# SPEC-1 – PowerPoint-to-PDF Label Generation on Microsoft Business

## Background

* **Problem**: Produce non-editable PDF labels for boxes from a designer-maintained PowerPoint template. Variable parts of the template are filled automatically by addressing shapes via the **Selection Pane** (shape names).
* **Final data flow**:
  1. User submits a form in **Power Apps**; **Power Automate Cloud (PAC)** writes a **JSON** job file into the **SharePoint Document Library** ServerPAD.
  2. The synced local folder is polled by **Power Automate Desktop (PAD)** every 5 seconds.
  3. When a new job arrives, PAD reads the JSON, loads the referenced PowerPoint template from the library, and caches values.
  4. For each label, a **QR Code** is generated locally using QRCoder.dll as **PNG**.
  5. **PowerPoint (COM/Interop)** fills text and images into named shapes (e.g., TXT\_PRODUCT\_NAME, IMG\_QR) and duplicates slides as needed.
  6. The presentation is exported to **PDF**.
  7. The PDF is saved back into the library; status is surfaced to Power Apps (status column or Flow callback).
* **Constraints/Prereqs**:
  + Environment is EN/FR; no Persian fonts required.
  + Always-on Windows agent with PowerPoint and PAD installed; local access to the synced SharePoint/OneDrive path.
  + **Idempotency** to avoid double-processing; concurrency control for multiple jobs.
  + Versioned, stable templates/fonts to guarantee print output.
* **Additional notes**:
  + The JSON acts as an instruction: **how many labels**, **which fields**, **which template**.
  + Printing/forwarding outside the library is out of scope for now.
  + PowerPoint is chosen to decouple creative layout work from automated PDF generation while staying within current licensing.

## Requirements

**MoSCoW**

**Must**

* Accept JSON with keys: job\_id, template\_path, language, and either labels[] or labels\_count plus shared fields.
* Support multi-slide jobs; each slide shows an item counter like i/N.
* Generate a **unique QR** per slide from “SKU + SERIAL” with configurable QR options.
* Map fields to PowerPoint shapes named in Selection Pane: TXT\_\* for text, IMG\_\* for images (e.g., IMG\_QR).
* Load template from the synced SharePoint path; export to PDF with embedded fonts; save back to the library.
* **Idempotency** keyed by job\_id; move/mark JSON as processing/done/error.
* Return job status to Power Apps.
* Execution/error logging.

**Should**

* Support multiple templates and versions (template\_version) and validate compatibility.
* Validate JSON against a schema (mandatory fields, types, lengths).
* Standardize output file naming (e.g., <job\_id>\_<template>\_<timestamp>.pdf).
* Concurrency control for ~5 jobs/min; up to 20 slides/job.
* QR options (ECC level, pixels per module, margin) configurable.

**Could**

* Generate a thumbnail (PNG of slide 1) for preview in Power Apps.
* Daily CSV summary of jobs/errors.
* Future support for other barcodes (Code128/Datamatrix).

**Won’t (for now)**

* Manage printing or external routing of PDFs.
* Use non-PowerPoint rendering engines.

## Method

### 1) JSON contract (proposed)

{  
 "job\_id": "9b2c6a7e-6c2e-4c7e-9e1e-12c0a3f1a001",  
 "template\_path": "Templates/Label\_v2.pptx",  
 "language": "en-CA",  
 "sku": "DLP-OM300-VU-GV-40-15D-M200-70-LN-BTU-DI",  
 "labels\_count": 60,  
 "serialProvider": {  
 "source": "local",  
 "format": "{JOB}-{SEQ:000000}",  
 "start": 1  
 },  
 "qr": {  
 "payload": "SKU|SERIAL",  
 "ecc": "Q",  
 "pixelsPerModule": 10,  
 "margin": 2  
 },  
 "fields": {  
 "TXT\_PRODUCT\_NAME": "Widget X",  
 "TXT\_PRODUCT\_VARIANT\_CODE": "{SKU}",  
 "TXT\_ITEM\_NO": "{I}/{N}",  
 "TXT\_PROJECT\_NO": "PRJ-2025-001",  
 "TXT\_PROPOSITION\_NO": "PROP-555",  
 "TXT\_CLIENT\_PO\_NO": "PO-123456",  
 "TXT\_CLIENT\_ADDRESS": "123 Rue Example\nMontreal, QC"  
 }  
}

*Instead of* ***labels\_count****, a* ***labels[]*** *array with precomputed serials is also supported.*

### 2) Serial strategy (pre-WMS)

* **Option A (recommended/MVP)**: local job-scoped serial: SERIAL = {JOB8}-{SEQ}; globally unique via job\_id and human-friendly; no central dependency.
* **Option B (centralized)**: SharePoint List + Flow allocates a **range** per job to avoid collisions across agents.
* **Option C (UUID/ULID)**: globally unique, less human-readable (fine for QR, not ideal for printed text).

Current decision: start with **A**; keep a switch to **B** later (serialProvider.source) without template changes.

### 3) Template mapping (Selection Pane)

* **Text**: shapes named TXT\_\* → Shape.TextFrame.TextRange.Text.
* **Images**: shapes named IMG\_\* → Shape.Fill.UserPicture(path) (preferred) or add a new picture at the placeholder’s bounds.
* **Counter**: TXT\_ITEM\_NO = i/N (zero-padded, e.g., 0001/0060).

### 4) Generation algorithm (pseudo)

for i in 1..N:  
 serial = (source==external) ? labels[i].serial : Format(format, JOB, i)  
 payload = Replace("SKU|SERIAL", {SKU, serial})  
 qrPng = QRCoder(payload, ecc=Q, ppm=10, margin=2)  
 slide = (i==1) ? open template first slide : duplicate base slide  
 set TXT\_\* fields  
 set IMG\_QR from qrPng (fill or add picture)  
export to PDF + thumbnail  
write status + path to SharePoint

### 5) Sequence diagram (PlantUML)

@startuml  
actor User  
participant "Power Apps" as PA  
participant "Power Automate Cloud" as PAC  
participant "SharePoint (ServerPAD)" as SP  
participant "PAD Agent" as PAD  
participant "LabelGen.ps1 (PowerShell)" as LG  
participant "PowerPoint COM" as PPT  
participant "QRCoder.dll" as QR  
  
User -> PA: Submit form (product\_variant\_code, language, count, template)  
PA -> PAC: Trigger flow  
PAC -> SP: Create job JSON (Jobs/incoming)  
PAD -> SP: Poll & move JSON to processing  
PAD -> LG: Run LabelGen.ps1 -JobJsonPath  
LG -> LG: Generate serials (JOB8-SEQ)  
LG -> QR: Make QR PNG (per label)  
LG -> PPT: Open template / duplicate slide(s)  
LG -> PPT: Fill TXT\_\* and IMG\_QR  
LG -> PPT: SaveAs(PDF) + Export thumbnail  
LG -> SP: Write Output/<job\_id>/... + status.json  
PAD -> SP: Move JSON to done/failed  
PAC -> PA: Return status + jobUrl  
@enduml

### 6) Print/QR parameters (proposed)

* **QR**: ECC=Q, pixelsPerModule=10, margin=2 (good for 300 DPI small labels).
* **PDF**: use **ExportAsFixedFormat/SaveAs(PDF)**; ensure required fonts are installed and embedded.

### 7) Serial Strategy & WMS alignment (deep dive)

**Option A — Local (Job-Scoped Counter)**

* Format: {JOB8}-{SEQ:000000}; QR payload: SKU|SERIAL (e.g., ABC-123|9B2C6A7E-000042).
* Pros: offline-friendly, fast, easy migration to B.
* Cons: uniqueness tied to job; WMS should normalize to an internal id.

**Option B — Central (Range Allocation)**

* SharePoint List SerialCounters(SKU, LastNumber, UpdatedAt) and a Flow AllocateRange(SKU, count) returning { start, end } atomically.
* Proposed org-wide serial for future: PLT-{SKU6}-{YYWW}-{NUM:00000} (configurable PLT, base36 SKU6).

**WMS minimal schema**

WMS.Items(ItemId PK, SKU UNIQUE, ...)  
WMS.SerialRanges(RangeId PK, SKU FK, StartNo, EndNo, AllocatedToJobId, AllocatedAt)  
WMS.Serials(SerialId PK, SKU FK, SerialText UNIQUE, JobId, Status ENUM('created','printed','consumed','scrapped'), CreatedAt)  
WMS.LabelJobs(JobId PK, TemplatePath, Language, Count, CreatedAt, Status)  
WMS.Labels(LabelId PK, JobId FK, PageNo, SerialId FK, QrPayload, PdfPath, PrintedAt)

## Implementation

### A) Folder structure (as in your SharePoint)

ServerPAD/  
 ├─ Cache/  
 ├─ Jobs/  
 │ ├─ incoming/  
 │ ├─ processing/ # <job\_id>.lock.json  
 │ ├─ done/  
 │ └─ failed/  
 ├─ Logs/ # agent.log + optional per-job logs  
 ├─ Output/  
 │ └─ <job\_id>/ # PDF + status + thumbnail  
 ├─ Templates/  
 └─ Tools/ # LabelGen.ps1, QRCoder.dll, configs

*Local synced path:* C:/Users/Learner/LightBase/Lightbase-Platform - ServerPAD

### B) JSON Schema (Draft 2020-12)

{  
 "$schema": "https://json-schema.org/draft/2020-12/schema",  
 "type": "object",  
 "required": ["job\_id", "template\_path", "language", "sku"],  
 "properties": {  
 "job\_id": {"type": "string", "format": "uuid"},  
 "template\_path": {"type": "string", "pattern": "^Templates/.+\\.pptx$"},  
 "language": {"type": "string", "enum": ["en-CA", "fr-CA"]},  
 "sku": {"type": "string", "minLength": 1},  
 "labels\_count": {"type": "integer", "minimum": 1},  
 "labels": {"type": "array", "items": {"type": "object", "properties": {"serial": {"type": "string"}}}},  
 "serialProvider": {  
 "type": "object",  
 "properties": {  
 "source": {"type": "string", "enum": ["local", "central"]},  
 "format": {"type": "string"},  
 "start": {"type": "integer", "minimum": 1}  
 },  
 "required": ["source", "format"]  
 },  
 "qr": {  
 "type": "object",  
 "properties": {  
 "payload": {"const": "SKU|SERIAL"},  
 "ecc": {"type": "string", "enum": ["L", "M", "Q", "H"], "default": "Q"},  
 "pixelsPerModule": {"type": "integer", "minimum": 3, "default": 10},  
 "margin": {"type": "integer", "minimum": 0, "default": 2}  
 },  
 "required": ["payload"]  
 },  
 "fields": {"type": "object"}  
 },  
 "oneOf": [  
 {"required": ["labels\_count"]},  
 {"required": ["labels"]}  
 ]  
}

### C) Power Automate Desktop (PAD) – recommended flow

**Variables**

* varIncoming = %SERVERPAD%\Jobs\incoming
* varProcessing = %SERVERPAD%\Jobs\processing
* varDone = %SERVERPAD%\Jobs\done
* varFailed = %SERVERPAD%\Jobs\failed
* varOutput = %SERVERPAD%\Output
* varTemplates = %SERVERPAD%\Templates
* rootShare = %SERVERPAD%

**Main loop**

1. Get files from incoming. If none → Wait 5s.
2. Move first file to processing as <job\_id>.lock.json (atomic rename).
3. Run PowerShell:

powershell.exe -ExecutionPolicy Bypass -File "%SERVERPAD%\Tools\LabelGen\LabelGen.ps1" -JobJsonPath "%jobMovedPath%" -Root "%SERVERPAD%"

1. On success: move JSON to done; on failure: move to failed and log.

### D) PowerShell-only path (no C#)

**Tools/LabelGen/LabelGen.ps1**

param(  
 [Parameter(Mandatory=$true)] [string]$JobJsonPath,  
 [string]$Root = "$env:SERVERPAD"  
)  
  
function Write-Log([string]$msg){  
 $logDir = Join-Path $Root 'Logs'  
 New-Item -ItemType Directory -Path $logDir -ErrorAction SilentlyContinue | Out-Null  
 ("[{0}] {1}" -f (Get-Date -Format o), $msg) | Add-Content -Path (Join-Path $logDir 'agent.log')  
}  
  
# Load job  
$job = Get-Content -Raw -Path $JobJsonPath | ConvertFrom-Json  
$jobId = $job.job\_id  
$sku = $job.sku  
$count = if ($job.labels\_count) { [int]$job.labels\_count } else { [int]$job.labels.Count }  
  
# Paths  
$templatePath = Join-Path $Root (Join-Path 'Templates' ($job.template\_path -replace '^Templates/',''))  
$outRoot = Join-Path $Root 'Output'  
$jobOut = Join-Path $outRoot $jobId  
$logsDir = Join-Path $Root 'Logs'  
$tempDir = Join-Path $env:TEMP "LabelGen\_$jobId"  
$null = New-Item -ItemType Directory -Path $tempDir -ErrorAction SilentlyContinue  
$null = New-Item -ItemType Directory -Path $jobOut -ErrorAction SilentlyContinue  
$null = New-Item -ItemType Directory -Path $logsDir -ErrorAction SilentlyContinue  
  
# QR config  
$ppm = $job.qr.pixelsPerModule; if (-not $ppm) { $ppm = 10 }  
$ecc = $job.qr.ecc; if (-not $ecc) { $ecc = 'Q' }  
$margin = $job.qr.margin; if (-not $margin) { $margin = 2 }  
  
# Load QRCoder  
Add-Type -Path (Join-Path $PSScriptRoot 'QRCoder.dll')  
$QRGen = New-Object QRCoder.QRCodeGenerator  
function New-QrPngBytes([string]$payload){  
 $ecl = [QRCoder.QRCodeGenerator+ECCLevel]::$ecc  
 $data = $QRGen.CreateQrCode($payload, $ecl)  
 $png = New-Object QRCoder.PngByteQRCode($data)  
 return $png.GetGraphic([int]$ppm)  
}  
  
# PowerPoint COM  
$ppt = New-Object -ComObject PowerPoint.Application  
$ppt.Visible = $false  
$pres = $ppt.Presentations.Open($templatePath, $false, $false, $false)  
  
# (optional) override slide size  
if ($job.pageSize){  
 $w = [double]$job.pageSize.width  
 $h = [double]$job.pageSize.height  
 if ($job.pageSize.unit -eq 'mm'){ $w = $w \* 2.83465; $h = $h \* 2.83465 }  
 if ($job.pageSize.unit -eq 'in'){ $w = $w \* 72; $h = $h \* 72 }  
 $pres.PageSetup.SlideWidth = [single]$w  
 $pres.PageSetup.SlideHeight = [single]$h  
}  
  
function Set-Text($slide, $name, $text){  
 foreach($s in @($slide.Shapes)){  
 if ($s.Name -eq $name -and $s.HasTextFrame){ $s.TextFrame.TextRange.Text = $text; return }  
 }  
}  
function Place-Image($slide, $name, $imgPath){  
 foreach($s in @($slide.Shapes)){  
 if ($s.Name -eq $name){  
 try { $s.Fill.UserPicture($imgPath) } catch { }  
 if (-not $?) { $slide.Shapes.AddPicture($imgPath, $false, $true, $s.Left, $s.Top, $s.Width, $s.Height) | Out-Null }  
 return  
 }  
 }  
}  
  
# Serial (Option A)  
$job8 = ($jobId -replace '-','').Substring(0,8).ToUpper()  
for($i=1; $i -le $count; $i++){  
 $serial = "{0}-{1}" -f $job8, ($i.ToString('000000'))  
 $payload = "{0}|{1}" -f $sku, $serial  
 $qrBytes = New-QrPngBytes $payload  
 $qrPath = Join-Path $tempDir ("qr\_{0}.png" -f $i)  
 [System.IO.File]::WriteAllBytes($qrPath, $qrBytes)  
  
 $slide = if($i -eq 1){ $pres.Slides.Item(1) } else { $pres.Slides.Item(1).Duplicate().Item(1) }  
 Set-Text $slide 'TXT\_PRODUCT\_VARIANT\_CODE' $sku  
 $pad = [math]::Max(4, $count.ToString().Length)  
 $iPadded = $i.ToString(("D{0}" -f $pad))  
 $nPadded = $count.ToString(("D{0}" -f $pad))  
 Set-Text $slide 'TXT\_ITEM\_NO' ("{0}/{1}" -f $iPadded, $nPadded)  
  
 if ($job.fields){  
 $job.fields.PSObject.Properties | ForEach-Object {  
 $val = [string]$\_.Value -replace '\{I\}',$i -replace '\{N\}',$count -replace '\{SKU\}',$sku  
 Set-Text $slide $\_.Name $val  
 }  
 }  
  
 Place-Image $slide 'IMG\_QR' $qrPath  
}  
  
# Export PDF + thumbnail + status  
$stamp = (Get-Date).ToString('yyyyMMdd-HHmmss')  
$pdfName = "{0}\_{1}\_{2}.pdf" -f $jobId, $sku, $stamp  
$pdfOut = Join-Path $jobOut $pdfName  
$pres.SaveAs($pdfOut, 32) # 32 = ppSaveAsPDF  
  
# thumbnail (slide 1 @ 150 DPI)  
$sw = [double]$pres.PageSetup.SlideWidth; $sh = [double]$pres.PageSetup.SlideHeight  
$thumbW = [int]([math]::Round(($sw/72.0)\*150))  
$thumbH = [int]([math]::Round(($sh/72.0)\*150))  
$thumb = Join-Path $jobOut 'thumb.png'  
$pres.Slides.Item(1).Export($thumb, 'PNG', $thumbW, $thumbH)  
  
$pres.Close(); $ppt.Quit()  
  
$status = @{ job\_id=$jobId; status='done'; pdf=$pdfName; folder=$jobOut; pages=$count; ts=(Get-Date) }  
$status | ConvertTo-Json | Set-Content -Path (Join-Path $jobOut ("{0}.status.json" -f $jobId))  
Write-Log ("DONE {0}" -f $pdfOut)  
exit 0

### E) Execution & naming

* Output: Output/<job\_id>/<job\_id>\_<sku>\_<yyyymmdd-HHmmss>.pdf
* Status file: Output/<job\_id>/<job\_id>.status.json with { status, pdf, folder, pages, ts }

### F) Power Automate Cloud (PAC) – Flow design

1. **Trigger**: Power Apps (product\_variant\_code, language, count, template, fieldsJson).
2. **Variables**: Job ID = guid(), Site = /sites/Lightbase-Platform, IncomingFolder = /ServerPAD/Jobs/incoming.
3. **Compose Job (JSON)**

{  
 "job\_id": "@{variables('Job ID')}",  
 "template\_path": "@{concat('Templates/', triggerBody()?['template'])}",  
 "language": "@{triggerBody()?['language']}",  
 "sku": "@{triggerBody()?['product\_variant\_code']}",  
 "labels\_count": @{int(triggerBody()?['count'])},  
 "serialProvider": { "source": "local", "format": "{JOB}-{SEQ:000000}", "start": 1 },  
 "qr": { "payload": "SKU|SERIAL", "ecc":"Q", "pixelsPerModule":10, "margin":2 },  
 "fields": @{json(triggerBody()?['fieldsJson'])}  
}

1. **Create file (SharePoint)** → Library: ServerPAD/Jobs/incoming, Name: @{concat(variables('Job ID'), '.json')}, Content: Compose output.
2. **Respond to Power App** with status, jobUrl.

### G) Power Apps – Power Fx snippets

**Templates dropdown (from** ``**)**

ddTemplate.Items =  
Sort(  
 Filter(  
 ServerPAD,  
 'Folder Path' = "/sites/Lightbase-Platform/ServerPAD/Templates/" &&  
 IsFolder = false &&  
 EndsWith(Name, ".pptx")  
 ),  
 Name,  
 Ascending  
)

**Generate button**

Set(  
 res,  
 GenerateLabelJob.Run(  
 txtProductVariantCode.Text, // product\_variant\_code → used as sku  
 drpLang.Selected.Value,  
 Value(txtCount.Text),  
 ddTemplate.Selected.Name,  
 JSON({  
 TXT\_PRODUCT\_NAME: txtProductName.Text,  
 TXT\_PROJECT\_NO: txtProjectNo.Text,  
 TXT\_PROPOSITION\_NO: txtPropositionNo.Text,  
 TXT\_CLIENT\_PO\_NO: txtClientPONo.Text,  
 TXT\_CLIENT\_ADDRESS: txtClientAddress.Text  
 }, JSONFormat.Compact)  
 )  
);  
If(res.status = "created",  
 Notify("Job queued", NotificationType.Success),  
 Notify("Failed to queue job", NotificationType.Error)  
);

### H) Template guidance for designers

* Name shapes (Selection Pane):
  + TXT\_PROJECT\_NO, TXT\_PROPOSITION\_NO, TXT\_CLIENT\_PO\_NO,
  + TXT\_PRODUCT\_NAME, TXT\_PRODUCT\_VARIANT\_CODE, TXT\_CLIENT\_ADDRESS (multi-line),
  + TXT\_ITEM\_NO (zero-padded i/N, e.g., 0001/0060),
  + IMG\_QR.
* Slide size = actual label size; keep safe margins; avoid unpredictable AutoFit.
* Install and embed required fonts; test long strings (e.g., product variant code).

### I) Future Option B (org-wide serial)

* Serial format: PLT-{SKU6}-{YYWW}-{NUM:00000} (configurable PLT, base36 SKU6).
* Range allocation via Flow to minimize contention; QR stays SKU|SERIAL.

## Milestones

* **M0 – Environment prep (1d)**: Folders, SERVERPAD, place Tools/LabelGen.ps1 & QRCoder.dll. **DOD**: script help runs; agent access OK.
* **M1 – Templates (1–2d)**: at least 2 sizes; placeholders as listed. **DOD**: manual fill test OK.
* **M2 – PAC (1d)**: Flow creates JSON in Jobs/incoming and returns status/url. **DOD**: JSON matches schema.
* **M3 – PAD (1–2d)**: poll/move/execute/log; **DOD**: end-to-end to Output/<job\_id>/ (PDF+status+thumb).
* **M4 – E2E MVP (1d)**: scenarios (1×, 20×, all fields). **DOD**: scannable QR; correct TXT\_ITEM\_NO.
* **M5 – Quality/Stability (1d)**: DPI/size/font embed, retry, error handling. **DOD**: ≥99% success over 100 labels.
* **M6 – Security/Access (0.5d)**: service account, least privilege, logging policy. **DOD**: documented.
* **M7 – Deploy & Train (0.5d)**: deploy to main agent + quick guide. **DOD**: handover checklist.

## Gathering Results

**Acceptance**

* All placeholders filled; TXT\_ITEM\_NO zero-padded (e.g., 0001/0060).
* QR strictly SKU|SERIAL and scans in 3 sample apps.
* PDF has correct slide size and embedded fonts; print-quality (≥300 DPI).
* Outputs in Output/<job\_id>/ (PDF, status.json, thumb.png), job JSON moved to Jobs/done/.

**KPIs**

* SLA: ≤20s for 10 labels; ≤90s for 100 labels.
* Throughput: ≥5 jobs/min with tuned concurrency.
* Reliability: ≥99% success in a 1-week pilot.
* Idempotency: re-running same job\_id does not duplicate outputs.

**Test scenarios**

1. Single label with all fields.
2. 60 labels: padding and QR uniqueness.
3. Concurrent load: 5 jobs × 20 labels.
4. Error paths: missing template/bad shape name/QRCoder failure → moved to failed with message.
5. Fonts: embed check & rendering on a second machine.

**Monitoring & reporting**

* Logs/agent.log (+ optional per-job logs), optional CSV/Power BI from Jobs/done/failed.

**Retention**

* Jobs (done|failed|processing) purge/archive >30 days.
* Logs rotation up to 100MB, keep 14 days.
* Output keep 90 days (tunable), then archive.

**Migration to Option B**

* Add a range allocator Flow (SerialCounters), set serialProvider.source='central'.
* Regression-safe: template and QR unchanged.

## Need Professional Help in Developing Your Architecture?

Please contact me at [sammuti.com](https://sammuti.com) :)