Email Report Delivery System - Technical Explanation

Architecture Overview

The system uses a client-server architecture:
- **Frontend**: JavaScript (vanilla) in `app.js` - runs in the browser
- **Backend**: Python Flask server (`backend.py`) - handles PDF generation and email sending
- **Proxy**: Node.js HTTP server (`server.js`) - routes API requests to the Python backend

Complete Flow

1. User Triggers Report Generation (Frontend - `app.js`)

When the user clicks "Generate Report", the `generateReport()` function (line 959 in `app.js`) executes:

```959:1066:app.js
async function generateReport() {
  // ... collects ranking data, captures canvas graph as base64 image ...

  const response = await fetch('/api/generate-report', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      rankingTable,
      podiumData,
      priorities: { r, g, b },
      graphImage,  // Base64 encoded PNG
      sessionId
    })
  });
```

What it sends:
- `rankingTable`: Sorted list of solutions with scores
- `podiumData`: Top 3 clusters (Gold, Silver, Bronze) with detailed solution info
- `priorities`: User's priority weights (cost, quality, deadline percentages)
- `graphImage`: Canvas element converted to base64 PNG string
- `sessionId`: User session identifier for tracking

2. Request Routing (Node.js Proxy - `server.js`)

The Node.js server (`server.js`) intercepts `/api/*` requests and proxies them to the Python backend:

```93:94:server.js
  if (filePath.startsWith('/api/')) {
```

```
    proxyToBackend(req, res);
```

The proxy forwards the request to `http://127.0.0.1:5000/api/generate-report` (Python Flask backend).

3. PDF Generation (Backend - `backend.py`)

The Flask endpoint `/api/generate-report` (line 358) handles the request:

```358:390:backend.py
@app.route('/api/generate-report', methods=['POST'])
def generate_report():
    """Generate PDF report and send via email"""
    try:
        data = request.json
        ranking = data.get('rankingTable', data.get('ranking', []))
        podium = data.get('podiumData', data.get('podium', []))
        priorities = data.get('priorities', {})
        graph_image_base64 = data.get('graphImage', '')
        session_id = data.get('sessionId', '')

        # Get current date and time
        now = datetime.now()
        date_str = now.strftime('%d/%m/%Y')
        time_str = now.strftime('%H:%M:%S')

        # ... get user IP and city from session data ...

        # Generate hash - usa rankingTable para consistência
        hash_data = f"{session_id}{now.isoformat()}{json.dumps(ranking, sort_keys=True)}"
        report_hash = hashlib.sha256(hash_data.encode()).hexdigest()[:16]

        # Create PDF in memory
        buffer = BytesIO()
        doc = SimpleDocTemplate(buffer, pagesize=A4, ...)
```

PDF Generation Process:
1. Creates an in-memory PDF using `reportlab` (`SimpleDocTemplate`)
2. Adds content:
   - Title, hash, IP, city
   - Ranking table with all solutions
   - Graph image (decoded from base64, converted from RGBA to RGB if needed)
   - Podium section with detailed solution data (cost, deadlines, quality metrics, risks, mitigations)
3. Builds the PDF into a byte buffer

Libraries used:
- `reportlab`: PDF generation
- `Pillow` (PIL): Image processing (converts base64 image, handles RGBA→RGB conversion)

4. Email Sending (Backend - `backend.py`)

After generating the PDF, the code attempts to send it via email:

```649:657:backend.py
        # Send email with PDF
        try:
            send_email_with_pdf(pdf_data, date_str, time_str, report_hash)
            print(f"✅ Email enviado com sucesso!")
        except Exception as e:
            print(f"❌ Erro ao enviar email: {e}")
            import traceback
            traceback.print_exc()
            # Continue even if email fails - PDF ainda será retornado
```

Email Configuration (function `send_email_with_pdf`, line 675):

```675:693:backend.py
def send_email_with_pdf(pdf_data, date_str, time_str, report_hash):
    """Send PDF report via email"""
    try:
        # Email configuration
        # You may need to configure these via environment variables
        smtp_server = os.getenv('SMTP_SERVER', 'smtp.gmail.com')
        smtp_port = int(os.getenv('SMTP_PORT', '587'))
        email_from = os.getenv('EMAIL_FROM', 'noetikaai@gmail.com')
        email_password = os.getenv('EMAIL_PASSWORD', '')
        email_to_str = os.getenv('EMAIL_TO', 'noetikaai@gmail.com,
gabriel.silva@ufabc.edu.br')

        # Parse multiple recipients (comma or semicolon separated)
        email_to_list = [email.strip() for email in email_to_str.replace(';', ',').split(',') if
email.strip()]
        if not email_to_list:
            email_to_list = ['noetikaai@gmail.com']

        if not email_password:
            print("⚠️ EMAIL_PASSWORD não configurado. Email não será enviado.")
            return
```

Email sending process:

1. Reads environment variables:
   - `SMTP_SERVER` (default: `smtp.gmail.com`)
   - `SMTP_PORT` (default: `587`)
   - `EMAIL_FROM` (default: `noetikaai@gmail.com`)
   - `EMAIL_PASSWORD` (required - no default)
   - `EMAIL_TO` (comma-separated recipients)
2. Creates MIME multipart message with:
   - Plain text body with report hash, date, time
   - PDF attachment (base64 encoded)
3. Connects to SMTP server via TLS
4. Authenticates using `EMAIL_PASSWORD`
5. Sends email to all recipients

Important: If `EMAIL_PASSWORD` is not set, the email is skipped but the function continues (the PDF is still returned to the user).

5. Response to Frontend

Regardless of email success/failure, the backend returns the PDF:

```659:667:backend.py
        # Return PDF
        from flask import Response
        return Response(
            pdf_data,
            mimetype='application/pdf',
            headers={
                'Content-Disposition': f'attachment;
filename=Tribussula_report_{date_str.replace("/", "")}_{time_str.replace(":", "")}.pdf'
            }
        )
```

The frontend receives the PDF blob and triggers a download:

```1072:1080:app.js
    const blob = await response.blob();
    const url = window.URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = `Tribussula_report_${new Date().toISOString().replace(/[:.]/g, '-').slice(0,
-5)}.pdf`;
    document.body.appendChild(a);
    a.click();
    document.body.removeChild(a);
    window.URL.revokeObjectURL(url);
```

Why It Might Fail on Railway

1. Missing system dependencies: `reportlab` and `Pillow` may need system libraries (fonts, image processing) that aren't installed in the Docker container.
2. Environment variables: `EMAIL_PASSWORD` must be set in Railway's environment variables dashboard.
3. Python dependencies: The Dockerfile installs Python packages, but if `reportlab` or `Pillow` fail to install, PDF generation will fail.
4. SMTP connectivity: Railway's network might block outbound SMTP connections (port 587), or Gmail might block authentication from Railway's IP ranges.

Error Handling

- Email failures are caught and logged, but don't stop PDF generation
- PDF generation errors return a 500 status with error details
- Frontend shows alerts if the backend is unavailable (502 Bad Gateway)

The system is designed to be resilient: even if email fails, the user still gets the PDF download.