Engineering the Modern Legal Infrastructure: A Technical and Procedural Analysis of the Python Legal Toolkit Expansion

1. Executive Summary

The convergence of rigorous legal procedure and advanced software engineering—a discipline increasingly recognized as "Legal Engineering"—represents a transformative shift in the operational dynamics of modern litigation. The "Legal Toolkit" project, currently a functional Command Line Interface (CLI) utility, stands at the precipice of this evolution. To transition from a utility script into a portfolio-grade application capable of impressing senior stakeholders and legal technology recruiters, the system must undergo a radical architectural expansion. This report provides an exhaustive technical roadmap for that expansion, focusing on the seamless integration of external compliance data, high-performance document engineering, accessible user experience (UX) design, and the implementation of "Algorithmic Law."

The contemporary legal landscape is unforgiving of procedural error. The Civil Procedure Rules (CPR), specifically Part 2.8 regarding time calculations and Part 6 regarding service, create a complex logic gate where a single miscalculation regarding a "business day" or a "4:30 PM cutoff" can lead to professional negligence claims or the striking out of a case. Consequently, software designed for this domain cannot rely on generic datetime libraries or hard-coded assumptions. It requires a dynamic, authoritative source of truth. This analysis mandates the integration of the **Gov.uk Bank Holiday API** to ensure that deadline algorithms possess strict adherence to jurisdiction-specific holidays in England, Wales, Scotland, and Northern Ireland.

Furthermore, the digitization of the courts, governed by Practice Direction 5B (PD 5B), demands sophisticated electronic bundle preparation. The manual collation of PDF documents, generation of indices, and application of Bates numbering are high-volume, low-value tasks ripe for automation. This report conducts a comparative analysis of Python's PDF ecosystem—specifically **pypdf**, **PyMuPDF**, and **pdfplumber**—to recommend a high-performance engine capable of manipulating thousands of pages without compromising file integrity or processing speed.

However, robust backend logic is insufficient if the tool remains inaccessible to the non-technical fee earner. The transition from CLI to GUI is critical. This report evaluates the architectural trade-offs between **Streamlit**, **Textual**, and **PyQt**, ultimately recommending a

web-based approach that leverages WebAssembly (Wasm) for secure, local deployment without the overhead of traditional software installation. Finally, to distinguish this toolkit in a crowded market, the report proposes the integration of a **Local Large Language Model (LLM)** for privacy-first document summarization, creating a "Killer Feature" that addresses the industry's twin obsessions: Generative AI and Client Confidentiality.

---

2. The Compliance Architecture: Temporal Logic and API Integration

2.1 The Legal Necessity of Dynamic Temporal Reference

In the realm of civil litigation, "time" is a legal construct defined by the Civil Procedure Rules (CPR), not merely a chronological progression. CPR Part 2.8 is the governing provision, explicitly defining a "period of time" for procedural steps. Crucially, CPR 2.8(2) excludes Saturdays, Sundays, and Bank Holidays from the computation of time for any period of 5 days or less, and defines "business day" for all service calculations.[1]

The fragility of existing legal calculators often stems from "static temporality"—the practice of hard-coding holiday dates. While weekends are algorithmically constant, Bank Holidays in the United Kingdom are subject to Royal Proclamation and political change. They are not fixed constants; they can be moved (e.g., a holiday falling on a weekend being substituted to the following Monday) or created ad hoc (e.g., for State Funerals or Jubilees). A calculator that fails to account for a newly announced holiday poses a severe risk. If a limitation period expires on a day the software erroneously identifies as a business day, the user may miss the true deadline, potentially leading to a professional negligence claim. Therefore, strict compliance requires the toolkit to consume a dynamic, government-maintained "Source of Truth."

2.2 Architectural Analysis of the Gov.uk Bank Holiday API

The research identifies the UK Government Digital Service (GDS) Bank Holiday API as the requisite authority. Located at https://www.gov.uk/bank-holidays.json, this endpoint provides the official, up-to-date schedule of public holidays.[3]

2.2.1 Data Structure and Jurisdictional Variance

The API returns a JSON object segmented by "division." This segmentation is architecturally critical because legal jurisdiction within the UK is not uniform. The High Court in London operates under the jurisdiction of England and Wales, while the Court of Session sits in Scotland. These jurisdictions observe different holidays (e.g., St Andrew’s Day in Scotland, St Patrick’s Day in Northern Ireland). A "Legal Toolkit" that treats the UK as a monolith is legally

defective.

The schema provided by the API [4] categorizes data into three primary keys:

1. england-and-wales
2. scotland
3. northern-ireland

Within each division, the events array contains objects with a date (ISO 8601 format: YYYY-MM-DD), a title, and notes. The bunting boolean, while present in the schema [4], indicates whether flags are flown on government buildings and is procedurally irrelevant for deadline calculations, though it could be parsed for UI decorations.

The technical implication here is that the toolkit's Calculator class must be instantiated with a specific jurisdiction context. It cannot simply import "UK Holidays"; it must import "CPR-Relevant Holidays" based on the venue of the litigation.

2.2.2 Reliability, Latency, and the "Offline" Requirement

While the Gov.uk API is open and currently requires no authentication [5], relying on a synchronous HTTP request for every calculation is poor software engineering practice. Lawyers often work in environments with intermittent connectivity—courtrooms with shielded walls, trains, or secure "air-gapped" discovery rooms. A tool that hangs or crashes because it cannot reach gov.uk is useless.

Furthermore, GDS imposes rate limits.[1] While the exact threshold is high, a "Smart Bundle" feature that checks dates for thousands of documents in a loop could trigger a temporary IP ban, rendering the tool inoperable.

2.3 Technical Recommendation: The "Resilient Provider" Pattern

To satisfy the requirements of "Strict CPR Compliance" and "Scalability" outlined in the expansion plan, the toolkit should implement a hybrid **Cached-First, Lazy-Update Strategy**. This ensures the tool functions immediately upon installation (even offline) while maintaining accuracy over time.

2.3.1 Implementation Strategy

The architecture should separate the holiday logic into a dedicated BankHolidayProvider class. This class is responsible for:

1. **Initialization:** Loading a local JSON file (bank_holidays.json) shipped with the Python package. This guarantees immediate availability.
2. **Validation:** Checking the timestamp of the local cache. If the cache is older than a user-configurable threshold (e.g., 30 days), it attempts to fetch fresh data from the API.
3. **Fail-Safe Fetching:** If the API request fails (due to timeout or connection error), the

system degrades gracefully, logging a warning to the user ("Warning: Using cached holiday data from") but proceeding with the calculation using the local file.

4. **Jurisdictional Filtering:** Parsing the JSON to return a set of datetime.date objects specific to the requested jurisdiction, enabling $O(1)$ lookup complexity during date calculations.

2.3.2 Proposed Integration Code

The following implementation demonstrates this robust architecture, utilizing the requests library for connectivity and json for persistence.

Python

```
import requests
import json
import os
from datetime import date, timedelta, datetime

class BankHolidayProvider:
    API_URL = "https://www.gov.uk/bank-holidays.json"
    CACHE_FILE = "bank_holidays_cache.json"

    def __init__(self, jurisdiction: str = 'england-and-wales'):
        self.jurisdiction = jurisdiction
        self.holidays = self._load_data()

    def _load_data(self) -> set[date]:
        """
        Loads holiday data. Prioritizes fresh API data if available
        and cache is stale; falls back to cache on failure.
        """
        data = None

        # Step 1: Try Network Refresh if cache is missing or stale
        try:
            response = requests.get(self.API_URL, timeout=3.0)
            response.raise_for_status()
            data = response.json()
            # Update Cache
            with open(self.CACHE_FILE, 'w') as f:
                json.dump(data, f)
```

```
        except requests.RequestException:
            # Step 2: Fallback to Local Cache
            if os.path.exists(self.CACHE_FILE):
                with open(self.CACHE_FILE, 'r') as f:
                    data = json.load(f)
            else:
                # Critical Failure: No network and no cache
                # In a real package, this would fallback to a shipped package resource
                return set()

        if not data or self.jurisdiction not in data:
            return set()

        # Step 3: Parse specific jurisdiction
        events = data[self.jurisdiction].get('events',)
        return {datetime.strptime(e['date'], '%Y-0%m-%d').date() for e in events}

    def is_holiday(self, target_date: date) -> bool:
        return target_date in self.holidays

# Integration with CPR 2.8 Logic
def next_business_day(start_date: date, provider: BankHolidayProvider) -> date:
    current = start_date
    while True:
        current += timedelta(days=1)
        # CPR 2.8: Exclude Sat (5), Sun (6), and Bank Holidays
        if current.weekday() < 5 and not provider.is_holiday(current):
            return current
```

This code snippet illustrates the "Resilient Provider" pattern. The next_business_day function utilizes the provider to ensure strict compliance with CPR 2.8, while the provider itself manages the complexity of network reliability.[2] This approach satisfies the high-priority requirement for "UK Bank Holiday API integration" while adhering to professional software standards.

---

3. Document Engineering: The Architecture of the Electronic Bundle

3.1 The Digital Mandate: Practice Direction 5B

The digitization of the UK courts is governed by Practice Direction 5B (PD 5B), which sets out

the technical standards for electronic bundles. These are not merely suggestions; non-compliance can result in bundles being rejected by the court filing system (CE-File) or judicial criticism during hearings.

The "Smart Bundle Indexer" functionality in the current toolkit scans directories, but the expansion plan demands active manipulation. Specifically, the tool must:

1. **Merge:** Combine heterogenous PDFs into a single litigation master file.
2. **Bates Stamp:** Apply sequential page numbering (e.g., "A-0001", "A-0002") to the footer of every page. This is the primary method of referencing evidence in court.
3. **Index:** Generate a clickable, nested Table of Contents (ToC) that mirrors the directory structure.

To implement this, the research evaluates three primary Python libraries: **pypdf**, **PyMuPDF (fitz)**, and **pdfplumber**.

3.2 Comparative Analysis of PDF Engines

3.2.1 pypdf (formerly PyPDF2): The Pure Python Standard

pypdf is a widely used, pure-Python library.[7] Its primary advantage is its license (BSD) and its lack of binary dependencies, making it universally installable.

- **Merging:** It excels at merging operations (PdfWriter.append), handling page rotation and basic metadata preservation effectively.
- **Bates Stamping:** This is where pypdf struggles. It treats PDF pages as immutable objects. To add text (like a Bates number), the developer cannot simply "write" onto the page. Instead, they must create a new, temporary PDF containing *only* the text (using a library like reportlab or canvas), and then "stamp" this temporary page over the original page like a watermark.[8] This process involves significant overhead: for a 5,000-page bundle, the system must generate 5,000 temporary PDF pages and perform 5,000 merge operations.
- **Indexing:** It supports add_outline_item, which allows for the programmatic creation of bookmarks.[10]

3.2.2 pdfplumber: The Analyst's Tool

pdfplumber is built on top of pdfminer.six.[7] Its architecture is optimized for *extraction* rather than *creation*. It is exceptionally robust at extracting tabular data and analyzing layout, making it ideal for the "Storage Heatmap" feature (identifying which pages are largest). However, it lacks the native capability to write or modify PDF files. It cannot merge documents or apply stamps, rendering it unsuitable as the core engine for bundle generation.[12]

3.2.3 PyMuPDF (fitz): The Performance Engine

PyMuPDF is a Python binding for the MuPDF engine, which is written in C.

- **Performance:** It is significantly faster than pypdf due to its C backend, a critical factor when processing gigabytes of legal evidence.[13]
- **Bates Stamping:** Unlike pypdf, PyMuPDF allows for direct modification of the PDF canvas. The page.insert_text() method allows the developer to write text directly to specific coordinates (e.g., bottom-right corner) with control over font, size, and color.[14] This avoids the "watermark" overhead and is vastly more efficient.
- **Indexing:** It offers a streamlined API for manipulating the Table of Contents via doc.set_toc(), which accepts a simple list of lists structure ([level, title, page_number]), simplifying the recursive logic required to turn a folder structure into a PDF outline.[16]

3.3 Technical Recommendation: A Hybrid Approach

For a portfolio piece intended to demonstrate "High Technical Competence," **PyMuPDF** is the superior choice for the heavy lifting of bundle generation. Its speed and direct manipulation capabilities align with the requirement for "Scalability" and "Performance Optimization."

However, pypdf remains valuable for its permissive license if the toolkit is intended to be fully open-source without the copyleft restrictions of PyMuPDF's AGPL license. Assuming the user prioritizes functionality and "impressiveness" to a recruiter, PyMuPDF is the recommendation.

3.3.1 Implementation: High-Performance Bates Stamping

The implementation of Bates stamping using PyMuPDF demonstrates a sophisticated understanding of coordinate systems and file manipulation.

Python

```
import fitz  # PyMuPDF

def apply_bates_stamping(input_path: str, output_path: str, prefix: str = "ABC"):
    doc = fitz.open(input_path)

    for i, page in enumerate(doc):
        # Determine Page Dimensions for relative positioning
        rect = page.rect

        # Calculate coordinates: Bottom Right, with 20pt padding
        # Point(x, y) where (0,0) is top-left
        insert_point = fitz.Point(rect.width - 100, rect.height - 20)

        bates_label = f"{prefix}-{i+1:06d}"
```

```
        # Insert text directly onto the existing page canvas
        page.insert_text(
            insert_point,
            bates_label,
            fontsize=12,
            fontname="helv",  # Standard Helvetica
            color=(0, 0, 0)   # Black
        )

    doc.save(output_path)
```

Implication: This code is concise and efficient. In a portfolio context, this demonstrates to a recruiter that the candidate understands not just *how* to write Python, but how to choose the right tool for high-performance file processing.[14]

3.3.2 Implementation: The Smart Indexer

To generate the INDEX.txt and the PDF bookmarks, the toolkit must traverse the directory structure. PyMuPDF's set_toc method is perfectly suited for this.

Python

```
def generate_toc(doc, file_structure):
    # toc structure: [level, title, page_number]
    toc =
    current_page = 1

    for folder, files in file_structure.items():
        # Add Folder as Level 1
        toc.append([1, folder, current_page])

        for file in files:
            # Add File as Level 2
            toc.append([2, file.name, current_page])
            current_page += file.page_count

    doc.set_toc(toc)
```

This logic maps directly to the user's requirement for a "Smart Bundle Indexer," automating

the creation of the navigable structure required by PD 5B.

---

4. User Experience Architecture: The Legal Dashboard

4.1 The UX Dilemma: CLI vs. The Fee Earner

While the Command Line Interface (CLI) is efficient for developers, it presents a significant barrier to entry for the average legal professional. Fee earners (lawyers and paralegals) are accustomed to rich, graphical interfaces (Word, Outlook). A tool that requires terminal interaction is unlikely to be adopted, regardless of its backend utility. To transform the "Legal Toolkit" into a showcase portfolio piece, the frontend must be intuitive, visual, and easy to deploy.

The research evaluates three frameworks for this purpose: **Streamlit**, **Textual**, and **PyQt**.

4.2 Framework Comparative Analysis

4.2.1 PyQt / PySide: The Native Desktop Standard

PyQt (wrapping the Qt framework) is the industry standard for building robust, native desktop applications.[17]

- **Pros:** It provides granular control over every pixel and window behavior. It feels like "real" software.
- **Cons:** The learning curve is steep. Building a simple dashboard requires significant boilerplate code. It follows an event-driven programming model (signals and slots) which is more complex than the procedural scripting lawyers might be used to.[18] Styling relies on QSS (Qt Style Sheets), which can look dated without significant effort.
- **Verdict:** Powerful, but potentially "over-engineering" for a portfolio piece where visual impact and speed of development are key.

4.2.2 Textual: The Terminal Renaissance

Textual is a TUI (Text User Interface) framework that brings mouse support, CSS styling, and layouts to the terminal.[19]

- **Pros:** It has a distinct "hacker-chic" aesthetic that appeals strongly to technical interviewers. It runs lightly and requires no browser.
- **Cons:** It is still a terminal application. For a non-technical lawyer, a black screen with text—even clickable text—can be intimidating. It lacks native support for complex data visualizations like Gantt charts or heatmaps, which are core requirements of the expansion plan.[20]
- **Verdict:** Excellent for a developer tool, but suboptimal for a general-purpose "Legal Dashboard."

4.2.3 Streamlit: The Data App Revolution

Streamlit is a framework designed to turn data scripts into shareable web apps in minutes.[21]

- **Pros:** It allows for rapid prototyping. The requested visualizations (Timeline View, Storage Heatmap) can be implemented in single lines of code using libraries like **Plotly**, which Streamlit supports natively.[22] It offers a modern, clean aesthetic out of the box.[23]
- **Cons:** It is fundamentally a web server. Distributing it as a standalone executable (.exe) is non-trivial because it typically requires a Python environment and a browser to run.
- **Verdict:** The optimal choice for "Impressiveness." Its ability to visualize data aligns perfectly with the goal of showing "Legal Utility."

4.3 Technical Recommendation: Streamlit with WebAssembly Deployment

The report recommends **Streamlit** as the frontend engine. It satisfies the "Visualizations" requirement most effectively.

4.3.1 Visualizing Legal Data

The user requested a "Timeline View" (Gantt chart) and a "Storage Heatmap." Streamlit, combined with Plotly Express, can render these interactively.

The Timeline View (Gantt Chart):
This visualization is crucial for illustrating the "Deemed Service" gap—the dangerous period between sending a document and the court recognizing it as served.

Python

```
import streamlit as st
import plotly.express as px
import pandas as pd

def render_timeline(service_date, deemed_date, deadline):
    data =
    df = pd.DataFrame(data)
    fig = px.timeline(df, x_start="Start", x_end="Finish", y="Task", color="Type")
    st.plotly_chart(fig)
```

This code produces an interactive chart where users can hover to see exact dates, significantly enhancing the "User-Centric Design".[24]

4.3.2 The Deployment Challenge: From Script to Executable

The user explicitly requested an .exe distribution strategy using PyInstaller. However, packaging Streamlit with PyInstaller is notoriously difficult because Streamlit spawns a server process that conflicts with the frozen environment of a PyInstaller executable.[25]

Innovative Solution: stlite (Serverless Streamlit)
To demonstrate "Innovation" and "High Technical Competence," the report recommends using stlite. This is a port of Streamlit to WebAssembly (Wasm). It allows the Python runtime and the Streamlit app to run entirely inside the browser, without a backend server.26

- **Mechanism:** The application is packaged into a single HTML file or a lightweight Electron wrapper (stlite-desktop).
- **Advantage:** It eliminates the dependency hell of Python installations on client machines. The "executable" is essentially a dedicated browser window running the Python code locally.[27]
- **PyInstaller Fallback:** If stlite is deemed too experimental, the streamlit-desktop-app wrapper (based on Electron) provides a bridge, or one can fall back to packaging a standard script that launches the system browser, though this feels less like a "native" app.[29]

---

5. Algorithmic Law: The Logic of Deemed Service

5.1 The "Black Box" of Legal Time

Implementing "Deemed Service" rules (CPR 6.14 and CPR 6.26) represents the most significant algorithmic challenge in the toolkit. It moves the project from a simple calculator to a system that encodes legal reasoning.

The complexity arises from the interaction between **Actual Service** (when the step was taken) and **Deemed Service** (when the court treats the document as served). These concepts are distinct and governed by different rules.

5.2 Deconstructing the Rules

1. **CPR 7.5 (The "Relevant Step"):** To stop the limitation clock, the claimant must complete the "relevant step" (e.g., posting the form, sending the email) before 12:00 midnight on the calendar day 4 months after the claim form was issued.[30] This rule focuses on the *act* of dispatch.
2. **CPR 6.26 (The "Electronic Cutoff"):** This rule governs *when* that step is considered effective for the purpose of the deemed service calculation.
    - **The 4:30 PM Rule:** If a document is sent by email **before 4:30 PM** on a business day, it is treated as sent that day.
    - If sent **after 4:30 PM**, or on a non-business day (weekend/bank holiday), it is treated

as sent on the **next business day**.[2]

3. **CPR 6.14 (The "Deeming" Provision):** Once the "effective day" of service is established (via CPR 6.26), the claim form is deemed served on the **second business day** after that effective day.[33]

5.3 Algorithmic Implementation: The Logic Pipeline

The toolkit must implement a multi-stage logic pipeline to handle this. A simple "add 2 days" calculation is legally dangerous.

Scenario:

- **Action:** Email sent containing Claim Form.
- **Timestamp:** Thursday, 28 March 2024 at 17:00 (5:00 PM).
- **Context:** Friday, 29 March is Good Friday (Bank Holiday). Monday, 1 April is Easter Monday.

The Logic Flow:

1. **Stage 1: Normalize "Sent Date" (CPR 6.26)**
   - Input: Thursday 17:00.
   - Check 1: Is it a business day? Yes.
   - Check 2: Is it before 16:30? No.
   - Rule: Roll over to the *next business day*.
   - Calculation: Next day is Friday (Bank Holiday). Next is Saturday (Weekend). Next is Sunday (Weekend). Next is Monday (Bank Holiday).
   - **Effective Step Date:** Tuesday, 2 April 2024.
2. **Stage 2: Calculate Deemed Service (CPR 6.14)**
   - Rule: Deemed Served = Effective Date + 2 Business Days.
   - Start: Tuesday, 2 April.
   - Business Day +1: Wednesday, 3 April.
   - Business Day +2: Thursday, 4 April.
   - **Result:** Deemed Date of Service is **Thursday, 4 April 2024**.

Implication: A naive calculator adding 2 days to Thursday 28th might return Saturday 30th or Monday 1st. Both are incorrect. The "Power User" feature correctly identifies the 7-day gap caused by the Easter weekend and the 4:30 PM rule.

Code Logic Structure:

Python

```
def calculate_deemed_service(sent_datetime: datetime, provider: BankHolidayProvider) -> date:
    # 1. Check CPR 6.26 (4:30 PM Rule)
    cutoff_time = datetime.time(16, 30)
    effective_day = sent_datetime.date()

    # If sent on non-business day OR after 4:30pm on business day
    is_business = is_business_day(effective_day, provider)
    is_late = sent_datetime.time() > cutoff_time

    if not is_business or is_late:
        effective_day = next_business_day(effective_day, provider)

    # 2. Check CPR 6.14 (Deemed Service = +2 Business Days)
    # Note: We add 2 business days to the EFFECTIVE day
    deemed_date = add_business_days(effective_day, 2, provider)

    return deemed_date
```

This algorithm demonstrates a nuanced understanding of "Algorithmic Law," satisfying the "Innovation" requirement of the project summary.[35]

---

6. The "Killer Feature": Privacy-First AI Summarization

6.1 Moving Beyond Calculation

To truly capture the attention of a Legal Tech recruiter, the project must address the defining technology of the era: **Generative AI**. However, simply connecting to the OpenAI API is insufficient. Law firms are notoriously risk-averse regarding client data confidentiality (SRA Code of Conduct). Uploading a client's confidential bundle to a public cloud model is often a non-starter.

The proposed "Killer Feature" is **Local RAG (Retrieval-Augmented Generation)**.

6.2 The Proposal: Local LLM Integration

The toolkit should integrate a mechanism to run a Large Language Model (LLM) *locally* on the user's machine, ensuring no data ever leaves the secure environment.

- **Technology Stack:**
  - **Ollama:** A framework for running open-source models (like Llama 3 or Mistral) locally.[37]
  - **LangChain:** A Python framework to orchestrate the interaction between the PDF text and the LLM.[38]

- **Text Extraction:** Utilizing the toolkit's existing PyMuPDF or pdfplumber engine to extract text from the "Smart Bundle."

6.3 Workflow and Value Proposition

The feature works as follows:

1. **Ingest:** The user selects a bundle via the Streamlit interface.
2. **Vectorize:** The toolkit extracts text, chunks it, and creates a local vector embedding (using a lightweight model like all-MiniLM-L6-v2).
3. **Query:** The user asks, "Summarize the key events in the Particulars of Claim."
4. **Generate:** The local LLM retrieves relevant chunks and generates a summary.

Why this is a "Killer Feature":

- **Privacy:** It solves the GDPR/Confidentiality blocker.[40]
- **Cost:** It requires no API credits (unlike GPT-4).
- **Competence:** It demonstrates knowledge of cutting-edge AI architecture (RAG, Vector Stores, Quantization) applied specifically to a legal constraint. This elevates the project from a "script" to an "intelligent platform".[41]

---

7. Conclusion

The transformation of the "Legal Toolkit" from a CLI script to a comprehensive legal platform requires a disciplined approach to both software engineering and legal procedure. This report has outlined a roadmap that ensures strict compliance through the **Gov.uk API**, achieves high-performance document handling via **PyMuPDF**, and delivers a modern user experience through **Streamlit** and **WebAssembly**.

By rigorously implementing the "Deemed Service" logic, the tool proves its worth as a piece of "Algorithmic Law," capable of navigating complex procedural traps that human lawyers often miss. Finally, the integration of **Local AI Summarization** positions the project at the forefront of legal technology, offering a privacy-preserving solution to the industry's demand for generative intelligence. This expansion plan satisfies every criterion for a "high technical competence" portfolio piece, bridging the gap between code and court.

Works cited

1. Reuse GOV.UK content, accessed December 21, 2025, https://www.gov.uk/help/reuse-govuk-content
2. 6.26. - The Civil Procedure Rules 1998, accessed December 21, 2025, https://www.legislation.gov.uk/uksi/1998/3132/rule/6.26
3. Bank Holidays - API Catalogue, accessed December 21, 2025, https://www.api.gov.uk/gds/bank-holidays/
4. UK Bank Holidays (Independent Publisher) - Connectors - Microsoft Learn,

accessed December 21, 2025, https://learn.microsoft.com/en-us/connectors/ukbankholidays/
5. Connecting to the UK Bank Holiday API - Wise Owl Training, accessed December 21, 2025, https://www.wiseowl.co.uk/power-bi/blogs/power-bi-desktop/power-bi-dates/bank_holiday_api/
6. govuk_bank_holidays - Rust - Docs.rs, accessed December 21, 2025, https://docs.rs/govuk-bank-holidays
7. pypdf vs X — pypdf 6.4.2 documentation, accessed December 21, 2025, https://pypdf.readthedocs.io/en/stable/meta/comparisons.html
8. How to add text to a PDF using Python? - LambdaTest Community, accessed December 21, 2025, https://community.lambdatest.com/t/how-to-add-text-to-a-pdf-using-python/34878
9. Add text to existing PDF document in Python - Stack Overflow, accessed December 21, 2025, https://stackoverflow.com/questions/6819336/add-text-to-existing-pdf-document-in-python
10. Add modified outline dictionary in pdf using Pypdf - Stack Overflow, accessed December 21, 2025, https://stackoverflow.com/questions/75897015/add-modified-outline-dictionary-in-pdf-using-pypdf
11. Quickly combine PDF files with Python | by Shawn Giese - Medium, accessed December 21, 2025, https://medium.com/@shawngiese/merge-pdf-files-with-python-30257676e97d
12. A Comparison of python libraries for PDF Data Extraction for text, images and tables, accessed December 21, 2025, https://pradeepundefned.medium.com/a-comparison-of-python-libraries-for-pdf-data-extraction-for-text-images-and-tables-c75e5dbcfef8
13. Technical Comparison — Python Libraries for Document Parsing | by chenna - Medium, accessed December 21, 2025, https://medium.com/@hchenna/technical-comparison-python-libraries-for-document-parsing-318d2c89c44e
14. adding text to a pdf using PyMuPDF - python - Stack Overflow, accessed December 21, 2025, https://stackoverflow.com/questions/63259157/adding-text-to-a-pdf-using-pymupdf
15. Mastering PDF Text with PyMuPDF's 'insert_htmlbox': What You Need to Know | Artifex, accessed December 21, 2025, https://artifex.com/blog/mastering-pdf-text-with-pymupdfs-insert-htmlbox-what-you-need-to-know
16. Text - PyMuPDF documentation, accessed December 21, 2025, https://pymupdf.readthedocs.io/en/latest/recipes-text.html
17. Which Python GUI library should you use in 2025?, accessed December 21, 2025, https://www.pythonguis.com/faq/which-python-gui-library/

18. Pyqt vs Streamlit approach for local execution - Random, accessed December 21, 2025, https://discuss.streamlit.io/t/pyqt-vs-streamlit-approach-for-local-execution/18929
19. Tutorial - Textual, accessed December 21, 2025, https://textual.textualize.io/tutorial/
20. Python Textual: Build Beautiful UIs in the Terminal, accessed December 21, 2025, https://realpython.com/python-textual/
21. Ultimate guide to the Streamlit library - Deepnote, accessed December 21, 2025, https://deepnote.com/blog/ultimate-guide-to-the-streamlit-library
22. Streamlit: Beyond Data Science Dashboards - Aceso Analytics - Articles, accessed December 21, 2025, https://articles.aceso.no/streamlit-beyond-data-science-dashboards/
23. The next frontier for Streamlit: our feature roadmap for 2023 and beyond, accessed December 21, 2025, https://discuss.streamlit.io/t/the-next-frontier-for-streamlit-our-feature-roadmap-for-2023-and-beyond/32163
24. Top Python Libraries for Timeline Charts Generation - SoftKraft, accessed December 21, 2025, https://www.softkraft.co/python-timeline-charts-generation/
25. Converting streamlit into executable file, accessed December 21, 2025, https://discuss.streamlit.io/t/converting-streamlit-into-executable-file/65451
26. Converting Streamlit application to exe file - Ploomber, accessed December 21, 2025, https://ploomber.io/blog/streamlit_exe/
27. Getting Started with Streamlit Desktop Projects - yono - xLog, accessed December 21, 2025, https://yono233.xlog.app/25_1_15_Streamlit_desktop?locale=en
28. stlite/packages/desktop/README.md at main - GitHub, accessed December 21, 2025, https://github.com/whitphx/stlite/blob/main/packages/desktop/README.md
29. ohtaman/streamlit-desktop-app - GitHub, accessed December 21, 2025, https://github.com/ohtaman/streamlit-desktop-app
30. CIVIL PROCEDURE RULES – PART 7 HOW TO START PROCEEDINGS, THE CLAIM FORM – RULE 7.5 – SERVICE OF A CLAIM FORM - White Collar Legal, accessed December 21, 2025, https://whitecollarlegalandadmin.com/civil-procedure-rules-part-7-how-to-start-proceedings-the-claim-form-rule-7-5-service-of-a-claim-form/
31. PART 7 – HOW TO START PROCEEDINGS – THE CLAIM FORM – Civil Procedure Rules, accessed December 21, 2025, https://www.justice.gov.uk/courts/procedure-rules/civil/rules/part07
32. The Civil Procedure Rules 1998 - Legislation.gov.uk, accessed December 21, 2025, https://www.legislation.gov.uk/uksi/1998/3132/rule/6.26/2008-10-01?view=plain+extent&timeline=false
33. CIVIL PROCEDURE RULES – PART 6 SERVICE OF DOCUMENTS – RULE 6.14 – SERVICE OF A CLAIM FORM - White Collar Legal, accessed December 21, 2025, https://whitecollarlegalandadmin.com/civil-procedure-rules-part-6-service-of-documents-rule-6-14-service-of-a-claim-form/

34. Changes over time for: Section 6.14 - The Civil Procedure Rules 1998, accessed December 21, 2025, https://www.legislation.gov.uk/uksi/1998/3132/rule/6.14
35. Claim forms and service by email - Mills & Reeve, accessed December 21, 2025, https://www.mills-reeve.com/publications/claim-forms-and-service-by-email/
36. Master McCloud gives clear guidance on service of a Claim Form | Medical Negligence and Personal Injury Blog | Kingsley Napley, accessed December 21, 2025, https://www.kingsleynapley.co.uk/insights/blogs/medical-negligence-and-personal-injury-blog/master-mccloud-gives-clear-guidance-on-service-of-a-claim-form
37. Build a Local AI-Powered Document Summarization Tool - DZone, accessed December 21, 2025, https://dzone.com/articles/build-a-local-ai-powered-document-summarization-tool
38. Unlocking Legal Insights: Effortless Document Summarization with OpenAI's LLM and LangChain - Velotio Technologies, accessed December 21, 2025, https://www.velotio.com/engineering-blog/unlocking-legal-insights-effortless-document-summarization-with-openais-llm-and-langchain
39. Efficient Text Summarization for Large Documents Using LangChain | Aionlinecourse, accessed December 21, 2025, https://www.aionlinecourse.com/blog/efficient-text-summarization-for-large-documents-using-langchain
40. LLM Summarization: Techniques, Metrics, and Top Models - ProjectPro, accessed December 21, 2025, https://www.projectpro.io/article/llm-summarization/1082
41. Building an AI Legal Summarizer that Reads 1000+ page Documents in seconds — with LLMs, RAG and FAISS | by Sid Gajraj | Medium, accessed December 21, 2025, https://medium.com/@sidgajraj/building-an-ai-legal-summarizer-that-reads-1000-page-documents-in-seconds-with-llms-rag-and-42e1591c536d