**Project Proposal: Transition from Adobe API to Automated Document Generation System**

**Objective**

The purpose of this project is to replace the current reliance on Adobe API calls for document generation and processing with a cost-effective, scalable, and efficient in-house solution. This includes transitioning from Adobe's proprietary syntax to **Jinja2 templating**, which is more flexible and easier to maintain. Additionally, the project will integrate with **apitemplate.io** for template creation and testing, enabling faster iterations and direct uploads to AWS. The new system will dynamically populate Word templates, convert them to PDFs, and merge multiple PDFs, all while maintaining seamless integration with existing workflows.

**Why We Are Doing This**

1. **Eliminate Costly Adobe API Calls**:
   * Adobe API calls are expensive and introduce external dependencies. Transitioning to an in-house solution will significantly reduce recurring costs and improve control over the document generation process.
2. **Simplify Template Management**:
   * Adobe's syntax for templates is complex and difficult to maintain. Switching to **Jinja2** will reduce complexity, improve readability, and make templates easier to update.
   * Using **apitemplate.io** provides a visual editor (WYSIWYG) for creating and testing templates, enabling quicker iterations and validation of logic.
3. **Reuse Existing Integration Patterns**:
   * By reusing the same integration patterns currently used for Adobe APIs, we can minimize rework. This means only updating the API URL and authentication, without requiring changes to the Pega-side logic or moving templates from S3.
4. **Scalability and Flexibility**:
   * Azure Functions provides a serverless architecture that scales automatically based on demand, while docxtpl allows for advanced templating features like conditional logic and nested data.

**Scope**

This project is **narrowly focused** on replacing the Adobe API calls with an in-house solution. The scope includes:

1. **Document Templating**:
   * Transition templates from Adobe's syntax to **Jinja2**.
   * Use docxtpl to dynamically populate Word templates with JSON data.
   * Integrate with **apitemplate.io** for template creation, testing, and direct uploads to AWS.
2. **Document Conversion**:
   * Convert Word documents to PDFs using Microsoft Graph API or other open-source tools.
3. **PDF Merging**:
   * Combine multiple PDFs into a single document using signed URLs for input and output.
4. **Integration**:
   * Reuse the existing integration patterns for Adobe APIs, ensuring minimal changes to Pega-side logic and no need to move templates from S3.

**Key Features**

1. **Dynamic Document Templating**:
   * Populate Word templates with data from JSON files using docxtpl.
   * Support for nested data structures, conditional logic, and loops for complex templates.
2. **Document Conversion**:
   * Convert Word documents to PDFs without relying on Adobe APIs.
3. **PDF Merging**:
   * Combine multiple PDF files into a single document using signed URLs for input and output.
4. **Error Handling**:
   * Validate inputs (e.g., file types, JSON structure) and provide meaningful error messages for unsupported or invalid files.
5. **Template Management**:
   * Use **apitemplate.io** for creating, testing, and managing templates with a visual editor.

**How It Works**

1. **Input**:
   * Users provide signed URLs for Word templates, JSON data, or PDF files via API endpoints.
2. **Processing**:
   * **ReplacePlaceholders**: Populate Word templates with JSON data using docxtpl.
   * **CombinePages**: Merge multiple PDFs into a single document using signed URLs.
3. **Output**:
   * Save the processed document (Word or PDF) to a signed URL for download.

**Deliverables**

1. **Functional Azure Functions**:
   * ReplacePlaceholders: Populate Word templates with JSON data and save as PDF or Word document.
   * CombinePages: Merge multiple PDFs into one using signed URLs.
2. **Refactored Word Templates**:
   * Templates updated to use docxtpl with Jinja2 syntax.
3. **Integration with Existing Systems**:
   * Maintain the same integration patterns as Adobe APIs to minimize rework.
4. **Documentation**:
   * User guide for uploading files and using the API endpoints.
   * Developer guide for maintaining and extending the solution.

**Benefits**

1. **Cost Savings**:
   * Eliminates recurring costs associated with Adobe API calls.
2. **Simplified Template Management**:
   * Jinja2 syntax is easier to use and maintain, reducing complexity in templates.
   * **apitemplate.io** enables faster testing and validation of templates.
3. **Minimal Rework**:
   * Reuses existing integration patterns, requiring only changes to the API URL and authentication.
4. **Efficiency**:
   * Automates repetitive tasks, saving time and reducing manual errors.
5. **Scalability**:
   * Serverless architecture ensures the system can handle varying workloads.
6. **Flexibility**:
   * Supports complex templates with conditional logic and nested data.

**Timeline**

* **Week 1**: Requirements gathering and template analysis.
* **Week 2-3**: Development of Azure Functions and template refactoring.
* **Week 4**: Testing and deployment to Azure.
* **Week 5**: Documentation and final review.