Architectural Specification and Master Directive for Automated KDP Publishing Engine

1. Executive Technical Overview and System Architecture

The development of a comprehensive web application designed to programmatically generate Amazon Kindle Direct Publishing (KDP) compliant manuscripts constitutes a significant engineering challenge. This system must function not merely as a content management system, but as a rigid print-physics engine capable of enforcing strict manufacturing constraints across a diverse array of literary genres. The target architecture leverages the Google Antigravity IDE, utilizing a multi-agent "Mission Control" paradigm to orchestrate the complex interplay between creative generation, layout algorithms, and compliance validation.

The overarching objective is to decouple the creative input from the technical output. A user—whether an author or a publisher—should interact with high-level parameters (genre, trim size, page count), while the underlying system autonomously handles the Cartesian mathematics of bleed, the shifting geometry of gutters based on paper thickness, and the metadata intricacies of fixed-layout versus reflowable digital formats. This report serves as the definitive technical specification for this system, breaking down the requirements for distinct genre modules: Visual & Illustrated (Manga, Comics, Children's), Non-Fiction (Biography, Business, Self-Help), and Activity/Low-Content (Coloring, Journals, Planners).

1.1 The Antigravity Agentic Paradigm

In the context of the Antigravity IDE, the development workflow shifts from manual coding to agent orchestration. The "Master Prompt" provided in Section 10 of this report is designed to configure the **Agent Manager**—the central nervous system of the IDE. We define specific "Agents" with distinct roles:

- **The Structural Engineer:** Responsible for calculating canvas dimensions, safe zones, and spine widths based on strict mathematical formulas.
- **The Layout Compositor:** Handles the placement of assets (text, images, puzzles) onto the canvas, respecting the constraints set by the Engineer.
- **The Compliance Officer:** A validation agent that runs pre-flight checks against KDP's published constraints (e.g., checking for text in the bleed zone or low-resolution images).
- **The Polyglot Translator:** Manages the conversion of internal data structures into format-specific outputs (PDF/X-1a for Print, EPUB3/KPF for Ebook).

1.2 Core Compliance Architecture

The fundamental requirement of this system is adherence to KDP's manufacturing tolerances. Unlike digital-first platforms, print-on-demand (POD) introduces physical variables that must be accounted for programmatically.

- **Dynamic Page Geometry:** The physical width of a page is not constant; it is a function of the binding type. As the page count increases, the inner margin (gutter) must expand to prevent content from vanishing into the glue binding.
- **Color Space Management:** The system must strictly manage color profiles. While screens operate in RGB, the print engine operates in CMYK. The application must handle the conversion of user-uploaded RGB assets into CMYK (specifically SWOP or Fogra profiles) to prevent unexpected color shifts, or desaturate them to Grayscale for black-and-white interiors to avoid "rich black" registration errors.[1]
- **Digital Rights Management (DRM) & Metadata:** For Ebooks, the system must generate precise OPF (Open Packaging Format) metadata to control reading direction (critical for Manga), fixed-layout behavior, and navigation structures (NCX).[3]

---

2. The Physics of Print: Global Manufacturing Constraints

Before delineating genre-specific requirements, the system must implement a "Global Physics Engine" that applies to all paperback and hardcover outputs. This engine calculates the immutable boundaries of the page.

2.1 Trim, Bleed, and Safety Logic

The distinction between "Trim Size" and "PDF Size" is the most common failure point in automated publishing. The application must support KDP's standard trim sizes while enforcing bleed logic.

- **Trim Size ($W_{trim} \times H_{trim}$):** The final cut size of the book.
- **Bleed Area:** If a book contains elements that extend to the edge of the page (common in Manga, Children's Books, and Coloring Books), the PDF must be physically larger than the trim size. KDP requires exactly $0.125”$ (3.2 mm) added to the **top**, **bottom**, and **outside** edge. Crucially, bleed is *never* added to the inside (binding) edge.
- **Live Element Margin (Safe Zone):** All critical content (text, page numbers, barcodes) must reside at least $0.25”$ (6.4 mm) inside the trim line.

Mathematical Definition for PDF Generation:

If Bleed == True:

$$PDF_{Width} = W_{trim} + 0.125”$$

$$PDF_{Height} = H_{trim} + 0.25” \ (0.125” \text{ top} + 0.125” \text{ bottom})$$

If Bleed == False:

$$PDF_{Width} = W_{trim}$$

$$PDF_{Height} = H_{trim}$$

2.2 Variable Gutter Calculation Algorithms

The application cannot use a static margin. The "Gutter" (inside margin) is a dynamic variable dependent on the total page count (N_{pages}). As the book block thickens, the bending radius of the spine increases, necessitating a wider inner margin to maintain legibility. The application must implement a lookup table or conditional logic to enforce these minimums.[1]

Table 1: Dynamic Gutter Specifications

Page Count (Npages)	Minimum Gutter (Inside Margin)	Outside Margin (No Bleed)	Outside Margin (With Bleed)
24 – 150	$0.375’$ (9.6 mm)	$0.25”$ (6.4 mm)	$0.375’$ (9.6 mm)
151 – 300	$0.500’$ (12.7 mm)	$0.25”$ (6.4 mm)	$0.375’$ (9.6 mm)
301 – 500	$0.625’$ (15.9 mm)	$0.25”$ (6.4 mm)	$0.375’$ (9.6 mm)
501 – 700	$0.750’$ (19.1 mm)	$0.25”$ (6.4 mm)	$0.375’$ (9.6 mm)
701 – 828	$0.875’$ (22.3 mm)	$0.25”$ (6.4 mm)	$0.375’$ (9.6 mm)

Implication: The rendering engine must be capable of "Shifting" content. For a book that grows from 150 to 151 pages, the entire text block on every page must shift outward by

$0.125”$.

2.3 Spine Width Mechanics

The cover file is a single continuous PDF containing the Back Cover, Spine, and Front Cover. The width of the spine is strictly determined by the paper type and page count. The application must calculate this to the thousandth of an inch.[1]

Spine Width (W_{spine}) Formulas:

1. **Black Ink / White Paper:** $N_{pages} \times 0.002252”$ (0.0572 mm)
2. **Black Ink / Cream Paper:** $N_{pages} \times 0.0025”$ (0.0635 mm)
3. **Standard Color / White Paper:** $N_{pages} \times 0.002252”$ (0.0572 mm)
4. **Premium Color / White Paper:** $N_{pages} \times 0.002347”$ (0.0596 mm)

Constraint: The application must disable spine text inputs if $N_{pages} < 79$. For books with ≥ 79 pages, spine text must maintain a safety margin of $0.0625”$ (1.6 mm) from the spine folds.[6]

---

3. Visual & Illustrated Engine Specifications

This module handles image-heavy genres (Manga, Comics, Children's Books) where layout preservation is paramount. The primary technical challenge here is the distinction between Fixed Layout (required for these genres) and Reflowable text.

3.1 Manga (Shonen)

Manga presents a unique architectural challenge due to its reading direction. Standard Western publishing software defaults to Left-to-Right (LTR) pagination, which destroys the narrative flow of Manga.

3.1.1 Paperback Specification (RTL)

- **Pagination Inversion:** The application must treat the "first" page of the file as the physical "front" of the book, but the pagination logic must be inverted. In a standard book, odd pages are on the right. In a Right-to-Left (RTL) Manga, **odd pages must appear on the left** and even pages on the right.[1]
- **File Ordering:** KDP’s print machines process files sequentially (Page 1 $\rightarrow$ Last Page).

The application must output the PDF in natural reading order (Story Start $\rightarrow$ Story End), but the visual previewer in the webapp must render the "Spread" (two pages side-by-side) with Page 2 on the Right and Page 3 on the Left to allow the user to visualize the artwork correctly across the spine.

- **Moiré Pattern Mitigation:** Manga often uses "screentones" (patterns of dots) for shading. When these digital dots overlay with the printer's halftone screens, severe visual artifacts (Moiré) can occur. The application should implement a "Descreening" or Gaussian Blur filter (radius 0.5px) on high-frequency halftone images during the PDF rasterization process to smooth out digital dots before printing.

3.1.2 Ebook Specification (Kindle Comic)

- **Metadata:** The EPUB/KPF export must include the primary-writing-mode metadata to force the Kindle device to page backward.
  - Markup: <meta name="primary-writing-mode" content="horizontal-rl"/>.[4]
  - Orientation: <meta name="orientation-lock" content="portrait"/>.
- **Virtual Panels (Panel View):** To support Kindle's "Panel View" (zooming into individual panels), the application requires a panel-detection algorithm.
  - **Algorithm:** Use OpenCV (Python backend) or a client-side library to detect rectangular contours in the Manga pages.
  - **Markup:** Generate a region-magnification JSON map that defines the $(x, y, width, height)$ of each panel. The output HTML must wrap these regions in tags with specific KDP classes to enable the double-tap-to-zoom functionality.[3]

3.2 Comic Books (Western)

While similar to Manga, Western comics generally follow LTR reading and use color.

3.2.1 Color Profile Management

- **Print (CMYK):** Users typically upload RGB PNG/JPGs. The application must perform a colorspace conversion to **CMYK (SWOP v2 or Fogra39)**.
  - *Warning:* Out-of-gamut colors (neon greens, bright cyans) will look dull in print. The application should include a "Soft Proofing" preview mode using a WebAssembly-based color engine (like LittleCMS compiled to Wasm) to simulate the dulling effect of CMYK print on the user's screen, managing expectations before print.[2]
- **Ebook (RGB):** For the digital version, the original RGB assets must be preserved to maximize vibrancy on OLED/LCD screens of tablets. The system must maintain two parallel asset pipelines: one for Print (CMYK/300dpi) and one for Digital (RGB/1920px width).

3.2.2 Guided View Architecture

- **Magnification Logic:** The system must generate magTarget structures for each panel.
  - **Zoom Factor:** Standard panels should be set to **150%** magnification.
  - **Small Text:** Panels with dense dialogue bubbles require **200%** magnification.[4]
  - **Resolution:** To support 200% zoom, the source image must be at least 3840×2400 pixels. The system must validate input resolution and warn users if images are too small for high-quality guided view.[4]

3.3 Children's Picture Books

These books rely heavily on full-bleed illustrations and large text. The primary challenge is accessibility and text legibility on small devices.

3.3.1 Text Pop-Up Technology

Children's books on Kindle cannot be reflowable, but full-page images make text unreadable on a phone. The solution is **Text Pop-Ups**.

- **Implementation:** The application must provide a UI for users to draw "Touch Zones" over the text areas of their uploaded images.
- **HTML Structure:**
  - The background image is set as a fixed-layout page.
  - The text is overlaid as a hidden HTML element (opacity: 0) to allow for search indexing.
  - A separate "Pop-Up Container" is defined in the HTML. When the user taps the active zone, the container (with a solid paper-textured background and large, readable font) becomes visible.[3]
- **Constraint:** The text in the pop-up must be at least **4mm** high on a 7-inch device.[3] The application must enforce a minimum font size (e.g., 30px) in the CSS generation logic.

3.3.2 Bleed & Safe Zone Enforcement

Children's books almost always use "Full Bleed." The application must auto-enable bleed ($0.125”$ addition) and overlay a rigorous "Safe Zone" ($0.25”$ from trim). Because children's books often place critical illustration details near the edge, the UI must explicitly warn users if faces or text bubbles cross into the danger zone where they might be trimmed.

---

4. Non-Fiction Engine Specifications

This module targets text-heavy genres: Biography, Business & Money, and Self-Help. The technical priority shifts from layout preservation to **Reflowable Text** architecture. The output

must be valid EPUB3 (for Ebook) and formatted PDF (for Print).

4.1 Reflowable Text Architecture (Global)

For Ebooks in these genres, Fixed Layout is strongly discouraged. The application must generate **Reflowable EPUBs**.

- **CSS Best Practices:**
  - **Units:** Use em or % for all spacing and font sizes. Never use px or pt, as these do not scale with user settings.[9]
  - **Alignment:** Headings should have explicit text-align (left/center) properties. Body text must *not* have forced alignment (allow the device to justify).
  - **Fonts:** Fonts should generally be left to the user's device defaults. If embedded fonts are used (e.g., for headers), a fallback stack ("Helvetica", sans-serif) is mandatory.[9]

4.2 Biography

Biographies often contain a mix of narrative text and photographic plates.

- **Image Anchoring:** Photos must be anchored to the text flow. The application must insert images using HTML <figure> tags with <figcaption> for accessibility.
- **Footnote Linking:** Biographies require extensive citation. The application must implement a **bidirectional linking algorithm**.
  - *Text:*
  - *Endnote:* ^ Return to text
  - The system must automate the generation of these IDs to prevent collision.[9]

4.3 Business & Money

This genre relies on data visualization: charts, graphs, and complex tables.

- **Table Rendering:**
  - **Small Tables:** Render as pure HTML <table>. The application must validate that the table has fewer than **10 columns** and **100 rows**.
  - **Large Tables:** If a table exceeds these dimensions, the system must automatically rasterize it into an image, as large HTML tables break the Kindle renderer.[10] Ideally, split the table image horizontally if it exceeds the screen height.
- **Charts:** Vector (SVG) charts should be converted to high-resolution Grayscale PNGs for Ebooks to ensure consistent rendering across E-ink devices.

4.4 Self-Help

Self-help books often include interactive elements like checklists, questions, or worksheets.

- **Worksheet Adaptation:**
  - *Print:* Standard lines or boxes.
  - *Ebook:* Input fields are not supported on most E-ink Kindles. The application must

convert worksheets into a "call to action" (e.g., "Take a moment to write in your journal...") or provide a link to a downloadable resource.

- **Sidebar/Callout CSS:** Self-help books use "Pull Quotes" or "Key Takeaway" boxes.
  - *CSS Pattern:* Use a with border, padding (in ems), and page-break-inside: avoid to prevent the box from being split across two pages.[9]
  - *Margins:* Use percentage-based margins (e.g., margin: 0 5%;) to separate the box from the main text flow.

---

5. Activity & Low-Content Engine Specifications

This module requires **Procedural Generation** rather than just formatting. The application must create content algorithmically.

5.1 Coloring Books (Adult vs. Kids)

The technical distinction lies in line weight and complexity.

5.1.1 Adult Coloring Books

- **Algorithmic Parameters:**
  - **Line Weight:** Minimum **0.5 pt** to **0.75 pt**. Intricate details are expected.
  - **Resolution:** Must be **600 DPI**. Thin lines pixelate visibly at 300 DPI.
- **Paper Science:** Adult coloring often uses markers. Standard KDP paper (55#) bleeds through.
  - **Blank Page Logic:** The application must force a **single-sided layout**. Every right-hand page (recto) contains art; every left-hand page (verso) must be blank (or contain a faint pattern) to buffer bleed-through.

5.1.2 Kids Coloring Books

- **Algorithmic Parameters:**
  - **Line Weight:** Minimum **2 pt**. Thicker lines help motor control.
  - **Subject Matter:** Large closed shapes.
- **Bleed:** Full bleed is standard. The system must ensure main subjects are centered and not cut off.

5.1.3 KDP Coloring Page (Single)

- **Format:** This likely refers to digital downloads or "Printable" pages. The system should export high-res PDFs or vector SVGs, not EPUBs.

5.2 Activity Books (Sudoku & Word Search)

The application requires a backend logic engine (Python/Node) to generate these puzzles.

5.2.1 Sudoku Generation Algorithm

- **Backtracking Solver:**
    1. Initialize a 9×9 grid.
    2. Fill the diagonal 3×3 subgrids independently (valid, as they share no rows/cols).
    3. Use a recursive backtracking algorithm to fill the rest of the grid.
    4. **Difficulty Control:** Remove N cells based on difficulty.
        - Easy: Remove ~30.
        - Hard: Remove ~50.
    5. **Validation:** Run the solver on the generated puzzle. If >1 solution exists, the puzzle is invalid (too ambiguous). Backtrack and replace a number.[11]

5.2.2 Word Search Algorithm

- **Grid Placement:**
    1. Create an empty $N \times N$ grid.
    2. Sort word list by length (Descending).
    3. Place words one by one:
        - Pick random coordinate (x, y) and direction (d).
        - Check bounds and collisions (overlap is allowed if letters match).
        - If placement fails after K attempts, restart grid.[13]
    4. **Fill:** Populate empty cells with random letters. *Refinement:* Use letter frequency analysis of the target language (e.g., more 'E's and 'A's) to make the noise look natural.

5.3 Journals, Planners, and Log Books

These are "Low-Content" books.

5.3.1 No-ISBN Workflow

- **Barcode Suppression:** If the user selects "No ISBN" (allowed for low-content), the system must **not** generate a barcode placeholder on the cover. KDP will place its own tracking code in the bottom right.
    - **Constraint:** The application must render a "Forbidden Zone" overlay in the bottom-right $2" \times 1.2"$ area of the back cover. Users must be prevented from placing art here.[15]
- **Limitations:** The system must warn users that "Look Inside" is disabled for No-ISBN books.[15]

5.3.2 Planner Algorithms

- **Calendar Logic:** The app needs a Date engine.
  - *Input:* Year, Start Month.
  - *Logic:* Handle leap years. Calculate correct start day of the week.
  - *Overflow:* Handling months that span 6 weeks (e.g., 30/31 days starting on Saturday). The layout engine must either split the bottom row cells or compress the row height.[16]
- **Optimization:** A 365-day planner can create massive PDF files if every page is a unique vector set.
  - *Technique:* Use PDF **XObjects** (Form XObjects). Define the page lines/grids *once* as a resource, and reference it on 365 pages. This keeps file size < 650MB.

---

6. Technical Implementation: The PDF Generation Pipeline

The core engine requires a robust stack to handle PDF/X-1a compliance.

6.1 Libraries and Tools

- **Primary Generation (Node.js):** pdf-lib or react-pdf for assembling structure.
- **Vector/Chart Generation:** D3.js rendered to SVG, then converted to PDF vectors.
- **Post-Processing (Ghostscript):** Required for PDF/X-1a compliance (flattening and color conversion).

6.2 PDF/X-1a Compliance Script

KDP prefers PDF/X-1a to ensure fonts are embedded and transparency is flattened. The system should execute a Ghostscript command on the generated PDF:

Bash

```
gs -dPDFA -dBATCH -dNOPAUSE -dUseCIEColor \
-sProcessColorModel=DeviceCMYK \
-sDEVICE=pdfwrite \
-sOutputFile=output_x1a.pdf \
-dOverrideICC=true \
-sOutputICCProfile=USWebCoatedSWOP.icc \
input_draft.pdf
```

Rationale: This ensures all RGB images are converted to CMYK using the specified profile and all transparency is flattened into the raster layer, preventing printing errors.[17]

---

7. The Antigravity Master Prompt

The following section provides the exact instructions to initialize the Antigravity IDE for this project.

7.1 Master Directive

Antigravity Mission Control: KDP Web Application Architecture

Role: You are the Chief Systems Architect and Product Manager for a high-performance web application designed to generate Amazon KDP-compliant manuscripts and covers.

Objective: Architect and implement a multi-agent modular system that allows users to create Books for the following genres: Manga, Comics, Children's Books, Coloring Books, Activity Books (Sudoku/Word Search), and Low-Content Books.

Global Constraints:

1. **Strict Separation of Concerns:** Frontend (React/Next.js) must be decoupled from the PDF Generation Engine (Node.js/Python Microservices).
2. **KDP Compliance:** All output must strictly adhere to KDP print physics (Bleed, Trim, Gutter per page count, Spine width formulas).
3. **PDF/X-1a Standard:** Final outputs must be flattened, fonts embedded, and CMYK/Grayscale converted.

Phase 1: Agent Orchestration & Rules Setup

Action: Initialize the .agent/rules/ directory with the following immutable laws.

Rule 1: The Print Physics Law (.agent/rules/kdp_physics.md)

"All agents involved in layout calculation must reference the 'KDP Margin Table'.

- IF page_count between 24-150 THEN gutter = 0.375in
- IF page_count between 151-300 THEN gutter = 0.5in
- IF page_count between 301-500 THEN gutter = 0.625in
- IF page_count between 501-700 THEN gutter = 0.75in

- IF page_count between 701-828 THEN gutter = 0.875in
- Bleed is ALWAYS +0.125in on Top, Bottom, and Outside Edge. NEVER on Gutter."

Rule 2: The Spine Calculus Law (.agent/rules/spine_calc.md)

"Spine width is strictly calculated as:

- White Paper (B&W/Color): PageCount * 0.002252 inches
- Cream Paper (B&W): PageCount * 0.0025 inches
- Premium Color (White): PageCount * 0.002347 inches
  Spine text is FORBIDDEN if PageCount < 79."

Rule 3: The Manga Pagination Law (.agent/rules/manga_rtl.md)

"For Genre=Manga OR ReadingDirection=RTL:

- Odd Page Numbers MUST appear on the LEFT physical page.
- Even Page Numbers MUST appear on the RIGHT physical page.
- Metadata 'primary-writing-mode' MUST be set to 'horizontal-rl'."

Phase 2: Implementation Plan (Task Assignment)

Agent 1: The Mathematician (Core Logic)

- **Task:** Create a shared utility library (@utils/kdp-calculator) that accepts TrimSize, PageCount, PaperType, and Format (Hardcover/Paperback).
- **Output:** Return a JSON object containing canvasDimensions, safeZoneCoordinates, gutterWidth, and spineWidth.
- **Validation:** Write unit tests covering boundary cases (e.g., exactly 150 pages vs 151 pages).

Agent 2: The Procedural Engine (Activity & Low Content)

- **Task:** Implement generation algorithms.
  - *Sudoku:* Backtracking algorithm with 3 difficulty tiers (cell removal counts).
  - *Word Search:* Grid placement with collision detection and backtracking.
  - *Planner:* Date-aware calendar grid generator handling leap years and 6-week months.
- **Constraint:** Ensure generated SVGs/Canvases respect the safeZoneCoordinates from Agent 1.

Agent 3: The Compositor (PDF Generation Service)

- **Task:** Build a Node.js microservice using pdf-lib and ghostscript4js.
- **Pipeline:**
  1. Assemble individual pages from Agent 2 or User Uploads.
  2. Apply 'Shift Logic' for Gutter (Move Odd pages Right, Even pages Left based on

Gutter width).
3. Flatten transparency.
4. Convert to PDF/X-1a using Ghostscript.
5. Validate against KDP max file size (650MB).

Agent 4: The Frontend Architect (UI/UX)

- **Task:** Build a Next.js dashboard.
- **Features:**
  - Real-time "Spread View" (showing Left/Right pages together).
  - "Safe Zone Overlay" (Visual toggle showing red overlay for trim/bleed).
  - "Pre-flight Check" component: Scans project for low-res images (<300 DPI) and text in bleed zones.

Phase 3: Execution Command

Execute the following workflow:

1. Scaffold the Next.js project structure with strict TypeScript interfaces for BookProject.
2. Implement the kdp-calculator utility first.
3. Build the Manga and Sudoku modules in parallel using the defined Agent Rules.
4. Report back with the Implementation Plan Artifact for review before writing the PDF generation code.

8. Conclusion

This specification provides the necessary architectural blueprint to build an automated KDP publishing system. By creating a rigid separation between the creative content layers and the physical compliance layers, the system ensures that every generated book—whether a complex Manga or a simple Journal—will pass Amazon's pre-flight checks. The integration of algorithmic generation for activity books and precise metadata controls for ebooks creates a scalable platform capable of handling the entire spectrum of self-publishing needs. The provided Master Prompt is ready for immediate deployment into the Antigravity IDE to begin the scaffolding process.

Works cited

1. Paperback Submission Guidelines - Kindle Direct Publishing, accessed February 7, 2026, https://kdp.amazon.com/help/topic/G201857950
2. ICC colour profiles - KDP Community, accessed February 7, 2026, https://www.kdpcommunity.com/s/question/0D58V00007ElqJ4SAL/icc-colour-profiles?language=en_US
3. Creating Fixed-Layout Books Without Pop-Ups - Kindle Direct Publishing, accessed February 7, 2026, https://kdp.amazon.com/en_US/help/topic/GEGU359TQLKDJZZH

4. Creating Fixed-Layout Books with Image Pop-Ups or Virtual Panels, accessed February 7, 2026, https://kdp.amazon.com/en_US/help/topic/G9GSTY4LTRT39D4Z
5. Set Trim Size, Bleed, and Margins - Kindle Direct Publishing, accessed February 7, 2026, https://kdp.amazon.com/help/topic/GVBQ3CMEQW3W2VL6
6. Create a Paperback Cover - Kindle Direct Publishing, accessed February 7, 2026, https://kdp.amazon.com/help/topic/G201953020
7. Fix Paperback and Hardcover Formatting Issues - Kindle Direct Publishing, accessed February 7, 2026, https://kdp.amazon.com/help/topic/G201834260
8. Comparing Formats - Kindle Direct Publishing, accessed February 7, 2026, https://kdp.amazon.com/en_US/help/topic/G42HENP2VHSN8VW8
9. Text Guidelines - Reflowable - Kindle Direct Publishing, accessed February 7, 2026, https://kdp.amazon.com/en_US/help/topic/GH4DRT75GWWAGBTU
10. Image Guidelines - Reflowable, accessed February 7, 2026, https://kdp.amazon.com/help/topic/G75V4YX5X8GRGXWV
11. Generating & Solving Sudoku in JS & Ruby with Backtracking - DEV Community, accessed February 7, 2026, https://dev.to/dsasse07/generating-solving-sudoku-in-js-ruby-with-backtracking-4hm
12. Program for Sudoku Generator - GeeksforGeeks, accessed February 7, 2026, https://www.geeksforgeeks.org/dsa/program-sudoku-generator/
13. 79. Word Search - In-Depth Explanation - AlgoMonster, accessed February 7, 2026, https://algo.monster/liteproblems/79
14. Generating Word Search Puzzles - Buckblog, accessed February 7, 2026, https://weblog.jamisbuck.org/2015/9/26/generating-word-search-puzzles.html
15. Low-Content Books - Kindle Direct Publishing - Amazon.com, accessed February 7, 2026, https://kdp.amazon.com/help/topic/GGE5T76TWKA85DJM
16. Create a calendar - The Modern JavaScript Tutorial, accessed February 7, 2026, https://javascript.info/task/calendar-table
17. How to use ghostscript to convert PDF to PDF/A or PDF/X? - Stack Overflow, accessed February 7, 2026, https://stackoverflow.com/questions/1659147/how-to-use-ghostscript-to-convert-pdf-to-pdf-a-or-pdf-x
18. Converting RGB PDF in CMYK with plain black using ghostscripts sOutputICCProfile, accessed February 7, 2026, https://stackoverflow.com/questions/71036645/converting-rgb-pdf-in-cmyk-with-plain-black-using-ghostscripts-soutputiccprofile