```

File 6: requirements.txt

```
reportlab>=4.0.0
google-generativeai>=0.3.0
aiohttp>=3.9.0
Pillow>=10.0.0
python-dotenv>=1.0.0
```

INTEGRATION WITH YOUR EXISTING REPLIT CODE

What to Tell Replit Agent:

CRITICAL INSTRUCTIONS FOR REPLIT AGENT:

1. DO NOT try to generate the entire flyer as a single AI image. This approach fundamentally cannot produce text-accurate, professional results.
2. EXAMINE the existing flyer templates in the system. These are your REFERENCE for layout and styling, not inputs to an image generator.
3. IMPLEMENT a multi-step pipeline:
 - Step 1: Use Gemini Flash to analyze the request and generate copy
 - Step 2: Use Gemini Imagen (or fallback to stock) for ONLY the hero image
 - Step 3: Use ReportLab to programmatically build the PDF with precise positioning
4. The PDF assembly step is where the magic happens - this is NOT image generation it's programmatic document construction using Python.
5. Run the generation in a background task and update status in the database. The frontend should poll for completion.
6. LOOK AT these files I'm providing and integrate them with the existing codebase

- `orchestrator.py` - coordinates the multi-agent pipeline
- `llm_agent.py` - handles copy generation
- `image_agent.py` - handles hero image generation
- `pdf_assembler.py` - builds the final PDF programmatically
- `background_worker.py` - handles async processing

7. The current "Create Custom Flyer" feature should call the background worker's `enqueue_job()` function instead of trying to generate an image directly.

QUICK START CHECKLIST

1. Install ReportLab: `pip install reportlab`
2. Set up Gemini API key in Replit Secrets
3. Create the `flyer_generator/` directory structure
4. Copy the Python files from this document
5. Update your existing flyer generation endpoint to use the new orchestrator
6. Add stock/fallback images for each industry
7. Test with a simple request
8. Refine the PDF layout to match your existing templates more precisely

DEBUGGING TIPS

If flyers are still blank:

1. Check ReportLab is installed: `python -c "from reportlab.pdfgen import canvas; print('OK')"`
2. Verify the output directory is writable
3. Check Gemini API key is set
4. Look at the job status endpoint for error messages
5. Check that hero images exist at the expected paths

If copy is generic:

1. Improve the LLM system prompt with more specific examples

2. Add your actual flyer text as few-shot examples
3. Increase the context about PCBancard's specific value props

If images don't match the industry:

1. Ensure you have good fallback stock images for each industry
2. Improve the image generation prompt with more specific scene descriptions
3. Consider curating a library of pre-approved hero images per industry