One-Click KDP Formatter — Starter Pack (MVP)

This canvas gives you a working blueprint for a single-use SaaS that takes messy manuscripts (TXT/DOCX/PDF), detects structure with an LLM, then renders **KDP-ready print PDFs** and a **Kindle-ready EPUB** deterministically.

---

1) Repo layout

```
/kdp-formatter
  /app
    app.py                # FastAPI service (upload/format/download)
    pipeline.py           # Ingestion → IDM → Render → Validate
    preflight.py          # PDF/EPUB checks
    gutter.py             # Page-count → gutter lookups
    image_tools.py        # DPI/colour/flatten utilities
    epub.py               # EPUB build helpers
  /templates
    base.html.j2          # Book body template (paged media)
    kdp.css.j2            # Print CSS (mirrored margins, folios, recto starts)
    epub.css              # Simple reflowable CSS
  /schemas
    idm.schema.json       # Intermediate Document Model (IDM)
    api.schema.json       # API input/output contracts
  /bin
    render_local.py       # CLI wrapper for quick local runs
  /fonts
    SourceSerif4-Regular.ttf
    SourceSerif4-Semibold.ttf
```

---

2) IDM (Intermediate Document Model) — JSON Schema (excerpt)

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "IDM",
  "type": "object",
  "required": ["meta", "blocks"],
  "properties": {
    "meta": {
      "type": "object",
      "required": ["title", "author", "language", "trim", "bleed", "interior"],
```

```
      "properties": {
        "title": {"type": "string"},
        "subtitle": {"type": "string"},
        "author": {"type": "string"},
        "language": {"type": "string", "default": "en"},
        "trim": {"type": "string", "enum": ["5x8","5.5x8.5","6x9","7x10"],
"default": "6x9"},
        "bleed": {"type": "boolean", "default": false},
        "interior": {"type": "string", "enum": ["bw","colour"], "default":
"bw"},
        "startRecto": {"type": "boolean", "default": true}
      }
    },
    "blocks": {
      "type": "array",
      "items": {
        "oneOf": [
          {"type": "object", "properties": {"type": {"const": "front-matter"},
"role": {"type":"string"}, "content": {"type":"array"}}},
          {"type": "object", "properties": {"type": {"const": "toc"}, "auto":
{"type":"boolean"}}},
          {"type": "object", "properties": {"type": {"const": "chapter"},
"label": {"type":"string"}, "number": {"type":"integer"}, "blocks":
{"type":"array"}}}
        ]
      }
    }
  }
}
```

Block atoms (used inside chapter `blocks`):

```
{"type":"heading","level":1,"text":"Chapter One"}
{"type":"para","text":"Paragraph text…"}
{"type":"quote","text":"Quoted line…"}
{"type":"image","src":"img001.png","caption":"…","full_bleed":false}
{"type":"hr"}
{"type":"footnote","id":"fn1","text":"…"}
```

---

3) API contracts (schemas/api.schema.json — abbreviated)

```
{
  "UploadResponse": {"type":"object","properties":{"file_id":
```

```
{"type":"string"}}},
  "FormatRequest": {
    "type":"object",
    "properties": {
      "file_id": {"type":"string"},
      "options": {
        "type":"object",
        "properties": {
          "trim": {"type":"string","default":"6x9"},
          "bleed": {"type":"boolean","default":false},
          "interior": {"type":"string","enum":["bw","colour"],"default":"bw"},
          "startRecto": {"type":"boolean","default":true},
          "bodyFont": {"type":"string","default":"SourceSerif4-Regular"},
          "headingFont": {"type":"string","default":"SourceSerif4-Semibold"}
        }
      }
    },
    "required": ["file_id"]
  },
  "FormatResponse": {"type":"object","properties":{"job_id":{"type":"string"}}}
}
```

---

4) Paged-Media CSS (templates/kdp.css.j2)

```
/* KDP Print CSS — generated with Jinja variables */
/* Trim size */
@page { size: {{ trim_width_in }}in {{ trim_height_in }}in; marks: none; }

/* Mirrored margins with gutter. For left (verso) inner margin is left; for
right (recto) inner margin is right */
@page:left  { margin: {{ top_in }}in {{ outer_in }}in {{ bottom_in }}in {{
inner_in }}in; }
@page:right { margin: {{ top_in }}in {{ inner_in }}in {{ bottom_in }}in {{
outer_in }}in; }

/* Folios (page numbers) bottom-centre; suppressed on chapter openers and front
matter */
@page { @bottom-center { content: counter(page); font-size: 9pt; } }
.front-matter, .chapter-opener { page: nofolio; }
@page:blank { @bottom-center { content: ""; } }

/* Running headers with chapter title on outer edge */
:root { --running-size: 9pt; }
.chapter h1 { string-set: chapter-title content(text); }
```

```
@page:right { @top-right { content: string(chapter-title); font-size: var(--
running-size);} }
@page:left  { @top-left  { content: string(chapter-title); font-size: var(--
running-size);} }

/* Typography */
html { hyphens: auto; }
body { font: 11.5pt/1.35 "{{ body_font }}"; color:#111; }
h1, h2, h3 { font-family: "{{ heading_font }}"; margin: 1.8em 0 0.6em; }
h1 { font-size: 20pt; }
h2 { font-size: 16pt; }
h3 { font-size: 13pt; }

p { margin: 0 0 0.9em; orphans: 2; widows: 2; }
blockquote { margin: 0.9em 1.2em; font-style: italic; }

/* Chapter openers: start on recto; suppress header/footer */
.chapter-opener { page-break-before: right; break-before: right; }
.chapter-opener h1 { margin-top: 35%; text-align: center; }
.chapter-opener + p { text-indent: 1.2em; }

/* Images */
.figure { text-align:center; break-inside: avoid; }
.figure img { max-width: 100%; height: auto; }
.caption { font-size: 9pt; font-style: italic; }

/* Front matter: no folios */
.front-matter { counter-reset: page 1; }
.front-matter, .nofolio { counter-increment: page 0; }

/* Optional bleed handling (content should extend to edges when {{ bleed }} is
true). For interiors, outer/top/bottom bleed only. */
{% if bleed %}
@page { size: {{ bleed_width_in }}in {{ bleed_height_in }}in; }
/* When using bleed, increase outer/top/bottom margins by bleed amount to keep
the text block identical. */
@page:left  { margin: {{ top_bleed_in }}in {{ outer_bleed_in }}in {{
bottom_bleed_in }}in {{ inner_in }}in; }
@page:right { margin: {{ top_bleed_in }}in {{ inner_in }}in {{
bottom_bleed_in }}in {{ outer_bleed_in }}in; }
{% endif %}
```

The CSS is rendered with numeric values (inches) injected from the gutter/margin calculator so every PDF is geometrically correct for the chosen trim and page count. WeasyPrint/Prince/Vivliostyle support these paged-media features.

---

5) HTML body template (templates/base.html.j2 — abbreviated)

```
<!doctype html>
<html lang="{{ meta.language }}">
  <head>
    <meta charset="utf-8" />
    <title>{{ meta.title }}</title>
    <link rel="stylesheet" href="kdp.css" />
    <style>
      @font-face { font-family: "{{ body_font }}"; src: url("../fonts/
SourceSerif4-Regular.ttf") format("truetype"); }
      @font-face { font-family: "{{ heading_font }}"; src: url("../fonts/
SourceSerif4-Semibold.ttf") format("truetype"); }
    </style>
  </head>
  <body>
    <section class="front-matter title-page nofolio">
      <h1>{{ meta.title }}</h1>
      {% if meta.subtitle %}<h2>{{ meta.subtitle }}</h2>{% endif %}
      <p>{{ meta.author }}</p>
    </section>

    {% if toc %}
    <section class="front-matter nofolio">
      <h1>Contents</h1>
      <ol>
        {% for ch in chapters %}
          <li>{{ ch.label }}</li>
        {% endfor %}
      </ol>
    </section>
    {% endif %}

    {% for chapter in chapters %}
      <section class="chapter">
        <div class="chapter-opener">
          <h1>{{ chapter.label }}</h1>
        </div>
        {% for b in chapter.blocks %}
          {% if b.type == 'heading' %}
            <h{{ b.level }}>{{ b.text }}</h{{ b.level }}>
          {% elif b.type == 'para' %}
            <p>{{ b.text }}</p>
          {% elif b.type == 'quote' %}
            <blockquote>{{ b.text }}</blockquote>
          {% elif b.type == 'image' %}
```

```
            <figure class="figure">
              <img src="{{ b.src }}" alt="{{ b.caption or '' }}"/>
              {% if b.caption %}<figcaption class="caption">{{ b.caption }}</
figcaption>{% endif %}
            </figure>
          {% elif b.type == 'hr' %}
            <hr />
          {% endif %}
        {% endfor %}
      </section>
    {% endfor %}
  </body>
</html>
```

---

6) Gutter calculator (app/gutter.py)

```
from dataclasses import dataclass

@dataclass
class Margins:
    top: float
    bottom: float
    outer: float
    inner: float

# KDP-style outer margins (inches)
MIN_OUTER_NO_BLEED = 0.25
MIN_OUTER_WITH_BLEED = 0.375
BLEED = 0.125

# Basic gutter lookup by page count bands (inches)
# 24–150: 0.375 | 151–300: 0.5 | 301–500: 0.625 | 501–700: 0.75 | 701–828: 0.875
BANDS = [
    (24, 150, 0.375),
    (151, 300, 0.5),
    (301, 500, 0.625),
    (501, 700, 0.75),
    (701, 828, 0.875),
]

TRIMS = {"5x8": (5.0, 8.0), "5.5x8.5": (5.5, 8.5), "6x9": (6.0, 9.0), "7x10":
(7.0, 10.0)}

def gutter_for_pages(pages:int)->float:
```

```
    for lo, hi, gut in BANDS:
        if lo <= pages <= hi:
            return gut
    # Fallback for >828
    return 1.0

def page_geometry(trim:str, pages:int, bleed:bool)->dict:
    w,h = TRIMS[trim]
    outer = MIN_OUTER_WITH_BLEED if bleed else MIN_OUTER_NO_BLEED
    top = bottom = MIN_OUTER_WITH_BLEED if bleed else MIN_OUTER_NO_BLEED
    inner = gutter_for_pages(pages)

    if bleed:
        bleed_w = w + (2*BLEED)  # outer + inner not bled in binding, but we
keep page box larger; text block adjusted
        bleed_h = h + (2*BLEED)
        return {
            "trim_w": w, "trim_h": h,
            "page_w": bleed_w, "page_h": bleed_h,
            "margins": Margins(top+BLEED, bottom+BLEED, outer+BLEED, inner),
            "bleed": BLEED,
        }
    else:
        return {
            "trim_w": w, "trim_h": h,
            "page_w": w, "page_h": h,
            "margins": Margins(top, bottom, outer, inner),
            "bleed": 0.0,
        }
```

---

7) FastAPI pipeline (app/app.py — abbreviated)

```
from fastapi import FastAPI, UploadFile, File
from fastapi.responses import FileResponse, JSONResponse
import uuid, os, json
from pipeline import build_idm_from_upload, render_print_pdf, build_epub,
preflight_report

app = FastAPI()
WORK = "/tmp/kdp"
os.makedirs(WORK, exist_ok=True)

@app.post("/upload")
async def upload(file: UploadFile = File(...)):
```

```
    fid = str(uuid.uuid4())
    path = os.path.join(WORK, fid + "_orig")
    with open(path, "wb") as f:
        f.write(await file.read())
    return {"file_id": fid}

@app.post("/format")
async def format(req: dict):
    fid = req["file_id"]
    options = req.get("options", {})
    idm = build_idm_from_upload(os.path.join(WORK, fid + "_orig"))
    job_id = str(uuid.uuid4())
    outdir = os.path.join(WORK, job_id)
    os.makedirs(outdir, exist_ok=True)

    pdf_path = render_print_pdf(idm, outdir, options)
    epub_path = build_epub(idm, outdir, options)
    report_path = preflight_report(pdf_path, epub_path, outdir)

    with open(os.path.join(outdir, "manifest.json"), "w") as f:
        json.dump({"pdf": pdf_path, "epub": epub_path, "report": report_path},
f)

    return {"job_id": job_id}

@app.get("/download")
async def download(job_id: str, type: str):
    manifest = json.load(open(os.path.join(WORK, job_id, "manifest.json")))
    path = manifest[type]
    return FileResponse(path, filename=os.path.basename(path))
```

---

8) Ingestion → IDM (app/pipeline.py — key ideas)

```
import os, json, subprocess, tempfile
from jinja2 import Environment, FileSystemLoader
from PyPDF2 import PdfReader
from gutter import page_geometry
from image_tools import normalise_images

TEMPLATES = Environment(loader=FileSystemLoader("templates"))

# 1) Convert to raw HTML/text then ask LLM to tag structure → IDM

def build_idm_from_upload(src_path: str) -> dict:
```

```
    ext = os.path.splitext(src_path)[1].lower()
    if ext in [".docx", ".md", ".txt"]:
        # Pandoc to plain text (keeps paragraph spacing); could also use HTML
        txt = subprocess.check_output(["pandoc", src_path, "-t",
"plain"]).decode("utf-8")
    elif ext == ".pdf":
        txt = subprocess.check_output(["pdftotext", src_path,
"-"]).decode("utf-8")
    else:
        raise ValueError("Unsupported file type")

    # Call your LLM with a deterministic JSON schema prompt to tag chapters/
headings/paras
    idm = call_llm_to_idm(txt)
    return idm

# 2) Render print PDF with correct geometry (two-pass to compute gutter from
page count)

def render_print_pdf(idm: dict, outdir: str, options: dict) -> str:
    # First pass: assume 200 pages for provisional margins
    geom = page_geometry(options.get("trim", "6x9"), 200, options.get("bleed",
False))
    html, css = render_html_css(idm, geom, options)

    pdf_path = os.path.join(outdir, "print_pass1.pdf")
    weasy_write_pdf(html, css, pdf_path)

    # Count pages
    pages = len(PdfReader(pdf_path).pages)
    geom = page_geometry(options.get("trim", "6x9"), pages, options.get("bleed",
False))
    html, css = render_html_css(idm, geom, options)

    final_pdf = os.path.join(outdir, "title_print.pdf")
    weasy_write_pdf(html, css, final_pdf)

    normalise_images(final_pdf, bw=(options.get("interior","bw")=="bw"))
    return final_pdf

# 3) EPUB build (reflowable)

def build_epub(idm: dict, outdir: str, options: dict) -> str:
    html = render_epub_html(idm)
    epub_path = os.path.join(outdir, "title_kindle.epub")
    with open(os.path.join(outdir, "book.html"), "w") as f: f.write(html)
    subprocess.check_call(["pandoc", os.path.join(outdir, "book.html"), "-o",
epub_path,
```

```
                            "--css=templates/epub.css", "--metadata",
f"title={idm['meta']['title']}"])
    return epub_path

# Helpers

def render_html_css(idm, geom, options):
    html = TEMPLATES.get_template("base.html.j2").render(
        meta=idm["meta"], chapters=[b for b in idm["blocks"] if
b.get("type")=="chapter"],
        toc=True, body_font=options.get("bodyFont","SourceSerif4-Regular"),
        heading_font=options.get("headingFont","SourceSerif4-Semibold")
    )
    css = TEMPLATES.get_template("kdp.css.j2").render(
        trim_width_in=geom["page_w"], trim_height_in=geom["page_h"],
        top_in=geom["margins"].top, bottom_in=geom["margins"].bottom,
        inner_in=geom["margins"].inner, outer_in=geom["margins"].outer,
        bleed=(geom["bleed"]>0), bleed_width_in=geom.get("page_w"),
bleed_height_in=geom.get("page_h"),
        top_bleed_in=geom["margins"].top,
bottom_bleed_in=geom["margins"].bottom, outer_bleed_in=geom["margins"].outer,
        body_font=options.get("bodyFont","SourceSerif4-Regular"),
heading_font=options.get("headingFont","SourceSerif4-Semibold")
    )
    return html, css

def weasy_write_pdf(html_str, css_str, out_path):
    from weasyprint import HTML, CSS
    HTML(string=html_str).write_pdf(out_path, stylesheets=[CSS(string=css_str)])
```

---

9) Image & preflight utilities (abbreviated)

app/image_tools.py

```
from PIL import Image
import tempfile, subprocess

def dpi_at_size(img_path):
    with Image.open(img_path) as im:
        dpi = im.info.get('dpi', (72,72))[0]
        return dpi

# Optionally force grayscale for B/W interiors and flatten transparency
```

```
def normalise_images(pdf_path, bw=True):
    # Use Ghostscript to process pages into a clean PDF; adjust settings as
needed
    gs = ["gs", "-dBATCH", "-dNOPAUSE", "-sDEVICE=pdfwrite",
          "-dDetectDuplicateImages=true", "-dCompressFonts=true",
          "-sOutputFile="+pdf_path.replace('.pdf','_norm.pdf'), pdf_path]
    subprocess.check_call(gs)
```

app/preflight.py

```
import json, subprocess, os
from PyPDF2 import PdfReader

def preflight_report(pdf_path, epub_path, outdir):
    pages = len(PdfReader(pdf_path).pages)
    # Very light checks; expand with pdffonts/pdfinfo/epubcheck in your
environment
    checks = {
        "pdf_pages": pages,
        "pdf_size": os.path.getsize(pdf_path),
        "epub_size": os.path.getsize(epub_path)
    }
    rep = os.path.join(outdir, "preflight_report.json")
    json.dump(checks, open(rep, "w"))
    return rep
```

---

10) Prompt stub for LLM structure-tagging (deterministic JSON)

```
SYSTEM: You convert raw manuscript text into a strict JSON structure (IDM
blocks). Do not rephrase prose. Output JSON only.

USER: Detect and number chapters; tag headings (H1–H3), paragraphs, quotes,
images with captions, and footnotes. Infer chapter labels like "Chapter 1" if
missing, but never change or invent body text. Return {meta:
{title,author,language}, blocks:[…]}.

TEXT:
{{chunk_text}}
```

---

11) How the two-pass gutter works

1. Render once with provisional margins to count pages.
2. Look up the correct **gutter** for that page count.
3. Re-render with mirrored margins (inner = gutter, outer = min outer).
4. Produce final print PDF.

---

12) Quick local run (bin/render_local.py)

```
# python bin/render_local.py manuscript.docx --trim 6x9 --bleed 0 --interior bw
```

Notes

- Default to **no-bleed** interiors for v1; enable bleed only when pages include true edge-to-edge graphics.
- Always embed fonts in the renderer and avoid any crop/registration marks in the PDF.
- Validate EPUB with `epubcheck`; smoke-test in Kindle Previewer as part of CI.